Technische Universität Braunschweig

# Einführung in algorithmische Geometrie - Punktlokalisierung

Arne Schmidt

# Punktlokalisierung

Wie finden wir den Bereich, in dem ein Punkt liegt?

Brute-Force: Teste jeden Bereich, ob Punkte darin liegt.

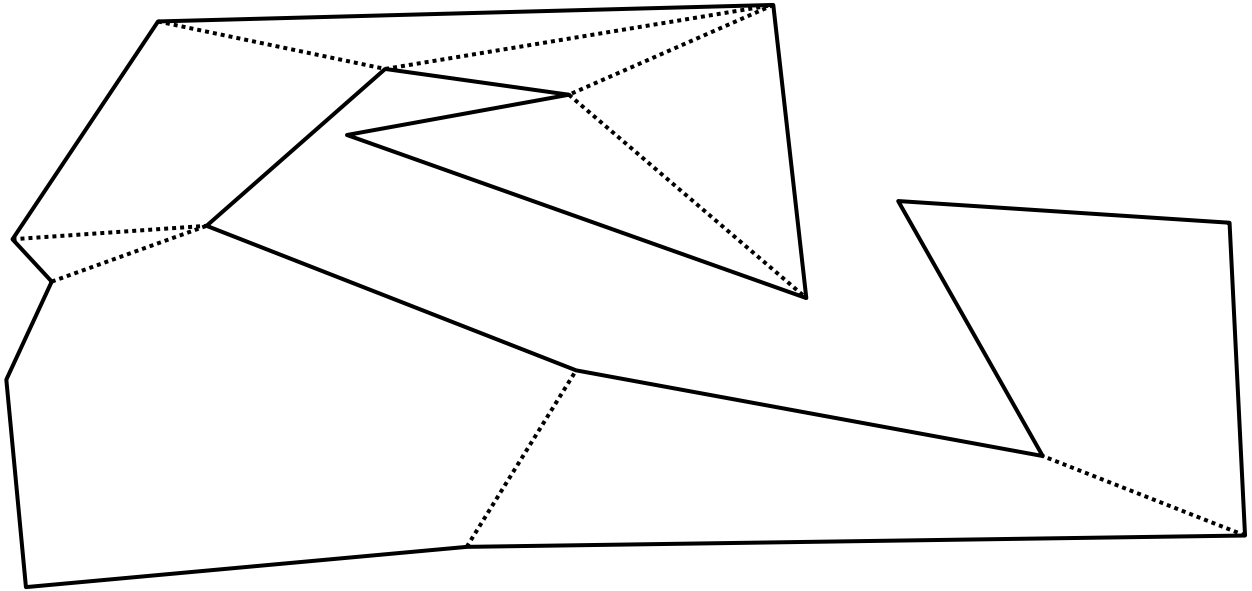
Wie schnell kann man prüfen, ob ein Punkt in einem Polygon liegt?

# Polygon Punktlokalisierung

Für konvexe Polygone kennen wir bereits ein Verfahren aus VL 0.

→ Unterteile Polygone in konvexe Bereiche!

→ Teste jeden Bereich.

