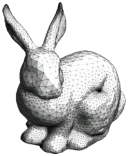
6 Algorithmische Geometrie

Navigationsgerät

Wo ist das nächste Postamt?

Paketdienste, Logistikdienstleister

In welcher Reihenfolge werden Kunden beliefert?



Geometrische Modellierung

Wie können wir geometrische Flächen darstellen?

Wir erweitern unser formales Rechnermodell, statt einer natürlichen Zahl, speichert ein Register eine **reelle Zahl**.

Wir erweitern unser formales Rechnermodell, statt einer natürlichen Zahl, speichert ein Register eine **reelle Zahl**.

Registermaschinen

Speicher: Register $c(0), c(1), c(2), \ldots \in \mathbb{R}$

Operationen: Arithmetische Operationen, Sprungbefehle

Laufzeit: eine Zeiteinheit pro Operation

6.1.1 Geometrische Graphen

geometrischer Graph $G = (V, E)$:

- V = Knotenmenge, endliche **Punktmenge** in der Ebene
- E = Kantenmenge, mit $e = \{u, v\} \in E$ zweielementige Teilmenge von V,

6.1.1 Geometrische Graphen

geometrischer Graph $G = (V, E)$:

- V = Knotenmenge, endliche **Punktmenge** in der Ebene
- E = Kantenmenge, mit $e = \{u, v\} \in E$ zweielementige Teilmenge von V, betrachten die **Strecke** $\overline{uv} = \{(1-t)u + tv \mid t \in [0, 1]\}$ als die Kante $\{u, v\}$.

6.1.1 Geometrische Graphen

geometrischer Graph $G = (V, E)$:

- V = Knotenmenge, endliche **Punktmenge** in der Ebene
- E = Kantenmenge, mit $e = \{u, v\} \in E$ zweielementige Teilmenge von V, betrachten die **Strecke** $\overline{uv} = \{(1-t)u + tv \mid t \in [0, 1]\}$ als die Kante $\{u, v\}$.
- **Zwei Kanten** $e \neq e'$ **kreuzen** sich, wenn sie einen **Schnittpunkt** haben, der nicht einer der Endpunkte von e, e' ist.

6.1.1 Geometrische Graphen

geometrischer Graph $\mathbf{G} = (\mathbf{V}, \mathbf{E})$:

- V = Knotenmenge, endliche **Punktmenge** in der Ebene
- E = Kantenmenge, mit $e = \{u, v\} \in E$ zweielementige Teilmenge von V, betrachten die **Strecke** $\overline{uv} = \{(1-t)u + tv \mid t \in [0, 1]\}$ als die Kante $\{u, v\}$.
- **Zwei Kanten** $e \neq e'$ **kreuzen** sich, wenn sie einen **Schnittpunkt** haben, der nicht einer der Endpunkte von e, e' ist.

  Annahme: Keine drei Punkte aus V liegen auf einer gemeinsamen Geraden.

6.1.1 Geometrische Graphen

geometrischer Graph $\mathbf{G = (V, E)}$:

- V = Knotenmenge, endliche **Punktmenge** in der Ebene
- E = Kantenmenge, mit $e = \{u, v\} \in E$ zweielementige Teilmenge von V, betrachten die **Strecke** $\overline{uv} = \{(1-t)u + tv \mid t \in [0, 1]\}$ als die Kante $\{u, v\}$.
- **Zwei Kanten** $e \neq e'$ **kreuzen** sich, wenn sie einen **Schnittpunkt** haben, der nicht einer der Endpunkte von e, e' ist.
  Annahme: Keine drei Punkte aus V liegen auf einer gemeinsamen Geraden.
- G ist **kreuzungsfrei** wenn keine zwei Kanten in E sich kreuzen.

6.1.1 Geometrische Graphen

geometrischer Graph $\mathbf{G} = (\mathbf{V}, \mathbf{E})$:

- V = Knotenmenge, endliche **Punktmenge** in der Ebene
- E = Kantenmenge, mit $e = \{u, v\} \in E$ zweielementige Teilmenge von V, betrachten die **Strecke** $\overline{uv} = \{(1 - t)u + tv \mid t \in [0, 1]\}$ als die Kante $\{u, v\}$.
- **Zwei Kanten** $e \neq e'$ **kreuzen** sich, wenn sie einen **Schnittpunkt** haben, der nicht einer der Endpunkte von e, e' ist.

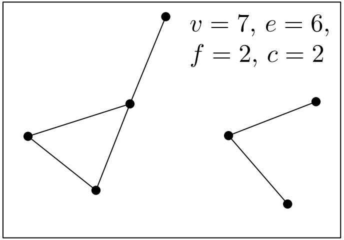
  Annahme: Keine drei Punkte aus V liegen auf einer gemeinsamen Geraden.
- G ist **kreuzungsfrei** wenn keine zwei Kanten in E sich kreuzen.
- Die Kanten eines kreuzungsfreien geometrischen Graphen unterteilt die Ebene in eine endliche Anzahl von **Flächen**.

6.1.1 Geometrische Graphen

Theorem 6.1 (Eulersche Polyederformel)

Sei $G = (V, E)$ ein kreuzungsfreier geometrischer Graph mit v Knoten, e Kanten, f Flächen und c Zusammenhangskomponenten. Es gilt $v + f = e + c + 1$.

Beispiel:

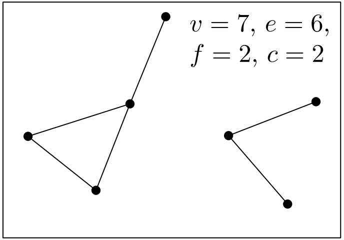


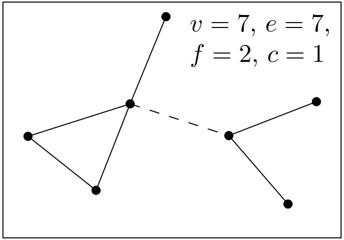
6.1.1 Geometrische Graphen

Theorem 6.1 (Eulersche Polyederformel)

Sei $G = (V, E)$ ein kreuzungsfreier geometrischer Graph mit v Knoten, e Kanten, f Flächen und c Zusammenhangskomponenten. Es gilt $v + f = e + c + 1$.

Beispiel:



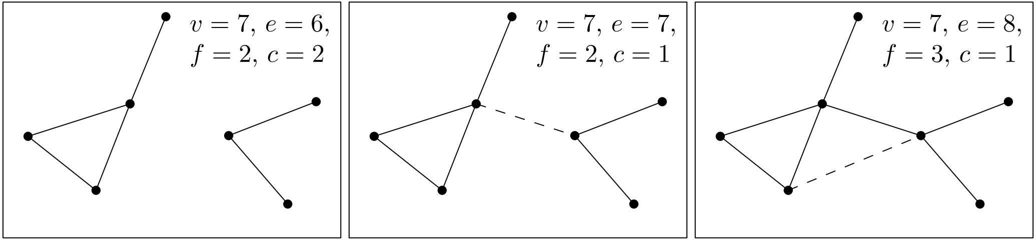


6.1.1 Geometrische Graphen

Theorem 6.1 (Eulersche Polyederformel)

Sei $G = (V, E)$ ein kreuzungsfreier geometrischer Graph mit v Knoten, e Kanten, f Flächen und c Zusammenhangskomponenten. Es gilt $v + f = e + c + 1$.

Beispiel:



6.1.1 Geometrische Graphen

Theorem 6.1

Sei $G = (V, E)$ ein kreuzungsfreier geometrischer Graph mit v Knoten, e Kanten, f Flächen und c Zusammenhangskomponenten. Es gilt $v + f = e + c + 1$.

Beweis: Per Induktion über die Menge an Knoten und Kanten.

6.1.1 Geometrische Graphen

Theorem 6.1

Sei $G = (V, E)$ ein kreuzungsfreier geometrischer Graph mit v Knoten, e Kanten, f Flächen und c Zusammenhangskomponenten. Es gilt $v + f = e + c + 1$.

Beweis: Per Induktion über die Menge an Knoten und Kanten.

Induktionsanfang: Leere Menge mit $v = 0, f = 1, e = 0, c = 0$.

6.1.1 Geometrische Graphen

Theorem 6.1

Sei $G = (V, E)$ ein kreuzungsfreier geometrischer Graph mit v Knoten, e Kanten, f Flächen und c Zusammenhangskomponenten. Es gilt $v + f = e + c + 1$.

Beweis: Per Induktion über die Menge an Knoten und Kanten.

Induktionsanfang: Leere Menge mit $v = 0, f = 1, e = 0, c = 0$.

Induktionsschritt: Angenommen, die Aussage gilt für G. Füge einen Knoten oder eine Kante hinzu und erhalte G' mit v', e', f', c' Knoten, Kanten, Flächen, Komponenten.

6.1.1 Geometrische Graphen

Theorem 6.1

Sei $G = (V, E)$ ein kreuzungsfreier geometrischer Graph mit v Knoten, e Kanten, f Flächen und c Zusammenhangskomponenten. Es gilt $v + f = e + c + 1$.

Beweis: Per Induktion über die Menge an Knoten und Kanten.

Induktionsanfang: Leere Menge mit $v = 0, f = 1, e = 0, c = 0$.

Induktionsschritt: Angenommen, die Aussage gilt für G. Füge einen Knoten oder eine Kante hinzu und erhalte G' mit v', e', f', c' Knoten, Kanten, Flächen, Komponenten.

Fall 1: Füge einen isolierten **Knoten** hinzu $\implies v' = v + 1, c' = c + 1, e' = e, f' = f$

6.1.1 Geometrische Graphen

Theorem 6.1

Sei $G = (V, E)$ ein kreuzungsfreier geometrischer Graph mit v Knoten, e Kanten, f Flächen und c Zusammenhangskomponenten. Es gilt $v + f = e + c + 1$.

Beweis: Per Induktion über die Menge an Knoten und Kanten.

Induktionsanfang: Leere Menge mit $v = 0, f = 1, e = 0, c = 0$.

Induktionsschritt: Angenommen, die Aussage gilt für G. Füge einen Knoten oder eine Kante hinzu und erhalte G' mit v', e', f', c' Knoten, Kanten, Flächen, Komponenten.

Fall 1: Füge einen isolierten **Knoten** hinzu $\implies v' = v + 1, c' = c + 1, e' = e, f' = f$

Fall 2: Verbinde zwei Knoten in G mit neuer **Kante** $h \implies v' = v$ und $e' = e + 1$

Theorem 6.1

Sei $G = (V, E)$ ein kreuzungsfreier geometrischer Graph mit v Knoten, e Kanten, f Flächen und c Zusammenhangskomponenten. Es gilt $v + f = e + c + 1$.

Beweis: Per Induktion über die Menge an Knoten und Kanten.

Induktionsanfang: Leere Menge mit $v = 0, f = 1, e = 0, c = 0$.

Induktionsschritt: Angenommen, die Aussage gilt für G. Füge einen Knoten oder eine Kante hinzu und erhalte G' mit v', e', f', c' Knoten, Kanten, Flächen, Komponenten.

Fall 1: Füge einen isolierten **Knoten** hinzu $\implies v' = v + 1, c' = c + 1, e' = e, f' = f$

Fall 2: Verbinde zwei Knoten in G mit neuer **Kante** $h \implies v' = v$ und $e' = e + 1$

(a) Kante h **schließt einen Kreis** in G, also teilt eine Fläche $\implies f' = f + 1, c' = c$

6.1.1 Geometrische Graphen

Theorem 6.1

Sei $G = (V, E)$ ein kreuzungsfreier geometrischer Graph mit v Knoten, e Kanten, f Flächen und c Zusammenhangskomponenten. Es gilt $v + f = e + c + 1$.

Beweis: Per Induktion über die Menge an Knoten und Kanten.

Induktionsanfang: Leere Menge mit $v = 0$, $f = 1$, $e = 0$, $c = 0$.

Induktionsschritt: Angenommen, die Aussage gilt für G. Füge einen Knoten oder eine Kante hinzu und erhalte G' mit v', e', f', c' Knoten, Kanten, Flächen, Komponenten.

Fall 1: Füge einen isolierten **Knoten** hinzu $\implies v' = v + 1$, $c' = c + 1$, $e' = e$, $f' = f$

Fall 2: Verbinde zwei Knoten in G mit neuer **Kante** $h \implies v' = v$ und $e' = e + 1$

- **(a)** Kante h **schließt einen Kreis** in G, also teilt eine Fläche $\implies f' = f + 1$, $c' = c$
- **(b)** Sonst, h verbindet zwei Komponenten $\implies f' = f$, $c' = c - 1$

$\square$

6.1.1 Geometrische Graphen

Theorem 6.2

Sei G ein kreuzungsfreier geometrischer Graph mit v Knoten, e Kanten und f Flächen, dann ist $e \in O(v)$ und $f \in O(v)$.

Theorem 6.2

Sei G ein kreuzungsfreier geometrischer Graph mit v Knoten, e Kanten und f Flächen, dann ist $e \in O(v)$ und $f \in O(v)$.

Beweis: Sei G zusammenhängend und sei $e \geq 3$.

Theorem 6.2

Sei G ein kreuzungsfreier geometrischer Graph mit v Knoten, e Kanten und f Flächen, dann ist $e \in O(v)$ und $f \in O(v)$.

Beweis: Sei G zusammenhängend und sei $e \geq 3$. Sei d_i **Anzahl der Kanten, die inzident zur i-ten Fläche** sind (für eine beliebige Reihenfolge der Flächen).

Theorem 6.2

Sei G ein kreuzungsfreier geometrischer Graph mit v Knoten, e Kanten und f Flächen, dann ist $e \in O(v)$ und $f \in O(v)$.

Beweis: Sei G zusammenhängend und sei $e \geq 3$. Sei d_i **Anzahl der Kanten, die inzident zur i-ten Fläche** sind (für eine beliebige Reihenfolge der Flächen).

Da $d_i \geq 3$ und jede Kante zu maximal **zwei Flächen** inzident ist, gilt

$$3f \leq \sum_{i=1}^{f} d_i \leq 2e$$

Theorem 6.2

Sei G ein kreuzungsfreier geometrischer Graph mit v Knoten, e Kanten und f Flächen, dann ist $e \in O(v)$ und $f \in O(v)$.

Beweis: Sei G zusammenhängend und sei $e \geq 3$. Sei d_i **Anzahl der Kanten, die inzident zur i-ten Fläche** sind (für eine beliebige Reihenfolge der Flächen).

Da $d_i \geq 3$ und jede Kante zu maximal **zwei Flächen** inzident ist, gilt

$$3f \leq \sum_{i=1}^{f} d_i \leq 2e$$

Daraus folgt $f \leq \frac{2}{3}e$.

Theorem 6.2

Sei G ein kreuzungsfreier geometrischer Graph mit v Knoten, e Kanten und f Flächen, dann ist $e \in O(v)$ und $f \in O(v)$.

Beweis: Sei G zusammenhängend und sei $e \geq 3$. Sei d_i **Anzahl der Kanten, die inzident zur i-ten Fläche** sind (für eine beliebige Reihenfolge der Flächen).

Da $d_i \geq 3$ und jede Kante zu maximal **zwei Flächen** inzident ist, gilt

$$3f \leq \sum_{i=1}^{f} d_i \leq 2e$$

Daraus folgt $f \leq \frac{2}{3}e$. Einsetzen in **Theorem 6.1** mit $c = 1$ ergibt

$$e = f + v - 2$$

Theorem 6.2

Sei G ein kreuzungsfreier geometrischer Graph mit v Knoten, e Kanten und f Flächen, dann ist $e \in O(v)$ und $f \in O(v)$.

Beweis: Sei G zusammenhängend und sei $e \geq 3$. Sei d_i **Anzahl der Kanten, die inzident zur i-ten Fläche** sind (für eine beliebige Reihenfolge der Flächen).

Da $d_i \geq 3$ und jede Kante zu maximal **zwei Flächen** inzident ist, gilt

$$3f \leq \sum_{i=1}^{f} d_i \leq 2e$$

Daraus folgt $f \leq \frac{2}{3}e$. Einsetzen in **Theorem 6.1** mit $c = 1$ ergibt

$$e = f + v - 2 \leq v + \frac{2}{3}e - 2$$

Theorem 6.2

Sei G ein kreuzungsfreier geometrischer Graph mit v Knoten, e Kanten und f Flächen, dann ist $e \in O(v)$ und $f \in O(v)$.

Beweis: Sei G zusammenhängend und sei $e \geq 3$. Sei d_i **Anzahl der Kanten, die inzident zur i-ten Fläche** sind (für eine beliebige Reihenfolge der Flächen).

Da $d_i \geq 3$ und jede Kante zu maximal **zwei Flächen** inzident ist, gilt

$$3f \leq \sum_{i=1}^{f} d_i \leq 2e$$

Daraus folgt $f \leq \frac{2}{3}e$. Einsetzen in **Theorem 6.1** mit $c = 1$ ergibt

$$e = f + v - 2 \leq v + \frac{2}{3}e - 2$$

Durch Umformung folgt $e \in O(v)$.

Theorem 6.2

Sei G ein kreuzungsfreier geometrischer Graph mit v Knoten, e Kanten und f Flächen, dann ist $e \in O(v)$ und $f \in O(v)$.

Beweis: Sei G zusammenhängend und sei $e \geq 3$. Sei d_i **Anzahl der Kanten, die inzident zur i-ten Fläche** sind (für eine beliebige Reihenfolge der Flächen).

Da $d_i \geq 3$ und jede Kante zu maximal **zwei Flächen** inzident ist, gilt

$$3f \leq \sum_{i=1}^{f} d_i \leq 2e$$

Daraus folgt $f \leq \frac{2}{3}e$. Einsetzen in **Theorem 6.1** mit $c = 1$ ergibt

$$e = f + v - 2 \leq v + \frac{2}{3}e - 2$$

Durch Umformung folgt $e \in O(v)$. Da $f \leq \frac{2}{3}e$ folgt direkt auch $f \in O(v)$.

Theorem 6.2

Sei G ein kreuzungsfreier geometrischer Graph mit v Knoten, e Kanten und f Flächen, dann ist $e \in O(v)$ und $f \in O(v)$.

Beweis: Sei G zusammenhängend und sei $e \geq 3$. Sei d_i **Anzahl der Kanten, die inzident zur i-ten Fläche** sind (für eine beliebige Reihenfolge der Flächen).

Da $d_i \geq 3$ und jede Kante zu maximal **zwei Flächen** inzident ist, gilt

$$3f \leq \sum_{i=1}^{f} d_i \leq 2e$$

Daraus folgt $f \leq \frac{2}{3}e$. Einsetzen in **Theorem 6.1** mit $c = 1$ ergibt

$$e = f + v - 2 \leq v + \frac{2}{3}e - 2$$

Durch Umformung folgt $e \in O(v)$. Da $f \leq \frac{2}{3}e$ folgt direkt auch $f \in O(v)$.

Falls G nicht zusammenhängend, betrachte Zusammenhangskomponenten einzeln. $\square$

6.1.1 Geometrische Graphen

Definition (Triangulation)

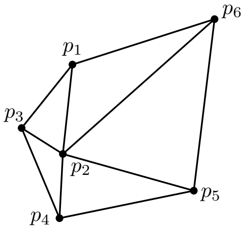
Eine **Triangulation** einer Menge von Punkten S in der Ebene ist ein kreuzungsfreier geometrischer Graph mit Knotenmenge S und mit **inklusions-maximaler Kantenmenge** E. Das heisst, es kann keine Kante zu E hinzugefügt werden ohne eine Kreuzung zu erzeugen.

6.1.1 Geometrische Graphen

Definition (Triangulation)

Eine **Triangulation** einer Menge von Punkten S in der Ebene ist ein kreuzungsfreier geometrischer Graph mit Knotenmenge S und mit **inklusions-maximaler Kantenmenge** E. Das heisst, es kann keine Kante zu E hinzugefügt werden ohne eine Kreuzung zu erzeugen.

- Flächen des Graphen sind **Dreiecke**, mit Ausnahme der **äußeren Fläche**.
- Triangulation für eine feste Punktmenge **nicht eindeutig**.

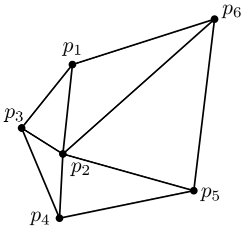


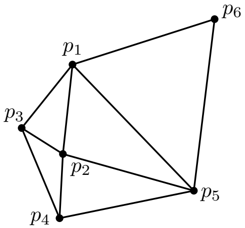
6.1.1 Geometrische Graphen

Definition (Triangulation)

Eine **Triangulation** einer Menge von Punkten S in der Ebene ist ein kreuzungsfreier geometrischer Graph mit Knotenmenge S und mit **inklusions-maximaler Kantenmenge** E. Das heisst, es kann keine Kante zu E hinzugefügt werden ohne eine Kreuzung zu erzeugen.

- Flächen des Graphen sind **Dreiecke**, mit Ausnahme der **äußeren Fläche**.
- Triangulation für eine feste Punktmenge **nicht eindeutig**.





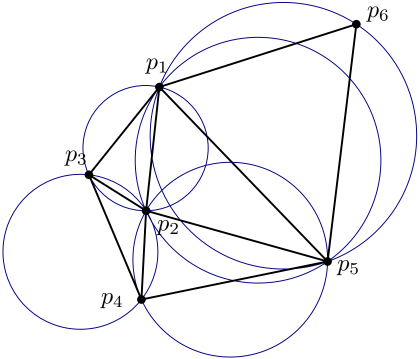
Im Folgenden machen wir die folgende Annahme:

Definition (Allgemeine Lage)

Eine endliche Punktmenge $S \subseteq \mathbb{R}^2$ ist in **allgemeiner Lage**, wenn (i) keine **vier Punkte** aus S auf einem gemeinsamen **Kreis** liegen, und (ii) keine **drei Punkte** aus S auf einer gemeinsamen **Geraden** liegen.

Definition (Delaunay-Triangulation)

Eine Triangulation einer Menge von Punkten S in der Ebene ist eine **Delaunay-Triangulation** wenn für jedes Dreieck der Triangulation gilt, dass sein Umkreis keine Punkte aus S in seinem Inneren enthält.



Definition (Delaunay-Triangulation)

Eine Triangulation einer Menge von Punkten S in der Ebene ist eine **Delaunay-Triangulation** wenn für jedes Dreieck der Triangulation gilt, dass sein Umkreis keine Punkte aus S in seinem Inneren enthält.

Wir wollen zeigen, dass die Delaunay-Triangulation von einer Punktmenge in allgemeiner Lage **eindeutig** ist. Dafür betrachten wir zunächst folgende **unabhängige Definition**.

Definition (Delaunay-Kante)

Eine Kante (p, q) zwischen zwei Punkten der Menge S ist eine **Delaunay-Kante** genau dann wenn **ein Kreis existiert**, der die zwei Endpunkte p und q auf dem Rand hat und keine weiteren Punkte von S in seinem Inneren enthält.

Lemma 6.6

Sei S eine Menge von Punkten in allgemeiner Lage in der Ebene und seien e und e' zwei verschiedene **Delaunay-Kanten** von S, dann können sich e und e' **nicht kreuzen**.

Lemma 6.6

Sei S eine Menge von Punkten in allgemeiner Lage in der Ebene und seien e und e' zwei verschiedene **Delaunay-Kanten** von S, dann können sich e und e' **nicht kreuzen**.

Beweis: Durch **Widerspruch**.

Lemma 6.6

Sei S eine Menge von Punkten in allgemeiner Lage in der Ebene und seien e und e' zwei verschiedene **Delaunay-Kanten** von S, dann können sich e und e' **nicht kreuzen**.

Beweis: Durch **Widerspruch**.

Angenommen, $e = \{u, v\}$ und $e' = \{u', v'\}$ kreuzen sich.

Lemma 6.6

Sei S eine Menge von Punkten in allgemeiner Lage in der Ebene und seien e und e' zwei verschiedene **Delaunay-Kanten** von S, dann können sich e und e' **nicht kreuzen**.

Beweis: Durch **Widerspruch**.

Angenommen, $e = \{u, v\}$ und $e' = \{u', v'\}$ kreuzen sich.

Da e Delaunay-Kante ist, **existiert ein Kreis** C, der u und v auf dem Rand, aber weder u' noch v' im Inneren enthält.