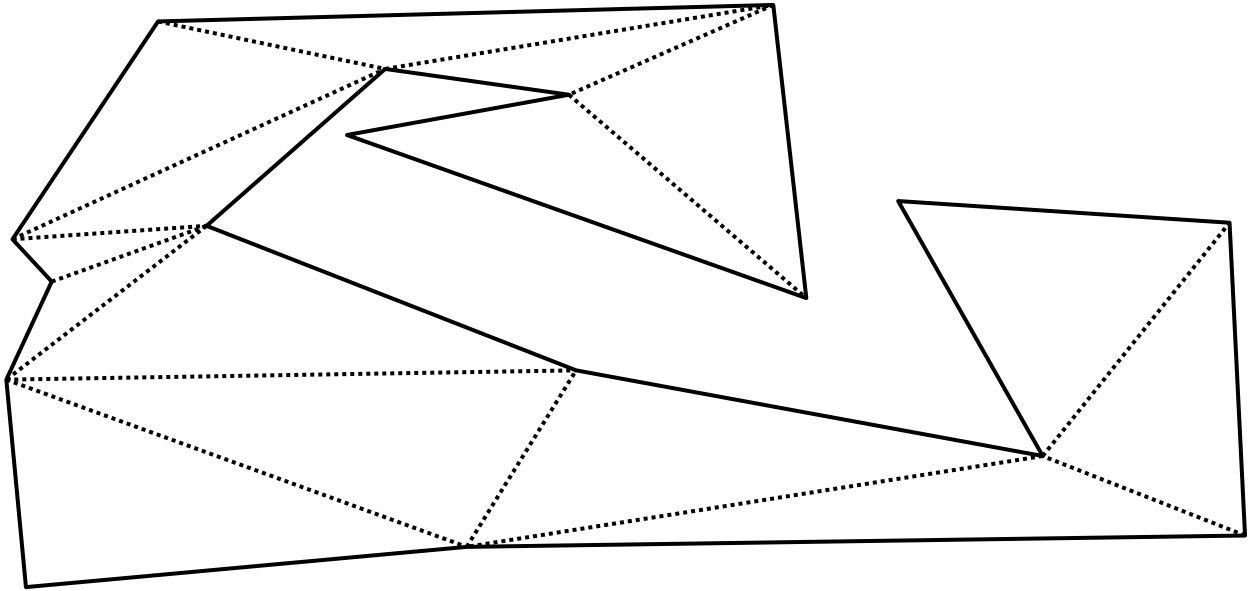


# Polygon Punktlokalisierung

Für konvexe Polygone kennen wir bereits ein Verfahren aus VL 0.

→ Unterteile Polygone in konvexe Bereiche! Aber wie viele?

→ Teste jeden Bereich.



# Polygon Triangulation

**Definition 2.26 ((einfaches) Polygon)**
Ein Polygon $P := (p_1, \dots, p_n)$ mit $p_i \in \mathbb{R}^2$ ist ein geschlossener, sich nicht selbst-schneidener Kantenzug auf $n > 2$ Punkten. $P$ besteht also aus …

a) Genau $n$ Eckpunkten
b) Insgesamt $n$ Segmenten $p_i p_{i+1}$ für $i \in \{1, \dots, n-1\}$, sowie $p_n p_1$

**Definition 2.27 (Polygon-Triangulierung)**
Eine Triangulierung $T(P)$ eines Polygons $P := (p_1, \dots, p_n)$ entsteht durch Hinzufügen von Segmenten zwischen Eckpunkten von $P$, sodass …

a) Keine zwei Segmente sich schneiden.
b) Jedes Segment innerhalb von $P$ verläuft.
c) Kein weiteres Segment hinzugefügt werden kann.

$T(P)$ ist damit eine Unterteilung von $P$ in Dreiecke.

# Polygon Triangulation

**Lemma 2.28**
Sei $P$ ein Polygon mit $n > 2$ Punkten. Dann besitzt $T(P)$ genau $n - 2$ Dreiecke.

Beweis: Selbst!
(Hinweis: Vollständige Induktion)

**Lemma 2.29**
Sei $P$ ein Polygon mit $n > 2$ Punkten mit gegebener Triangulierung $T(P)$. Der Test, ob ein Punkt $q \in \mathbb{R}^2$ innerhalb von $P$ liegt, kann in $O(n)$ Zeit erfolgen.

# Geht das schneller?

Für k Queries dauert das $O(kn)$ Zeit.

Können wir wieder eine Datenstruktur vorbereiten?

Am besten sollte Query-Zeit in O(log n) liegen.