Lemma 6.6

Sei S eine Menge von Punkten in allgemeiner Lage in der Ebene und seien e und e' zwei verschiedene **Delaunay-Kanten** von S, dann können sich e und e' **nicht kreuzen**.

Beweis: Durch **Widerspruch**.

Angenommen, $e = \{u, v\}$ und $e' = \{u', v'\}$ kreuzen sich.

Da e Delaunay-Kante ist, **existiert ein Kreis** C, der u und v auf dem Rand, aber weder u' noch v' im Inneren enthält.

Symmetrisch gibt es einen Kreis C' für e', der u' und v' auf dem Rand, aber weder u noch v im Inneren enthält.

6.1.1 Geometrische Graphen

Lemma 6.6

Sei S eine Menge von Punkten in allgemeiner Lage in der Ebene und seien e und e' zwei verschiedene **Delaunay-Kanten** von S, dann können sich e und e' **nicht kreuzen**.

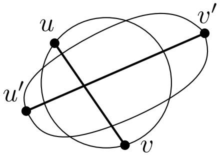
Beweis: Durch **Widerspruch**.

Angenommen, $e = \{u, v\}$ und $e' = \{u', v'\}$ kreuzen sich.

Da e Delaunay-Kante ist, **existiert ein Kreis** C, der u und v auf dem Rand, aber weder u' noch v' im Inneren enthält.

Symmetrisch gibt es einen Kreis C' für e', der u' und v' auf dem Rand, aber weder u noch v im Inneren enthält.

Dann müssten sich C und C' **an mehr als zwei Punkten schneiden**, obwohl $C \neq C'$.



□

Theorem 6.7

Die Menge der Kanten der Delaunay-Triangulation einer Menge S von Punkten in allgemeiner Lage ist genau die Menge der Delaunay-Kanten von S.

Beweis: Wir zeigen Inklusion der Kantenmengen in beiden Richtungen.

Theorem 6.7

Die Menge der Kanten der Delaunay-Triangulation einer Menge S von Punkten in allgemeiner Lage ist genau die Menge der Delaunay-Kanten von S.

Beweis: Wir zeigen Inklusion der Kantenmengen in beiden Richtungen.

Sei T die Delaunay-Triangulation von S.

Theorem 6.7

Die Menge der Kanten der Delaunay-Triangulation einer Menge S von Punkten in allgemeiner Lage ist genau die Menge der Delaunay-Kanten von S.

Beweis: Wir zeigen Inklusion der Kantenmengen in beiden Richtungen.

Sei T die Delaunay-Triangulation von S.

$(\Rightarrow)$ Für jedes **Dreieck** von T gilt, dass sein Umkreis keinen Punkt aus S enthält.

Theorem 6.7

Die Menge der Kanten der Delaunay-Triangulation einer Menge S von Punkten in allgemeiner Lage ist genau die Menge der Delaunay-Kanten von S.

Beweis: Wir zeigen Inklusion der Kantenmengen in beiden Richtungen.

Sei T die Delaunay-Triangulation von S.

($\Rightarrow$) Für jedes **Dreieck** von T gilt, dass sein Umkreis keinen Punkt aus S enthält. Jede **Kante** von T kommt im mindestens einem Dreieck vor, ist also Delaunay-Kante.

Theorem 6.7

Die Menge der Kanten der Delaunay-Triangulation einer Menge S von Punkten in allgemeiner Lage ist genau die Menge der Delaunay-Kanten von S.

Beweis: Wir zeigen Inklusion der Kantenmengen in beiden Richtungen.

Sei T die Delaunay-Triangulation von S.

($\Rightarrow$) Für jedes **Dreieck** von T gilt, dass sein Umkreis keinen Punkt aus S enthält. Jede **Kante** von T kommt im mindestens einem Dreieck vor, ist also Delaunay-Kante.

($\Leftarrow$) (Durch **Widerspruch**) Angenommmen, es existiert eine **Delaunay-Kante** e von S, die **nicht** in T enthalten ist.

Theorem 6.7

Die Menge der Kanten der Delaunay-Triangulation einer Menge S von Punkten in allgemeiner Lage ist genau die Menge der Delaunay-Kanten von S.

Beweis: Wir zeigen Inklusion der Kantenmengen in beiden Richtungen.

Sei T die Delaunay-Triangulation von S.

($\Rightarrow$) Für jedes **Dreieck** von T gilt, dass sein Umkreis keinen Punkt aus S enthält. Jede **Kante** von T kommt im mindestens einem Dreieck vor, ist also Delaunay-Kante.

($\Leftarrow$) (Durch **Widerspruch**) Angenommmen, es existiert eine **Delaunay-Kante** e von S, die **nicht** in T enthalten ist. Da e laut Lemma 6.6 keine andere Delaunay-Kante kreuzen kann, können wir sie zu T hinzufügen und erhalten eine Triangulation von S.

Theorem 6.7

Die Menge der Kanten der Delaunay-Triangulation einer Menge S von Punkten in allgemeiner Lage ist genau die Menge der Delaunay-Kanten von S.

Beweis: Wir zeigen Inklusion der Kantenmengen in beiden Richtungen.

Sei T die Delaunay-Triangulation von S.

($\Rightarrow$) Für jedes **Dreieck** von T gilt, dass sein Umkreis keinen Punkt aus S enthält. Jede **Kante** von T kommt im mindestens einem Dreieck vor, ist also Delaunay-Kante.

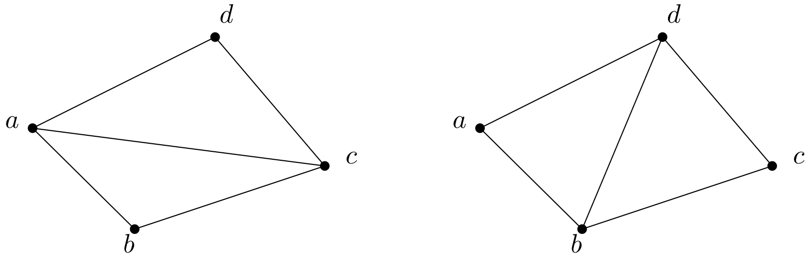
($\Leftarrow$) (Durch **Widerspruch**) Angenommmen, es existiert eine **Delaunay-Kante** e von S, die **nicht** in T enthalten ist. Da e laut Lemma 6.6 keine andere Delaunay-Kante kreuzen kann, können wir sie zu T hinzufügen und erhalten eine Triangulation von S.

Also ist die Kantenmenge von T **nicht inklusions-maximal**, also ist T keine Triangulation. $\square$

Definition (Flip)

Angenommen, (a, b, c) und (a, c, d) sind benachbarte Dreiecke einer Triangulation T. Wir bezeichnen die Kanten (a, c) und (b, d) als **Diagonalen** des Vierecks (a, b, c, d) sofern sie innerhalb des Vierecks verlaufen. Ein **Flip** ersetzt (a, c) durch (b, d) in T. Der Flip ist **zulässig**, wenn beide Kanten Diagonalen sind. In diesem Fall nennen wir (b, d) die **Flip-Diagonale** von (a, c) in T.

Beispiel:

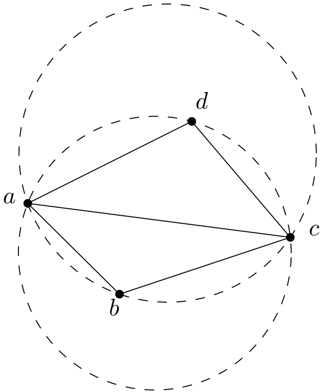


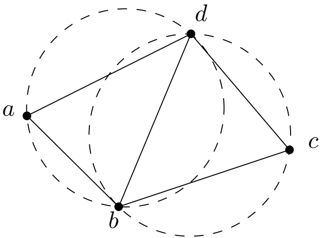
6.1.2 Greedy-Algorithmus von Fortune

Definition (Delaunay-Flip)

Sei (a, c) die gemeinsame Kante zweier Dreiecke (a, b, c) und (a, c, d) einer Triangulation, sodass der Flip von (a, c) zu (b, d) zulässig ist. Ist d im Umkreis von (a, b, c) enthalten, so sprechen wir von einem **Delaunay-Flip**.

Beispiel:





6.1.2 Greedy-Algorithmus von Fortune

```
GreedyFlips(Triangulation T)

while (∃ Kante e in T, die einen Delaunay-Flip erlaubt) {
        Ersetze e durch die entsprechende Flip-Diagonale in T.
}
```

6.1.2 Greedy-Algorithmus von Fortune

```
GreedyFlips(Triangulation T)
    while (∃ Kante e in T, die einen Delaunay-Flip erlaubt) {
        Ersetze e durch die entsprechende Flip-Diagonale in T.
    }
```

- Uns geht es zunächst nicht darum zu zeigen, wie man den Greedy-Algorithmus effizient implementieren kann

6.1.2 Greedy-Algorithmus von Fortune

```
GreedyFlips(Triangulation T)

while (∃ Kante e in T, die einen Delaunay-Flip erlaubt) {
        Ersetze e durch die entsprechende Flip-Diagonale in T.
}
```

- Uns geht es zunächst nicht darum zu zeigen, wie man den Greedy-Algorithmus effizient implementieren kann
- Zunächst wollen wir zeigen, dass der Algorithmus überhaupt **terminiert** und damit das **korrekte Ergebnis** liefert.

Definition (Konfliktfunktion)

Sei $C(a, b, c)$ der Umkreis des Dreiecks (a, b, c). Sei T eine Triangulation von S. Wir definieren die Funktion Φ auf der Menge der Triangulationen von S.

$$\Phi(T) = \sum_{(a,b,c) \text{ Dreieck in } T} |\{p \in S \mid p \in C(a, b, c)\}|$$

Wenn ein Punkt aus S im Umkreis eines Dreiecks von T enthalten ist, dann sagen wir dieser Punkt ist mit dem Dreieck im Konflikt. $\Phi(T)$ ist die Anzahl der Konflikte in T.

6.1.2 Greedy-Algorithmus von Fortune

Definition (Konfliktfunktion)

Sei $C(a,b,c)$ der Umkreis des Dreiecks (a,b,c). Sei T eine Triangulation von S. Wir definieren die Funktion Φ auf der Menge der Triangulationen von S.

$$\Phi(T) = \sum_{(a,b,c) \text{ Dreieck in } T} |\{p \in S \mid p \in C(a,b,c)\}|$$

Wenn ein Punkt aus S im Umkreis eines Dreiecks von T enthalten ist, dann sagen wir dieser Punkt ist mit dem Dreieck im Konflikt. $\Phi(T)$ ist die Anzahl der Konflikte in T.

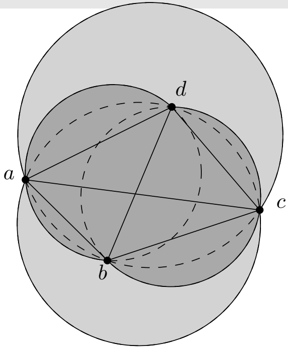
Idee: Wir wollen zeigen, dass sich im Laufe des Greedy-Algorithmus die Anzahl der Konflikte mit jedem Delaunay-Flip verringert.

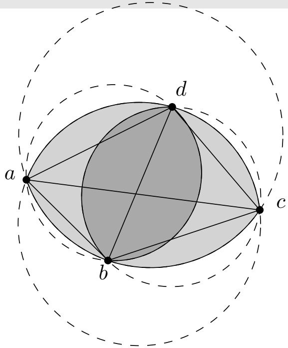
Lemma 6.11 (Ohne Beweis)

Betrachte einen Delaunay-Flip im Viereck (a, b, c, d) der die Diagonale (a, c) mit der Diagonale (b, d) ersetzt. Es gilt

(i) $C(a, b, d) \cup C(b, c, d) \subseteq C(a, b, c) \cup C(a, c, d)$

(ii) $C(a, b, d) \cap C(b, c, d) \subseteq C(a, b, c) \cap C(a, c, d)$





Lemma 6.12

Sei T' eine Triangulation, die man aus T durch einen Delaunay-Flip erhält. Dann gilt $\Phi(T') \leq \Phi(T) - 2$.

6.1.2 Greedy-Algorithmus von Fortune

Lemma 6.12

Sei T' eine Triangulation, die man aus T durch einen Delaunay-Flip erhält. Dann gilt $\Phi(T') \leq \Phi(T) - 2$.

Beweis: Betrachte einen Delaunay-Flip. Zwei Konflikte der direkt involvierten Punkte und Dreiecke werden aufgelöst (d mit (a, b, c) und b mit (a, c, d)).

Lemma 6.12

Sei T' eine Triangulation, die man aus T durch einen Delaunay-Flip erhält. Dann gilt $\Phi(T') \leq \Phi(T) - 2$.

Beweis: Betrachte einen Delaunay-Flip. Zwei Konflikte der direkt involvierten Punkte und Dreiecke werden aufgelöst (d mit (a, b, c) und b mit (a, c, d)).

- Aus Lemma 6.11 (i) folgt, dass ein Punkt aus S, der mit einem der neuen Dreiecke im Konflikt ist, auch im Konflikt mit **mindestens** einem der alten Dreiecke im Konflikt war.

6.1.2 Greedy-Algorithmus von Fortune

Lemma 6.12

Sei T' eine Triangulation, die man aus T durch einen Delaunay-Flip erhält. Dann gilt $\Phi(T') \leq \Phi(T) - 2$.

Beweis: Betrachte einen Delaunay-Flip. Zwei Konflikte der direkt involvierten Punkte und Dreiecke werden aufgelöst (d mit (a, b, c) und b mit (a, c, d)).

- Aus Lemma 6.11 (i) folgt, dass ein Punkt aus S, der mit einem der neuen Dreiecke im Konflikt ist, auch im Konflikt mit **mindestens** einem der alten Dreiecke im Konflikt war.
- Aus Lemma 6.11 (ii) folgt, dass ein Punkt, der mit **beiden** neuen Dreiecken im Konflikt ist, auch mit **beiden** alten Dreiecken im Konflikt war.

Lemma 6.12

Sei T' eine Triangulation, die man aus T durch einen Delaunay-Flip erhält. Dann gilt $\Phi(T') \leq \Phi(T) - 2$.

Beweis: Betrachte einen Delaunay-Flip. Zwei Konflikte der direkt involvierten Punkte und Dreiecke werden aufgelöst (d mit (a, b, c) und b mit (a, c, d)).

- Aus Lemma 6.11 (i) folgt, dass ein Punkt aus S, der mit einem der neuen Dreiecke im Konflikt ist, auch im Konflikt mit **mindestens** einem der alten Dreiecke im Konflikt war.
- Aus Lemma 6.11 (ii) folgt, dass ein Punkt, der mit **beiden** neuen Dreiecken im Konflikt ist, auch mit **beiden** alten Dreiecken im Konflikt war.
- Alle anderen Konflikte bleiben unberührt.

Lemma 6.12

Sei T' eine Triangulation, die man aus T durch einen Delaunay-Flip erhält. Dann gilt $\Phi(T') \leq \Phi(T) - 2$.

Beweis: Betrachte einen Delaunay-Flip. Zwei Konflikte der direkt involvierten Punkte und Dreiecke werden aufgelöst (d mit (a, b, c) und b mit (a, c, d)).

- Aus Lemma 6.11 (i) folgt, dass ein Punkt aus S, der mit einem der neuen Dreiecke im Konflikt ist, auch im Konflikt mit **mindestens** einem der alten Dreiecke im Konflikt war.
- Aus Lemma 6.11 (ii) folgt, dass ein Punkt, der mit **beiden** neuen Dreiecken im Konflikt ist, auch mit **beiden** alten Dreiecken im Konflikt war.
- Alle anderen Konflikte bleiben unberührt.

Die Gesamtzahl der Konflikte reduziert sich also um mindestens zwei. $\square$

6.1.2 Greedy-Algorithmus von Fortune

Lemma 6.13

Sei T eine Triangulation einer Punktmenge S mit $\Phi(T) > 0$, dann gibt es zwei benachbarte Dreiecke in T die einen Delaunay-Flip erlauben.

6.1.2 Greedy-Algorithmus von Fortune

Lemma 6.13

Sei T eine Triangulation einer Punktmenge S mit $\Phi(T) > 0$, dann gibt es zwei benachbarte Dreiecke in T die einen Delaunay-Flip erlauben.

Beweis: Wähle einen Konflikt, der den Abstand zwischen Punkt und Dreieck **minimiert**.

6.1.2 Greedy-Algorithmus von Fortune

Lemma 6.13

Sei T eine Triangulation einer Punktmenge S mit $\Phi(T) > 0$, dann gibt es zwei benachbarte Dreiecke in T die einen Delaunay-Flip erlauben.

Beweis: Wähle einen Konflikt, der den Abstand zwischen Punkt und Dreieck **minimiert**. Sei d der Punkt und (a, b, c) das Dreieck. Falls d mit einer Kante von (a, b, c) ein Dreieck in T bildet, dann **existiert** ein Delaunay-Flip.

Lemma 6.13

Sei T eine Triangulation einer Punktmenge S mit $\Phi(T) > 0$, dann gibt es zwei benachbarte Dreiecke in T die einen Delaunay-Flip erlauben.

Beweis: Wähle einen Konflikt, der den Abstand zwischen Punkt und Dreieck **minimiert**. Sei d der Punkt und (a, b, c) das Dreieck. Falls d mit einer Kante von (a, b, c) ein Dreieck in T bildet, dann **existiert** ein Delaunay-Flip. Ansonsten, sei (a, c) die zu d nächste Kante. Die Kante (a, c) bildet mit einem Punkt $e \neq d$ ein Dreieck in T.

Lemma 6.13

Sei T eine Triangulation einer Punktmenge S mit $\Phi(T) > 0$, dann gibt es zwei benachbarte Dreiecke in T die einen Delaunay-Flip erlauben.

Beweis: Wähle einen Konflikt, der den Abstand zwischen Punkt und Dreieck **minimiert**. Sei d der Punkt und (a, b, c) das Dreieck. Falls d mit einer Kante von (a, b, c) ein Dreieck in T bildet, dann **existiert** ein Delaunay-Flip. Ansonsten, sei (a, c) die zu d nächste Kante. Die Kante (a, c) bildet mit einem Punkt $e \neq d$ ein Dreieck in T. Betrachte **zwei Fälle**:

(a) $e \in C(a, b, c) \implies$ **existiert** Delaunay-Flip (a, c) zu (e, b)

6.1.2 Greedy-Algorithmus von Fortune

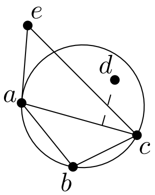
Lemma 6.13

Sei T eine Triangulation einer Punktmenge S mit $\Phi(T) > 0$, dann gibt es zwei benachbarte Dreiecke in T die einen Delaunay-Flip erlauben.

Beweis: Wähle einen Konflikt, der den Abstand zwischen Punkt und Dreieck **minimiert**. Sei d der Punkt und (a, b, c) das Dreieck. Falls d mit einer Kante von (a, b, c) ein Dreieck in T bildet, dann **existiert** ein Delaunay-Flip. Ansonsten, sei (a, c) die zu d nächste Kante. Die Kante (a, c) bildet mit einem Punkt $e \neq d$ ein Dreieck in T. Betrachte **zwei Fälle**:

(a) $e \in C(a, b, c) \implies$ **existiert** Delaunay-Flip (a, c) zu (e, b)

(b) Ansonsten, $e \notin C(a, b, c)$. Dann steht d im Konflikt mit (e, a, c). Weiterhin ist (e, a, c) **näher** an d, da die Strecke von d zu dem Punkt mit kleinstem Abstand das Dreieck (a, c, e) kreuzt. Das **widerspricht** der Wahl von d und (a, b, c). $\square$



Lemma 6.14

Sei T eine Triangulation einer Menge S von n Punkten. Der Greedy-Algorithmus terminiert und führt maximal $O(n^2)$ Delaunay-Flips durch.

Lemma 6.14

Sei T eine Triangulation einer Menge S von n Punkten. Der Greedy-Algorithmus terminiert und führt maximal $O(n^2)$ Delaunay-Flips durch.

Beweis: Aus Lemma 6.13 folgt, dass der Algorithmus terminiert.

Lemma 6.14

Sei T eine Triangulation einer Menge S von n Punkten. Der Greedy-Algorithmus terminiert und führt maximal $O(n^2)$ Delaunay-Flips durch.

Beweis: Aus Lemma 6.13 folgt, dass der Algorithmus terminiert.
Aus Theorem 6.7 folgt, dass die Anzahl der Dreiecke in T in $O(n)$ ist. Daher kann es höchstens $O(n^2)$ viele Konflikte geben, da jedes Paar von Dreieck und Punkt nur einen Konflikt erzeugen kann.

6.1.2 Greedy-Algorithmus von Fortune

Lemma 6.14

Sei T eine Triangulation einer Menge S von n Punkten. Der Greedy-Algorithmus terminiert und führt maximal $O(n^2)$ Delaunay-Flips durch.

Beweis: Aus Lemma 6.13 folgt, dass der Algorithmus terminiert.
Aus Theorem 6.7 folgt, dass die Anzahl der Dreiecke in T in $O(n)$ ist. Daher kann es höchstens $O(n^2)$ viele Konflikte geben, da jedes Paar von Dreieck und Punkt nur einen Konflikt erzeugen kann.
Aus Lemma 6.12 folgt, dass sich die Anzahl der Konflikte mit jedem Delaunay-Flip um mindestens zwei verringert. $\square$

6.1.2 Greedy-Algorithmus von Fortune

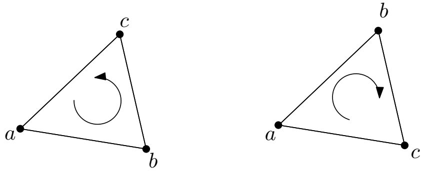
Lemma 6.14

Sei T eine Triangulation einer Menge S von n Punkten. Der Greedy-Algorithmus terminiert und führt maximal $O(n^2)$ Delaunay-Flips durch.

Beweis: Aus Lemma 6.13 folgt, dass der Algorithmus terminiert.
Aus Theorem 6.7 folgt, dass die Anzahl der Dreiecke in T in $O(n)$ ist. Daher kann es höchstens $O(n^2)$ viele Konflikte geben, da jedes Paar von Dreieck und Punkt nur einen Konflikt erzeugen kann.
Aus Lemma 6.12 folgt, dass sich die Anzahl der Konflikte mit jedem Delaunay-Flip um mindestens zwei verringert. □

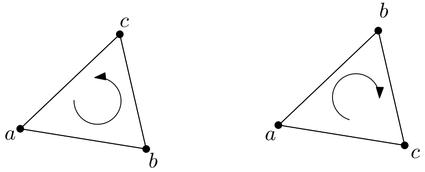
Als nächstes wollen wir uns mit der Implementierung des Greedy-Algorithmus befassen.

Orientierung eines Dreiecks: Ist das Tupel (a, b, c) gegen den Uhrzeigersinn orientiert?



6.1.3 Halbkanten-Datenstruktur

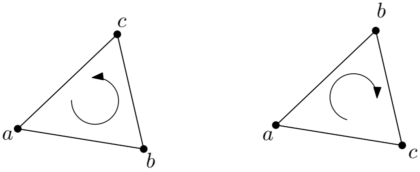
Orientierung eines Dreiecks: Ist das Tupel (a, b, c) gegen den Uhrzeigersinn orientiert?



Formel der Dreiecksfläche für Punkte $a = (a_1, a_2), b = (b_1, b_2), c = (c_1, c_2)$:

$$\frac{1}{2} \det \begin{pmatrix} a_1 & a_2 & 1 \\ b_1 & b_2 & 1 \\ c_1 & c_2 & 1 \end{pmatrix} = \frac{(c_1 - a_1)(c_2 + a_2) + (b_1 - c_1)(b_2 + c_2) + (a_1 - b_1)(a_2 + b_2)}{2}$$

Orientierung eines Dreiecks: Ist das Tupel (a, b, c) gegen den Uhrzeigersinn orientiert?