# Kapitel 2.2.5 – Arrangement Triangulationen

# Arrangement Triangulation – Definition

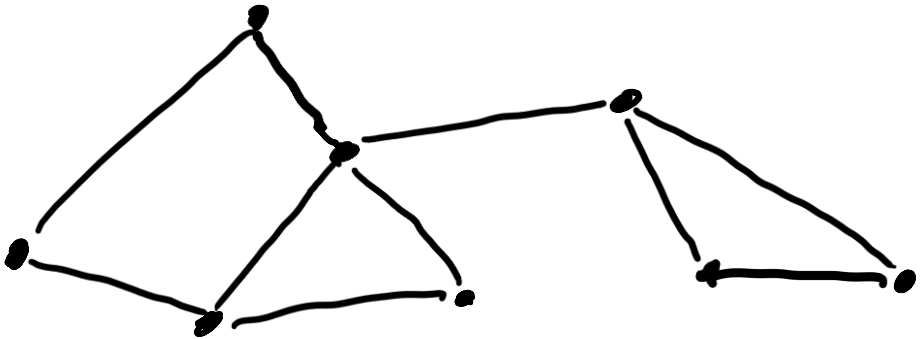
**Definition 2.30 (Arrangement-Triangulation)**
Eine Triangulierung $T(\mathcal{A})$ eines Arrangements $\mathcal{A}$ entsteht durch Hinzufügen eines Dreiecks, welches $\mathcal{A}$ umschließt, sowie Segmenten zwischen Knoten, sodass …

a) Keine zwei Segmente sich schneiden.

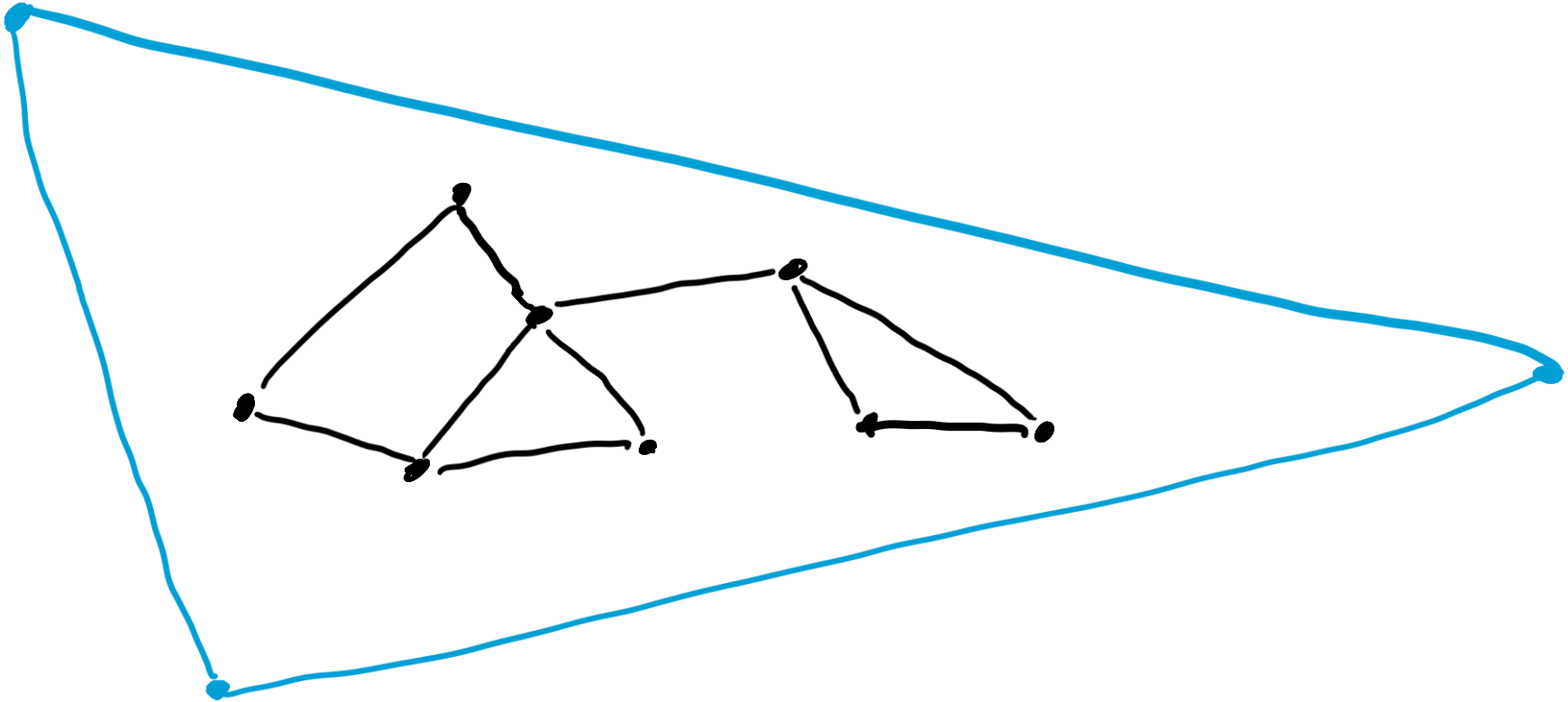
b) Kein weiteres Segment hinzugefügt werden kann.

$T(\mathcal{A})$ enthält damit nur Dreiecke.

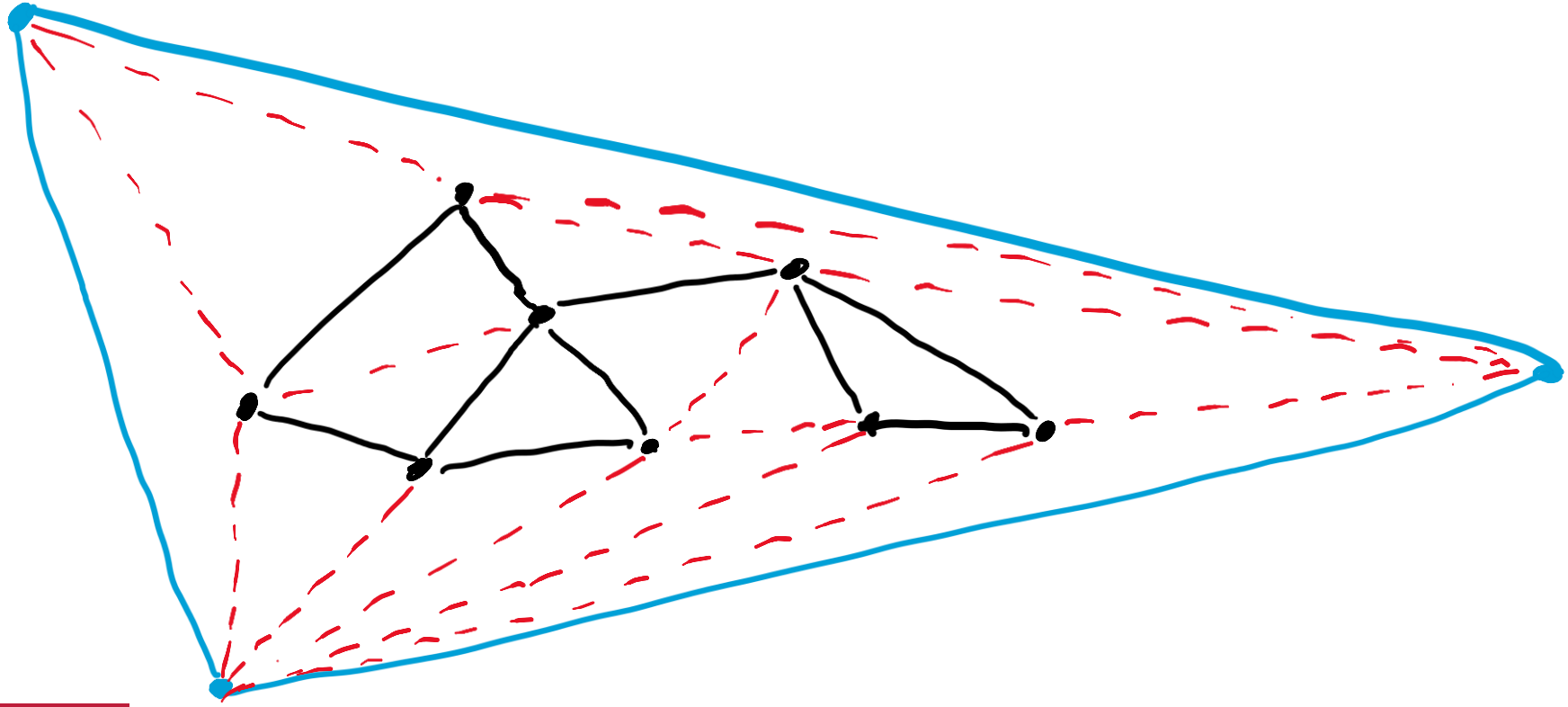
# Triangulations-Beispiel