Formel der Dreiecksfläche für Punkte $a = (a_1, a_2)$, $b = (b_1, b_2)$, $c = (c_1, c_2)$:

$$\frac{1}{2} \det \begin{pmatrix} a_1 & a_2 & 1 \\ b_1 & b_2 & 1 \\ c_1 & c_2 & 1 \end{pmatrix} = \frac{(c_1 - a_1)(c_2 + a_2) + (b_1 - c_1)(b_2 + c_2) + (a_1 - b_1)(a_2 + b_2)}{2}$$

Genau dann wenn das Tupel (a, b, c) **gegen den Uhrzeigersinn** orientiert ist, ist die Dreiecksfläche **positiv**. Wir legen die Konvention fest, dass Dreiecke positiv orientiert sind.

Punkt links von der Geraden:

Gegeben drei Punkte $a, b, c \in \mathbb{R}^2$, liegt c links von der Geraden durch a und b, die in Richtung von a nach b orientiert ist?

Punkt links von der Geraden:

Gegeben drei Punkte $a, b, c \in \mathbb{R}^2$, liegt c links von der Geraden durch a und b, die in Richtung von a nach b orientiert ist?

$\implies$ Punkt c liegt **links** von der Geraden genau dann, wenn die Orientierung des Tupels (a, b, c) **gegen den Uhrzeigersinn** ist.

6.1.3 Halbkanten-Datenstruktur

Punkt links von der Geraden:

Gegeben drei Punkte $a, b, c \in \mathbb{R}^2$, liegt c links von der Geraden durch a und b, die in Richtung von a nach b orientiert ist?

$\Longrightarrow$ Punkt c liegt **links** von der Geraden genau dann, wenn die Orientierung des Tupels (a, b, c) **gegen den Uhrzeigersinn** ist.

Punkt im Dreieck:

Gegeben $q \in \mathbb{R}^2$ und drei Punkte $a, b, c \in \mathbb{R}^2$, ist q im Dreieck (a, b, c) enthalten? (Wir nehmen an, dass das Tupel (a, b, c) gegen den Uhrzeigersinn orientiert ist.)

6.1.3 Halbkanten-Datenstruktur

Punkt links von der Geraden:

Gegeben drei Punkte $a, b, c \in \mathbb{R}^2$, liegt c links von der Geraden durch a und b, die in Richtung von a nach b orientiert ist?

$\implies$ Punkt c liegt **links** von der Geraden genau dann, wenn die Orientierung des Tupels (a, b, c) **gegen den Uhrzeigersinn** ist.

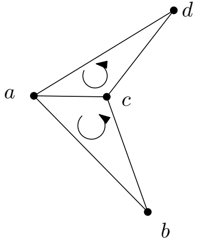
Punkt im Dreieck:

Gegeben $q \in \mathbb{R}^2$ und drei Punkte $a, b, c \in \mathbb{R}^2$, ist q im Dreieck (a, b, c) enthalten? (Wir nehmen an, dass das Tupel (a, b, c) gegen den Uhrzeigersinn orientiert ist.)

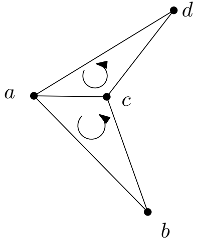
$\implies$ Der Punkt q ist genau dann **im Dreieck enthalten**, wenn **alle drei Tupel** (a, b, q), (b, c, q), und (c, a, q) **gegen den Uhrzeigersinn** orientiert sind.

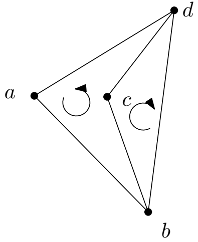
6.1.3 Halbkanten-Datenstruktur

Testen, ob ein Flip zulässig ist:

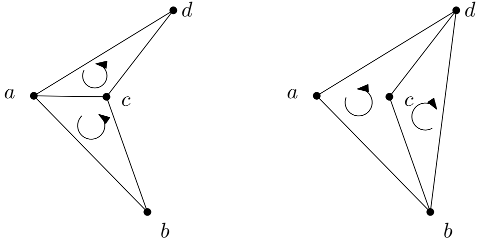


Testen, ob ein Flip zulässig ist:



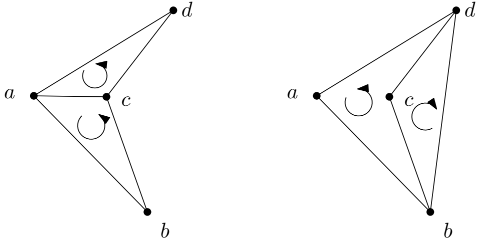


Testen, ob ein Flip zulässig ist:



In dem Beispiel ist der Flip von (a, c) zu (b, d) **nicht zulässig**, da die Kante (b, d) nicht innerhalb des Vierecks (a, b, c, d) verläuft.

Testen, ob ein Flip zulässig ist:



In dem Beispiel ist der Flip von (a, c) zu (b, d) **nicht zulässig**, da die Kante (b, d) nicht innerhalb des Vierecks (a, b, c, d) verläuft.

$\Longrightarrow$ Der Flip ist **zulässig** genau dann wenn beide Dreiecke (a, b, d) und (b, c, d) **gegen den Uhrzeigersinn** orientiert sind.

Punkt im Kreis: Gegeben ein Punkt $q = (q_1, q_2) \in \mathbb{R}^2$ und ein Kreis C mit Mittelpunkt $p = (p_1, p_2)$ und Radius r, ist q im Inneren von C enthalten?

Punkt im Kreis: Gegeben ein Punkt $q = (q_1, q_2) \in \mathbb{R}^2$ und ein Kreis C mit Mittelpunkt $p = (p_1, p_2)$ und Radius r, ist q im Inneren von C enthalten?

Das ist der Fall genau dann, wenn gilt

$$(p_1 - q_1)^2 + (p_2 - q_2)^2 < r^2.$$

Punkt im Kreis: Gegeben ein Punkt $q = (q_1, q_2) \in \mathbb{R}^2$ und ein Kreis C mit Mittelpunkt $p = (p_1, p_2)$ und Radius r, ist q im Inneren von C enthalten?

Das ist der Fall genau dann, wenn gilt

$$(p_1 - q_1)^2 + (p_2 - q_2)^2 < r^2.$$

Punkt im Umkreis eines Dreiecks: Gegeben ein Punkt $q \in \mathbb{R}^2$ und drei Punkte $a, b, c \in \mathbb{R}^2$, ist q im Umkreis von (a, b, c) enthalten?

Punkt im Kreis: Gegeben ein Punkt $q = (q_1, q_2) \in \mathbb{R}^2$ und ein Kreis C mit Mittelpunkt $p = (p_1, p_2)$ und Radius r, ist q im Inneren von C enthalten?

Das ist der Fall genau dann, wenn gilt

$$(p_1 - q_1)^2 + (p_2 - q_2)^2 < r^2.$$

Punkt im Umkreis eines Dreiecks: Gegeben ein Punkt $q \in \mathbb{R}^2$ und drei Punkte $a, b, c \in \mathbb{R}^2$, ist q im Umkreis von (a, b, c) enthalten?

Die kartesischen Koordinaten des Umkreismittelpunktes und der (quadratische) Radius des Umkreises lassen sich aus den Koordinaten der Punkte a, b, c herleiten.

6.1.3 Halbkanten-Datenstruktur

Halbkanten-Datenstruktur:

- Datenstruktur um **Triangulationen** zu speichern und zu manipulieren

6.1.3 Halbkanten-Datenstruktur

Halbkanten-Datenstruktur:

- Datenstruktur um **Triangulationen** zu speichern und zu manipulieren
- **Objekte** in der Datenstruktur werden durch **Zeiger** miteinander verknüpft

6.1.3 Halbkanten-Datenstruktur

Halbkanten-Datenstruktur:

- Datenstruktur um **Triangulationen** zu speichern und zu manipulieren
- **Objekte** in der Datenstruktur werden durch **Zeiger** miteinander verknüpft
- Datenstruktur speichert ein Objekt für jeden **Knoten** der Triangulation

Halbkanten-Datenstruktur:

- Datenstruktur um **Triangulationen** zu speichern und zu manipulieren
- **Objekte** in der Datenstruktur werden durch **Zeiger** miteinander verknüpft
- Datenstruktur speichert ein Objekt für jeden **Knoten** der Triangulation
- Jeweils **zwei** Objekte (sogenannte **Halbkanten**) repräsentieren eine **Kante** der Triangulation (eine Halbkante für jeweils eine Richtung)

6.1.3 Halbkanten-Datenstruktur

Halbkanten-Datenstruktur:

- Datenstruktur um **Triangulationen** zu speichern und zu manipulieren
- **Objekte** in der Datenstruktur werden durch **Zeiger** miteinander verknüpft
- Datenstruktur speichert ein Objekt für jeden **Knoten** der Triangulation
- Jeweils **zwei** Objekte (sogenannte **Halbkanten**) repräsentieren eine **Kante** der Triangulation (eine Halbkante für jeweils eine Richtung)
- **Flächen** der Triangulation werden **nicht explizit** gespeichert

6.1.3 Halbkanten-Datenstruktur

Halbkanten-Datenstruktur:

- Datenstruktur um **Triangulationen** zu speichern und zu manipulieren
- **Objekte** in der Datenstruktur werden durch **Zeiger** miteinander verknüpft
- Datenstruktur speichert ein Objekt für jeden **Knoten** der Triangulation
- Jeweils **zwei** Objekte (sogenannte **Halbkanten**) repräsentieren eine **Kante** der Triangulation (eine Halbkante für jeweils eine Richtung)
- **Flächen** der Triangulation werden **nicht explizit** gespeichert
- Jede Halbkante ist implizit mit der Fläche assoziiert, die zu ihrer **linken Seite** liegt wenn man entlang der Richtung der Halbkante schaut

Halbkanten-Datenstruktur:

- Datenstruktur um **Triangulationen** zu speichern und zu manipulieren
- **Objekte** in der Datenstruktur werden durch **Zeiger** miteinander verknüpft
- Datenstruktur speichert ein Objekt für jeden **Knoten** der Triangulation
- Jeweils **zwei** Objekte (sogenannte **Halbkanten**) repräsentieren eine **Kante** der Triangulation (eine Halbkante für jeweils eine Richtung)
- **Flächen** der Triangulation werden **nicht explizit** gespeichert
- Jede Halbkante ist implizit mit der Fläche assoziiert, die zu ihrer **linken Seite** liegt wenn man entlang der Richtung der Halbkante schaut
- **Halbkanten**, die mit einer Fläche assoziiert sind, sind mithilfe von Zeigern **zyklisch** entlang des Randes der Fläche verkettet

6.1.3 Halbkanten-Datenstruktur

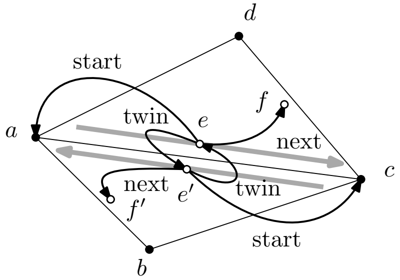
Zeiger eines Halfedge-Objekts:

start: Jede Halbkante speichert einen Zeiger auf seinen **Startknoten**.

twin: Die zwei Halbkanten, die eine feste Kante jeweils in eine der beiden Richtungen repräsentieren, speichern jeweils einen Zeiger aufeinander.

next: Jede Halbkante speichert einen Zeiger auf ihren **Nachfolger entlang des Randes der Fläche** die zu ihrer Linken liegt. Durch diese Relation sind Halbkanten, die einem Dreieck zugeordnet sind, zyklisch gegen den Uhrzeigersinn entlang des Randes der Fläche geordnet. Halbkanten die an der äußeren Fläche liegen sind dagegen im Uhrzeigersinn geordnet.

outer: Jede Halbkante speichert eine boolesche Variable, die angibt, ob die Fläche zu ihrer Linken die **äußeren Fläche** der Triangulation ist



d
start
f
twin
e
a
next
c
next
e'
twin
f'
start
b

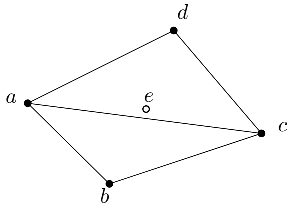
Zeiger eines Node-Objekts:

coords: Jeder Knoten speichert seine eigenen **Koordinaten**.

edge: Jeder Knoten speichert einen Zeiger auf **eine** der Halbkanten, welche diesen Knoten als Startknoten haben.

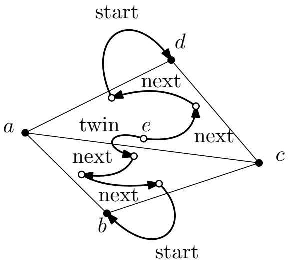
6.1.3 Halbkanten-Datenstruktur

```
IsLocallyDelaunay(Halfedge e)
    if(e.twin.outer or e.outer){ return True;}
    a = e.start;
    c = e.next.start;
    d = e.next.next.start;
    b = e.twin.next.next.start;
    return (d ∉ C(a, b, c) and b ∉ C(a, c, d));
```



6.1.3 Halbkanten-Datenstruktur

```
IsLocallyDelaunay(Halfedge e)
    if(e.twin.outer or e.outer){ return True;}
    a = e.start;
    c = e.next.start;
    d = e.next.next.start;
    b = e.twin.next.next.start;
    return (d ∉ C(a, b, c) and b ∉ C(a, c, d));
```



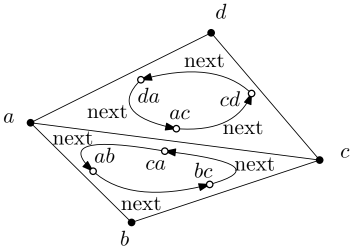
6.1.3 Halbkanten-Datenstruktur

FLIP(**Halfedge** *ac*)

$b = ac$.twin.next.next.start;
$d = ac$.next.next.start;
Initialisiere Zwillingskanten *bd* und *db* mit *b* und *d*.

//Aktualisiere next-Zeiger der Halbkanten
$cd = ac$.next; $ab = ac$.twin.next;
$da = cd$.next; $bc = ab$.next;
bd.next $=$ *da*; *da*.next = *ab*; *ab*.next = *bd*;
db.next $=$ *bc*; *bc*.next = *cd*; *cd*.next = *db*;

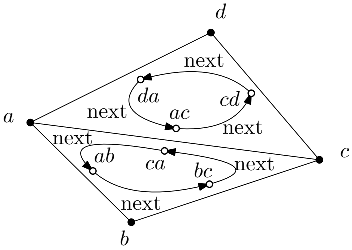
//Aktualisiere Zeiger auf ausgehende Kante
a.edge=*ab*; *c*.edge=*cd*;
return *bd*;

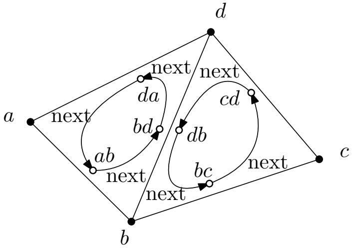


6.1.3 Halbkanten-Datenstruktur

Flip(Halfedge *ac*)

```
b = ac.twin.next.next.start;
d = ac.next.next.start;
Initialisiere Zwillingskanten bd und db mit b und d.

//Aktualisiere next-Zeiger der Halbkanten
cd = ac.next; ab = ac.twin.next;
da = cd.next; bc = ab.next;
bd.next = da; da.next = ab; ab.next = bd;
db.next = bc; bc.next = cd; cd.next = db;

//Aktualisiere Zeiger auf ausgehende Kante
a.edge=ab; c.edge=cd;
return bd;
```





```
GreedyFlips(Triangulation T, Kante e)

while (∃ Kante f in T, die einen Delaunay-Flip erlaubt) {
        Ersetze f durch die entsprechende Flip-Diagonale in T.
}
```

6.1.3 Halbkanten-Datenstruktur

GreedyFlips(**Triangulation** T**, Kante** e**)**

```
while (∃ Kante f in T, die einen Delaunay-Flip erlaubt) {
    Ersetze f durch die entsprechende Flip-Diagonale in T.
}
```

- Um die Abbruchbedingung der while-Schleife zu testen, führen wir eine **Breitensuche** auf T durch (siehe Übung), **ausgehend vom Startknoten von** e. Auf jeder besuchten Kante rufen wir IsLocallyDelaunay auf.

6.1.3 Halbkanten-Datenstruktur

```
GreedyFlips(Triangulation T, Kante e)

while (∃ Kante f in T, die einen Delaunay-Flip erlaubt) {
    Ersetze f durch die entsprechende Flip-Diagonale in T.
}
```

- Um die Abbruchbedingung der while-Schleife zu testen, führen wir eine **Breitensuche** auf T durch (siehe Übung), **ausgehend vom Startknoten von e**. Auf jeder besuchten Kante rufen wir IsLocallyDelaunay auf.
- Falls die Funktion IsLocallyDelaunay für eine Halbkante f **False** zurück gibt, dann erlaubt diese Kante einen **Delaunay-Flip**. Wir rufen dann Flip auf f auf.

6.1.3 Halbkanten-Datenstruktur

```
GreedyFlips(Triangulation T, Kante e)

while (∃ Kante f in T, die einen Delaunay-Flip erlaubt) {
    Ersetze f durch die entsprechende Flip-Diagonale in T.
}
```

- Um die Abbruchbedingung der while-Schleife zu testen, führen wir eine **Breitensuche** auf T durch (siehe Übung), **ausgehend vom Startknoten von** e. Auf jeder besuchten Kante rufen wir IsLocallyDelaunay auf.
- Falls die Funktion IsLocallyDelaunay für eine Halbkante f **False** zurück gibt, dann erlaubt diese Kante einen **Delaunay-Flip**. Wir rufen dann Flip auf f auf.
- Falls die Breitensuche keine solche Kante findet, dann ist die Delaunay-Triangulation erreicht (Lemma 6.13).

6.1.3 Halbkanten-Datenstruktur

Lemma 6.15

Sei eine Triangulation T mit n Punkten in einer Halbkanten-Datenstruktur und ein Zeiger auf eine Kante e auf der äußeren Fläche gegeben. Die Laufzeit des Algorithmus GreedyFlips ist in $O(n^3)$.

Beweis: Die Laufzeit von IsLocallyDelaunay und Flip ist jeweils in $O(1)$.

Lemma 6.15

Sei eine Triangulation T mit n Punkten in einer Halbkanten-Datenstruktur und ein Zeiger auf eine Kante e auf der äußeren Fläche gegeben. Die Laufzeit des Algorithmus GreedyFlips ist in $O(n^3)$.

Beweis: Die Laufzeit von IsLocallyDelaunay und Flip ist jeweils in $O(1)$.

Die Breitensuche benötigt im schlimmsten Fall Laufzeit $O(n)$ um einen Delaunay-Flip zu finden, oder um festzustellen, dass keiner existiert.

Lemma 6.15

Sei eine Triangulation T mit n Punkten in einer Halbkanten-Datenstruktur und ein Zeiger auf eine Kante e auf der äußeren Fläche gegeben. Die Laufzeit des Algorithmus GreedyFlips ist in $O(n^3)$.

Beweis: Die Laufzeit von IsLocallyDelaunay und Flip ist jeweils in $O(1)$.

Die Breitensuche benötigt im schlimmsten Fall Laufzeit $O(n)$ um einen Delaunay-Flip zu finden, oder um festzustellen, dass keiner existiert.

Laut Lemma 6.14 werden höchstens $O(n^2)$ Flips durchgeführt. Es gibt also $O(n^2)$ Iterationen der while-Schleife, und diese haben jeweils Laufzeit $O(n)$. □

Lemma 6.15

Sei eine Triangulation T mit n Punkten in einer Halbkanten-Datenstruktur und ein Zeiger auf eine Kante e auf der äußeren Fläche gegeben. Die Laufzeit des Algorithmus GREEDYFLIPS ist in $O(n^3)$.

Beweis: Die Laufzeit von ISLOCALLYDELAUNAY und FLIP ist jeweils in $O(1)$.

Die Breitensuche benötigt im schlimmsten Fall Laufzeit $O(n)$ um einen Delaunay-Flip zu finden, oder um festzustellen, dass keiner existiert.

Laut Lemma 6.14 werden höchstens $O(n^2)$ Flips durchgeführt. Es gibt also $O(n^2)$ Iterationen der while-Schleife, und diese haben jeweils Laufzeit $O(n)$. □

Die Delaunay-Triangulation kann auch **schneller** berechnet werden. Wir werden also nächstes einen Algorithmus mit Laufzeit $O(n^2)$ kennenlernen.

6.1.4 Inkrementeller Algorithmus

DELAUNAYTRIANGULATION(*S*)

Seien $p_1, \ldots, p_n$ die Punkte in S.
Initialisiere T als Triangulation mit Dreieck (p_1, p_2, p_3).
for $(i = 4\,;\, i \leq n\,;\, i{+}{+})\{$
    Füge p_i in T ein und ergänze Kantenmenge zu einer Triangulation.
    foreach(Halfedge e bildet mit p_i ein Dreieck in T)$\{$
        RESTOREDELAUNAY(e);
    $\}$
$\}$

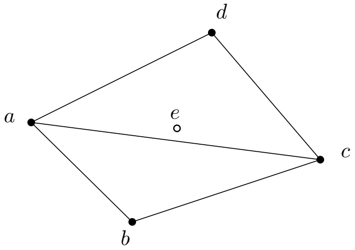
6.1.4 Inkrementeller Algorithmus

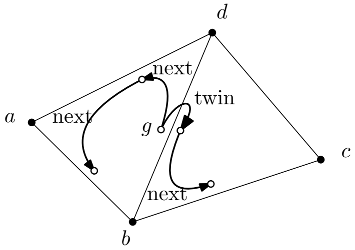
```
DelaunayTriangulation(S)

Seien p_1, ..., p_n die Punkte in S.
Initialisiere T als Triangulation mit Dreieck (p_1, p_2, p_3).
for (i = 4 ; i ≤ n ; i++){
    Füge p_i in T ein und ergänze Kantenmenge zu einer Triangulation.
    foreach(Halfedge e bildet mit p_i ein Dreieck in T){
        RestoreDelaunay(e);
    }
}
```

Die Funktion RestoreDelaunay stellt die Delaunay-Eigenschaft nach Einfügen des Punktes p_i rekursiv wieder her.

6.1.4 Inkrementeller Algorithmus

```
RestoreDelaunay(Halfedge e)

if(not IsLocallyDelaunay(e)){
        Halfedge g = Flip(e);
        RestoreDelaunay(g.next.next);
        RestoreDelaunay(g.twin.next);
}
```





Lemma 6.17

Betrachte einen Durchlauf der for-Schleife von DelaunayTriangulation, in dem der Punkt p_i hinzugefügt wird. RestoreDelaunay wird rekursiv auf genau den Kanten e aufgerufen, die zu einem Zeitpunkt ein Dreieck mit p_i in der Triangulation bilden.

Lemma 6.17

Betrachte einen Durchlauf der for-Schleife von DELAUNAYTRIANGULATION, in dem der Punkt p_i hinzugefügt wird. RESTOREDELAUNAY wird rekursiv auf genau den Kanten e aufgerufen, die zu einem Zeitpunkt ein Dreieck mit p_i in der Triangulation bilden.

Beweis: Induktion über die Rekursionstiefe.

Lemma 6.17

Betrachte einen Durchlauf der for-Schleife von DELAUNAYTRIANGULATION, in dem der Punkt p_i hinzugefügt wird. RESTOREDELAUNAY wird rekursiv auf genau den Kanten e aufgerufen, die zu einem Zeitpunkt ein Dreieck mit p_i in der Triangulation bilden.

Beweis: Induktion über die Rekursionstiefe. Der **initiale Aufruf** von RESTOREDELAUNAY passiert im Algorithmus DELAUNAYTRIANGULATION innerhalb der foreach-Schleife über die Halbkanten, die mit p_i ein Dreieck bilden.

Lemma 6.17

Betrachte einen Durchlauf der for-Schleife von DelaunayTriangulation, in dem der Punkt p_i hinzugefügt wird. RestoreDelaunay wird rekursiv auf genau den Kanten e aufgerufen, die zu einem Zeitpunkt ein Dreieck mit p_i in der Triangulation bilden.

Beweis: Induktion über die Rekursionstiefe. Der **initiale Aufruf** von RestoreDelaunay passiert im Algorithmus DelaunayTriangulation innerhalb der foreach-Schleife über die Halbkanten, die mit p_i ein Dreieck bilden.