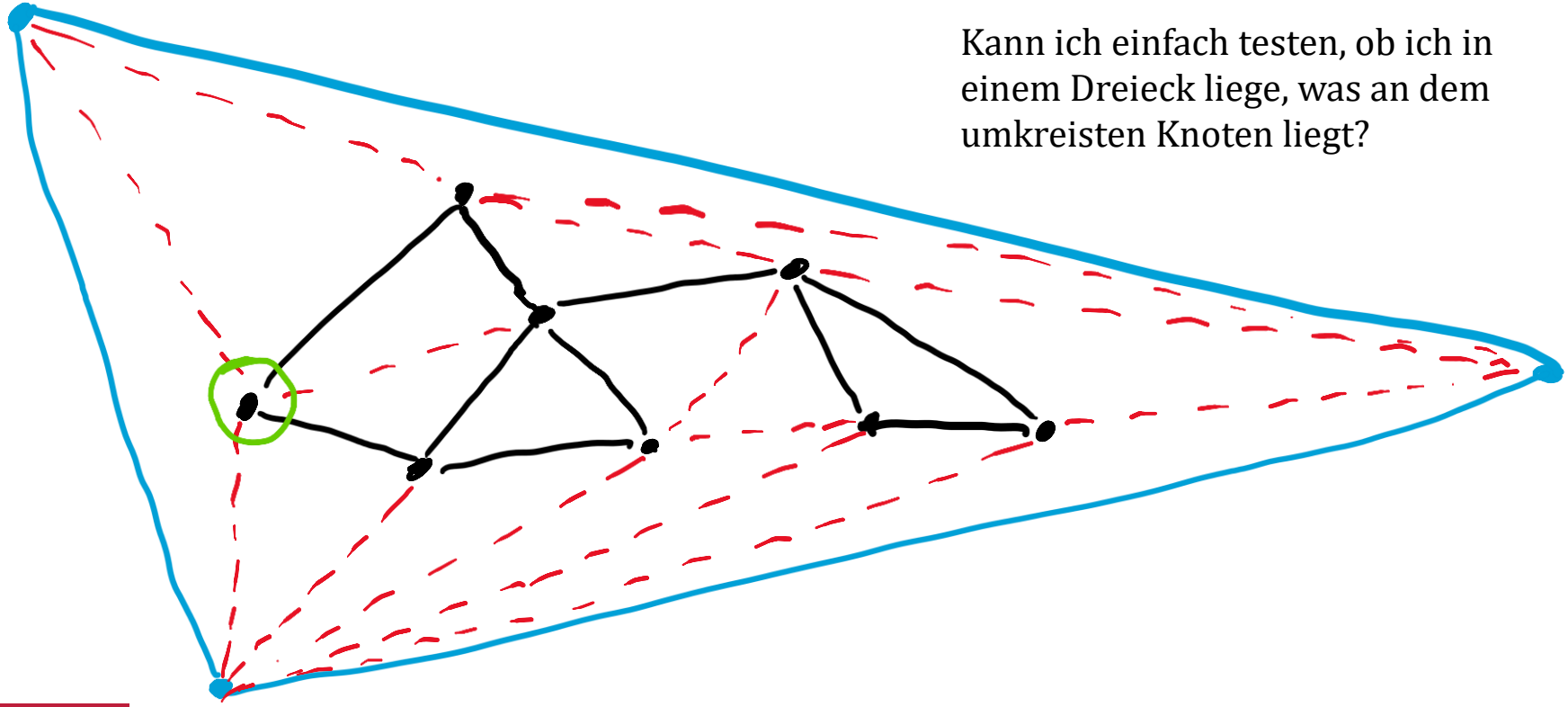
# Triangulations-Beispiel



# Triangulations-Beispiel