Betrachten wir einen **beliebigen rekursiven Aufruf** von RestoreDelaunay (innerhalb von RestoreDelaunay aus aufgerufen). Per **Induktionsvoraussetzung** bildet die Kante e mit p_i ein Dreieck in der aktuellen Triangulation.

Lemma 6.17

Betrachte einen Durchlauf der for-Schleife von DelaunayTriangulation, in dem der Punkt p_i hinzugefügt wird. RestoreDelaunay wird rekursiv auf genau den Kanten e aufgerufen, die zu einem Zeitpunkt ein Dreieck mit p_i in der Triangulation bilden.

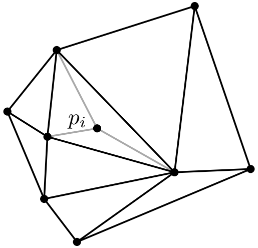
Beweis: Induktion über die Rekursionstiefe. Der **initiale Aufruf** von RestoreDelaunay passiert im Algorithmus DelaunayTriangulation innerhalb der foreach-Schleife über die Halbkanten, die mit p_i ein Dreieck bilden.

Betrachten wir einen **beliebigen rekursiven Aufruf** von RestoreDelaunay (innerhalb von RestoreDelaunay aus aufgerufen). Per **Induktionsvoraussetzung** bildet die Kante e mit p_i ein Dreieck in der aktuellen Triangulation.

Falls ein Flip durchgeführt wird, wird e durch g ersetzt. Danach, bildet p_i jeweils ein Dreieck mit beiden Kanten, mit denen RestoreDelaunay rekursiv aufgerufen wird. □

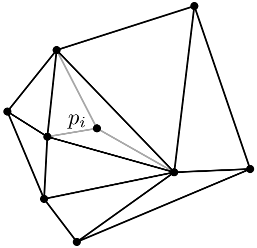
Beispiel:

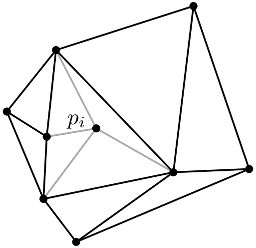
Sequenz von Delaunay-Flips nach Einfügen von Punkt p_i.



Beispiel:

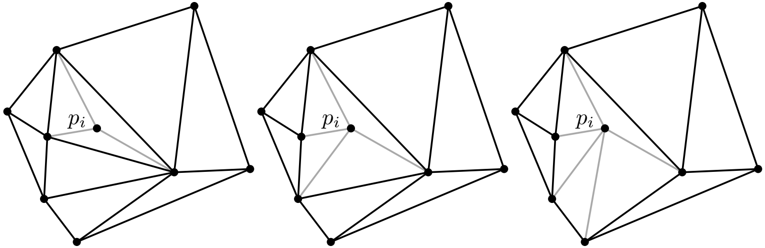
Sequenz von Delaunay-Flips nach Einfügen von Punkt p_i.





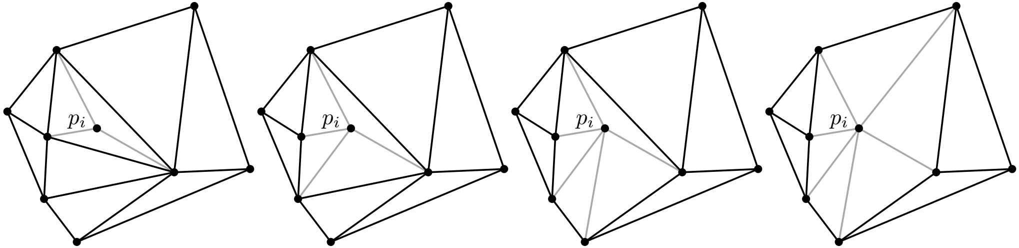
Beispiel:

Sequenz von Delaunay-Flips nach Einfügen von Punkt p_i.



Beispiel:

Sequenz von Delaunay-Flips nach Einfügen von Punkt p_i.

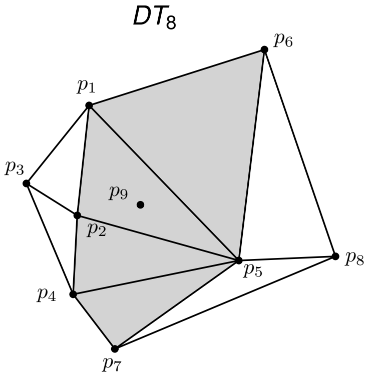


Lemma 6.16 (Stern-Lemma)

Seien $p_1, \ldots, p_n$ Punkte in der Ebene und sei DT_i die Delaunay-Triangulation der Punkte $p_1, \ldots, p_i$. Betrachte ein $i > 3$.

(i) Jedes Dreieck, das in DT_i enthalten ist, aber nicht in DT_{i-1}, muss p_i als einen seiner Eckpunkte haben.

(ii) Jedes in DT_{i-1} enthaltene Dreieck, das nicht mit p_i in Konflikt steht, ist auch in DT_i enthalten.



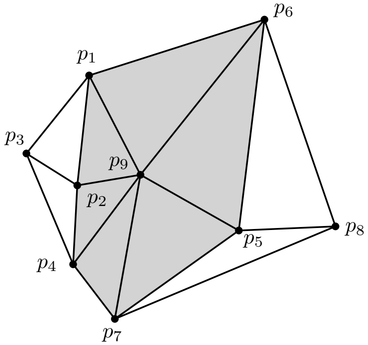
Lemma 6.16 (Stern-Lemma)

Seien $p_1, \ldots, p_n$ Punkte in der Ebene und sei DT_i die Delaunay-Triangulation der Punkte $p_1, \ldots, p_i$. Betrachte ein $i > 3$.

(i) Jedes Dreieck, das in DT_i enthalten ist, aber nicht in DT_{i-1}, muss p_i als einen seiner Eckpunkte haben.

(ii) Jedes in DT_{i-1} enthaltene Dreieck, das nicht mit p_i in Konflikt steht, ist auch in DT_i enthalten.

DT_9



Beweis (Stern-Lemma): Wir zeigen beide Teile des Lemmas separat.

(i) Sei D Dreieck in DT_i welches nur Punkte aus $p_1, \ldots, p_{i-1}$ als Eckpunkte hat.

Beweis (Stern-Lemma): Wir zeigen beide Teile des Lemmas separat.

(i) Sei D Dreieck in DT_i welches nur Punkte aus $p_1, \ldots, p_{i-1}$ als Eckpunkte hat. Da D keinen Konflikt mit $p_1, \ldots, p_i$ hat, ist D auch ein Dreieck in DT_{i-1}.

Beweis (Stern-Lemma): Wir zeigen beide Teile des Lemmas separat.

(i) Sei D Dreieck in DT_i welches nur Punkte aus $p_1, \ldots, p_{i-1}$ als Eckpunkte hat. Da D keinen Konflikt mit $p_1, \ldots, p_i$ hat, ist D auch ein Dreieck in DT_{i-1}. Also muss jedes Dreieck in DT_i, das nicht in DT_{i-1} enthalten ist, p_i als Eckpunkt haben.

6.1.4 Inkrementeller Algorithmus

Beweis (Stern-Lemma): Wir zeigen beide Teile des Lemmas separat.

(i) Sei D Dreieck in DT_i welches nur Punkte aus $p_1, \ldots, p_{i-1}$ als Eckpunkte hat. Da D keinen Konflikt mit $p_1, \ldots, p_i$ hat, ist D auch ein Dreieck in DT_{i-1}. Also muss jedes Dreieck in DT_i, das nicht in DT_{i-1} enthalten ist, p_i als Eckpunkt haben.

(ii) Wenn ein Dreieck aus DT_{i-1} nicht mit p_i im Konflikt steht, dann steht es mit keinem Punkt aus der Menge $p_1, \ldots, p_i$ im Konflikt und muss in DT_i enthalten sein. $\square$

Lemma 6.18

Betrachte einen Durchlauf der for-Schleife in DELAUNAYTRIANGULATION, in dem der Punkt p_i hinzugefügt wurde. Am Ende ist die Delaunay-Eigenschaft in Bezug auf p_i hergestellt.

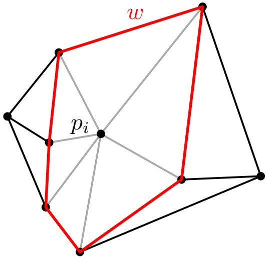
6.1.4 Inkrementeller Algorithmus

Lemma 6.18

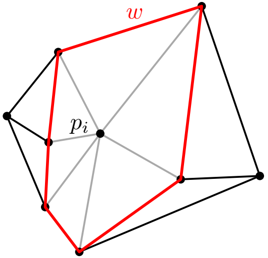
Betrachte einen Durchlauf der for-Schleife in DelaunayTriangulation, in dem der Punkt p_i hinzugefügt wurde. Am Ende ist die Delaunay-Eigenschaft in Bezug auf p_i hergestellt.

Beweis:

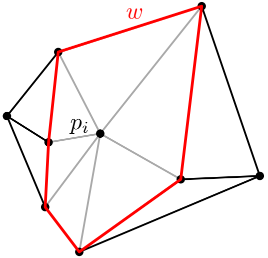
Sei T die **Triangulation** am Ende des Durchlaufs der for-Scheife.
Betrachte **Kreis** w aus den Kanten die mit p_i in T ein Dreieck bilden.
Aus Lemma 6.17 folgt, dass IsLocallyDelaunay auf jeder **Kante** e **von** w aufgerufen wurde und **True** zurückgegeben hat, sonst wäre e durch einen Flip ersetzt worden.



6.1.4 Inkrementeller Algorithmus



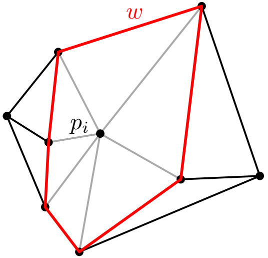
Die **anliegenden Dreiecke** der Triangulation, die eine Kante auf dem **Kreis** w haben, sind also **nicht im Konflikt mit** p_i.



Die **anliegenden Dreiecke** der Triangulation, die eine Kante auf dem **Kreis w** haben, sind also **nicht im Konflikt mit p_i**.

Aus Lemma 6.16 (ii) folgt somit, dass **w in DT_i** enthalten ist.

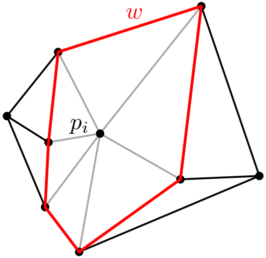
6.1.4 Inkrementeller Algorithmus



Die **anliegenden Dreiecke** der Triangulation, die eine Kante auf dem **Kreis w** haben, sind also **nicht im Konflikt mit p_i**.

Aus Lemma 6.16 (ii) folgt somit, dass **w in DT_i** enthalten ist.

Angenommen es gibt ein **Dreieck** (a, b, c) in T das **mit p_i im Konflikt** steht, dann müsste mindestens eine Kante $(p_i, a), (p_i, b)$ oder (p_i, c) in DT_i sein, da laut Lemma 6.16 (i) alle neuen Dreiecke in DT_i zu p_i inzident sind.



Die **anliegenden Dreiecke** der Triangulation, die eine Kante auf dem **Kreis** w haben, sind also **nicht im Konflikt mit** p_i.

Aus Lemma 6.16 (ii) folgt somit, dass w **in** DT_i enthalten ist.

Angenommen es gibt ein **Dreieck** (a, b, c) in T das **mit** p_i **im Konflikt** steht, dann müsste mindestens eine Kante $(p_i, a), (p_i, b)$ oder (p_i, c) in DT_i sein, da laut Lemma 6.16 (i) alle neuen Dreiecke in DT_i zu p_i inzident sind.

Diese Kante müsste w **kreuzen**, da a, b, c ausserhalb von w liegen. Dies ist ein **Widerspruch**, da Delaunay-Kanten sich nicht kreuzen (Lemma 6.6). $\square$

Implementierung von Zeile 4:

Wie wird der neue Punkt p_i der Triangulation eingefügt?

Implementierung von Zeile 4:

Wie wird der neue Punkt p_i der Triangulation eingefügt?

- Finde die **Fläche** in der Triangulation, die p_i **enthält**, mittels **Breitensuche** über die Kanten der Triangulation. Für jede Kante, teste die maximal zwei inzidenten Dreiecke.

Implementierung von Zeile 4:

Wie wird der neue Punkt p_i der Triangulation eingefügt?

- Finde die **Fläche** in der Triangulation, die p_i **enthält**, mittels **Breitensuche** über die Kanten der Triangulation. Für jede Kante, teste die maximal zwei inzidenten Dreiecke.
- Wenn die Fläche ein **Dreieck** ist, dann füge drei Kanten von p_i zu allen Knoten des Dreiecks hinzu.

Implementierung von Zeile 4:

Wie wird der neue Punkt p_i der Triangulation eingefügt?

- Finde die **Fläche** in der Triangulation, die p_i **enthält**, mittels **Breitensuche** über die Kanten der Triangulation. Für jede Kante, teste die maximal zwei inzidenten Dreiecke.
- Wenn die Fläche ein **Dreieck** ist, dann füge drei Kanten von p_i zu allen Knoten des Dreiecks hinzu.
- Wenn **kein Dreieck** gefunden wird, das den Punkt p_i enthält, dann ist p_i in der **äußeren Fläche**.

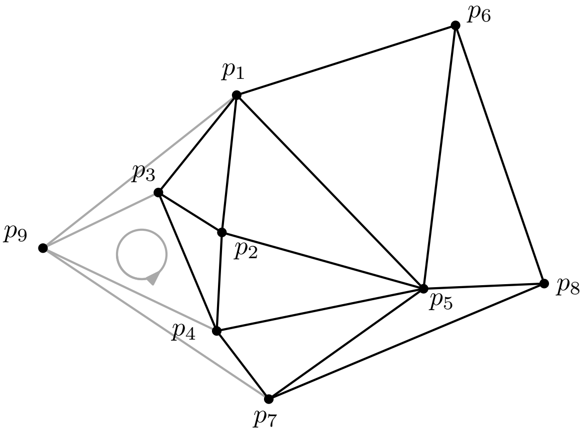
6.1.4 Inkrementeller Algorithmus

Implementierung von Zeile 4:

Wie wird der neue Punkt p_i der Triangulation eingefügt?

- Finde die **Fläche** in der Triangulation, die p_i **enthält**, mittels **Breitensuche** über die Kanten der Triangulation. Für jede Kante, teste die maximal zwei inzidenten Dreiecke.
- Wenn die Fläche ein **Dreieck** ist, dann füge drei Kanten von p_i zu allen Knoten des Dreiecks hinzu.
- Wenn **kein Dreieck** gefunden wird, das den Punkt p_i enthält, dann ist p_i in der **äußeren Fläche**. In diesem Fall müssen nur diese Halbkanten (a, c) mit dem neuen Punkt p_i zu einem Dreieck verbunden werden, wo das Tupel (a, c, p_i) entgegen dem Uhrzeigersinn orientiert ist.

6.1.4 Inkrementeller Algorithmus



Theorem 6.19

Die Delaunay-Triangulation von n Punkten in allgemeiner Lage in der Ebene kann in $O(n^2)$ Zeit berechnet werden.

Beweis: Die Korrektheit von DELAUNAYTRIANGULATION folgt aus Lemmas 6.18 und 6.16.

Theorem 6.19

Die Delaunay-Triangulation von n Punkten in allgemeiner Lage in der Ebene kann in $O(n^2)$ Zeit berechnet werden.

Beweis: Die Korrektheit von DelaunayTriangulation folgt aus Lemmas 6.18 und 6.16.

Das Einfügen von p_i in Zeile 4 kann in $O(n)$ Zeit durchgeführt werden.

Theorem 6.19

Die Delaunay-Triangulation von n Punkten in allgemeiner Lage in der Ebene kann in $O(n^2)$ Zeit berechnet werden.

Beweis: Die Korrektheit von DELAUNAYTRIANGULATION folgt aus Lemmas 6.18 und 6.16.

Das Einfügen von p_i in Zeile 4 kann in $O(n)$ Zeit durchgeführt werden.

Die Anzahl der Flips per Durchlauf der for-Scheife ist in $O(n)$, da jede Flip-Diagonale zu p_i inzident ist und die Anzahl der Kanten der Triangulation in $O(n)$ ist (Theorem 6.2).

Theorem 6.19

Die Delaunay-Triangulation von n Punkten in allgemeiner Lage in der Ebene kann in $O(n^2)$ Zeit berechnet werden.

Beweis: Die Korrektheit von DELAUNAYTRIANGULATION folgt aus Lemmas 6.18 und 6.16.

Das Einfügen von p_i in Zeile 4 kann in $O(n)$ Zeit durchgeführt werden.

Die Anzahl der Flips per Durchlauf der for-Scheife ist in $O(n)$, da jede Flip-Diagonale zu p_i inzident ist und die Anzahl der Kanten der Triangulation in $O(n)$ ist (Theorem 6.2).

Es gibt n Durchläufe der for-Schleife, also insgesamt ist die Laufzeit in $O(n^2)$. $\square$

6.1.4 Inkrementeller Algorithmus

Theorem 6.19

Die Delaunay-Triangulation von n Punkten in allgemeiner Lage in der Ebene kann in $O(n^2)$ Zeit berechnet werden.

Beweis: Die Korrektheit von DELAUNAYTRIANGULATION folgt aus Lemmas 6.18 und 6.16.

Das Einfügen von p_i in Zeile 4 kann in $O(n)$ Zeit durchgeführt werden.

Die Anzahl der Flips per Durchlauf der for-Scheife ist in $O(n)$, da jede Flip-Diagonale zu p_i inzident ist und die Anzahl der Kanten der Triangulation in $O(n)$ ist (Theorem 6.2).

Es gibt n Durchläufe der for-Schleife, also insgesamt ist die Laufzeit in $O(n^2)$. □

Insgesamt werden $O(n^2)$ Flips durchgeführt. Als nächstes wollen wir zeigen, dass die erwartete Anzahl der Flips viel kleiner sein kann.

Theorem 6.20

Sei die Einfügereihenfolge der n Punkte zufällig gleichverteilt über alle möglichen Permutationen gewählt. Dann ist die erwartete Anzahl der Flips, die der Algorithmus DelaunayTriangulation insgesamt durchführt, in $O(n)$.

Theorem 6.20

Sei die Einfügereihenfolge der n Punkte zufällig gleichverteilt über alle möglichen Permutationen gewählt. Dann ist die erwartete Anzahl der Flips, die der Algorithmus DELAUNAYTRIANGULATION insgesamt durchführt, in $O(n)$.

Beweis: Wir nehmen an, die zufällige Permutation wird **rückwärts** generiert. Wähle zuerst den Punkt p_n zufällig gleichverteilt aus der Gesamtmenge der Punkte, dann für $j = 1, \ldots n - 2$ wähle p_{n-j} zufällig gleichverteilt aus der Restmenge S_j.

Theorem 6.20

Sei die Einfügereihenfolge der n Punkte zufällig gleichverteilt über alle möglichen Permutationen gewählt. Dann ist die erwartete Anzahl der Flips, die der Algorithmus DELAUNAYTRIANGULATION insgesamt durchführt, in $O(n)$.

Beweis: Wir nehmen an, die zufällige Permutation wird **rückwärts** generiert. Wähle zuerst den Punkt p_n zufällig gleichverteilt aus der Gesamtmenge der Punkte, dann für $j = 1, \ldots n - 2$ wähle p_{n-j} zufällig gleichverteilt aus der Restmenge S_j.

Betrachte die **erwartete Anzahl von Flips**, die in **einem Durchlauf** der for-Schleife durchgeführt werden, unter der Annahme, dass die Restmenge S_j fest ist.

6.1.4 Inkrementeller Algorithmus

Theorem 6.20

Sei die Einfügereihenfolge der n Punkte zufällig gleichverteilt über alle möglichen Permutationen gewählt. Dann ist die erwartete Anzahl der Flips, die der Algorithmus DELAUNAYTRIANGULATION insgesamt durchführt, in $O(n)$.

Beweis: Wir nehmen an, die zufällige Permutation wird **rückwärts** generiert. Wähle zuerst den Punkt p_n zufällig gleichverteilt aus der Gesamtmenge der Punkte, dann für $j = 1, \ldots n-2$ wähle p_{n-j} zufällig gleichverteilt aus der Restmenge S_j.

Betrachte die **erwartete Anzahl von Flips**, die in **einem Durchlauf** der for-Schleife durchgeführt werden, unter der Annahme, dass die Restmenge S_j fest ist.

Menge S_j ist gleichzeitig **Knotenmenge** von DT_{i-1} für $i = n - j$. Da S_j fest ist, ist auch die Delaunay-Triangulation DT_{i-1} fest.

6.1.4 Inkrementeller Algorithmus

Laut Stern-Lemma ist die **Anzahl der Flips**, die nach dem Hinzufügen von p_i zu DT_{i-1} vom Algorithmus durchgeführt werden, gleich dem **Knotengrad** von p_i in DT_i.

6.1.4 Inkrementeller Algorithmus

Laut Stern-Lemma ist die **Anzahl der Flips**, die nach dem Hinzufügen von p_i zu DT_{i-1} vom Algorithmus durchgeführt werden, gleich dem **Knotengrad** von p_i in DT_i.

Aus Theorem 6.2 folgt, dass die Anzahl der Kanten in DT_i in $O(i)$ ist.

6.1.4 Inkrementeller Algorithmus

Laut Stern-Lemma ist die **Anzahl der Flips**, die nach dem Hinzufügen von p_i zu DT_{i-1} vom Algorithmus durchgeführt werden, gleich dem **Knotengrad** von p_i in DT_i.

Aus Theorem 6.2 folgt, dass die Anzahl der Kanten in DT_i in $O(i)$ ist.

Der **erwartete Knotengrad** eines gleichverteilt zufällig gewählten Knoten in DT_i ist also konstant.

6.1.4 Inkrementeller Algorithmus

Laut Stern-Lemma ist die **Anzahl der Flips**, die nach dem Hinzufügen von p_i zu DT_{i-1} vom Algorithmus durchgeführt werden, gleich dem **Knotengrad** von p_i in DT_i.

Aus Theorem 6.2 folgt, dass die Anzahl der Kanten in DT_i in $O(i)$ ist.

Der **erwartete Knotengrad** eines gleichverteilt zufällig gewählten Knoten in DT_i ist also konstant. Da wir p_i gleichverteilt zufällig aus S_j wählen, ist die erwartete Anzahl von Flips nach Einfügen von p_i konstant.

6.1.4 Inkrementeller Algorithmus

Laut Stern-Lemma ist die **Anzahl der Flips**, die nach dem Hinzufügen von p_i zu DT_{i-1} vom Algorithmus durchgeführt werden, gleich dem **Knotengrad** von p_i in DT_i.

Aus Theorem 6.2 folgt, dass die Anzahl der Kanten in DT_i in $O(i)$ ist.

Der **erwartete Knotengrad** eines gleichverteilt zufällig gewählten Knoten in DT_i ist also konstant. Da wir p_i gleichverteilt zufällig aus S_j wählen, ist die erwartete Anzahl von Flips nach Einfügen von p_i konstant.

Dies gilt unabhängig von der konkreten Wahl der Restmenge S_j. Es gilt also allgemein für jede solche Wahl.

6.1.4 Inkrementeller Algorithmus

Laut Stern-Lemma ist die **Anzahl der Flips**, die nach dem Hinzufügen von p_i zu DT_{i-1} vom Algorithmus durchgeführt werden, gleich dem **Knotengrad** von p_i in DT_i.

Aus Theorem 6.2 folgt, dass die Anzahl der Kanten in DT_i in $O(i)$ ist.

Der **erwartete Knotengrad** eines gleichverteilt zufällig gewählten Knoten in DT_i ist also konstant. Da wir p_i gleichverteilt zufällig aus S_j wählen, ist die erwartete Anzahl von Flips nach Einfügen von p_i konstant.

Dies gilt unabhängig von der konkreten Wahl der Restmenge S_j. Es gilt also allgemein für jede solche Wahl.

Insgesamt ist die erwartete Anzahl von Flips also in $O(n)$. $\square$