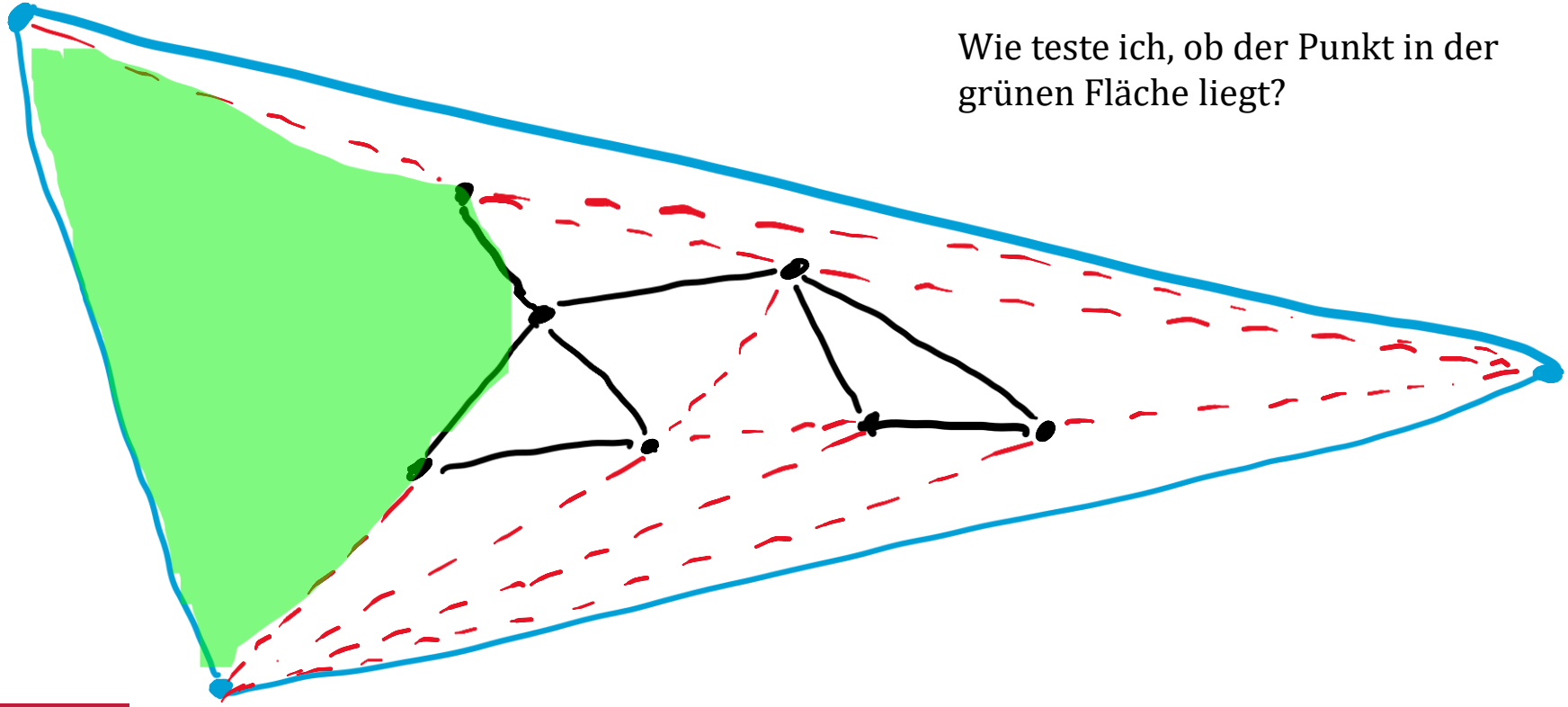


# Triangulations-Beispiel



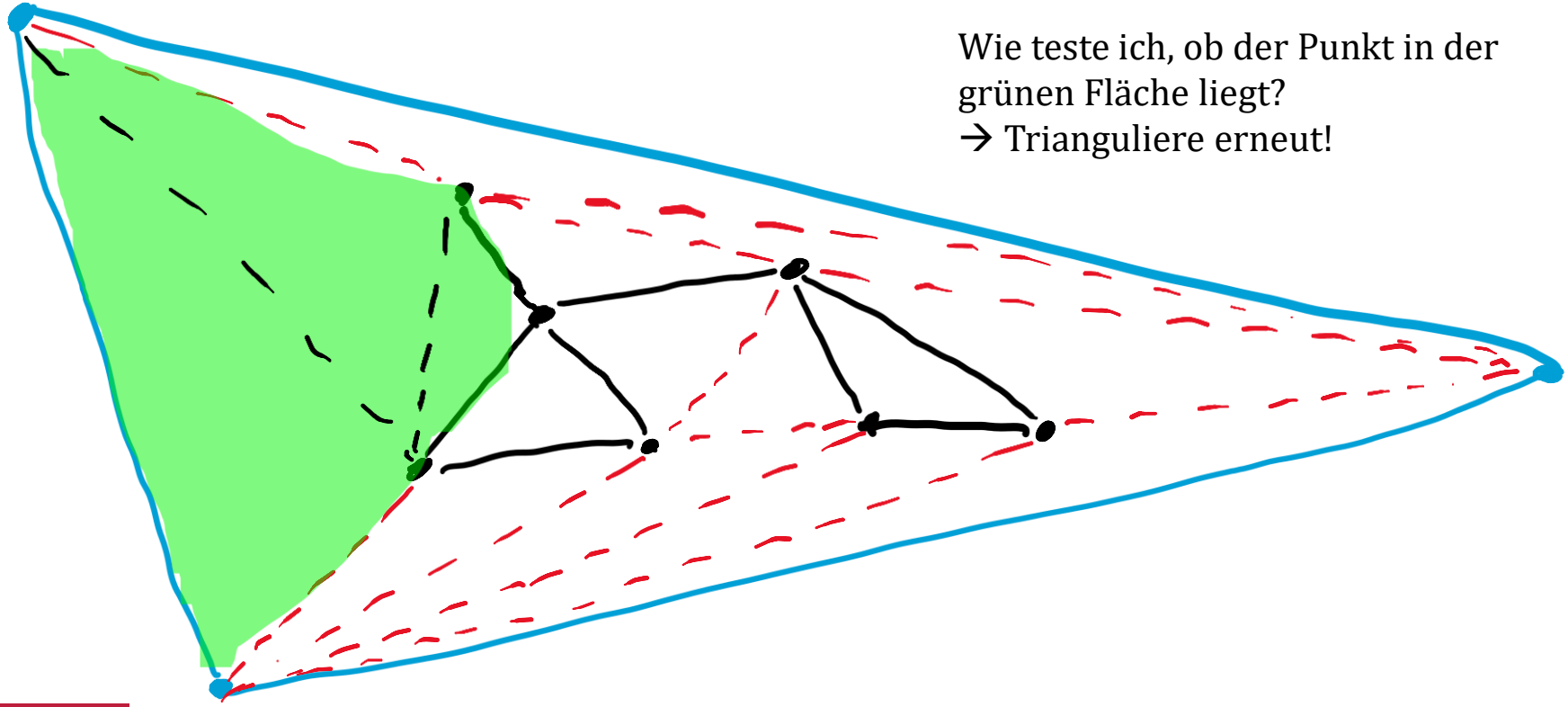
Kann ich einfach testen, ob ich in einem Dreieck liege, was an dem umkreisten Knoten liegt?

# Simplifizieren



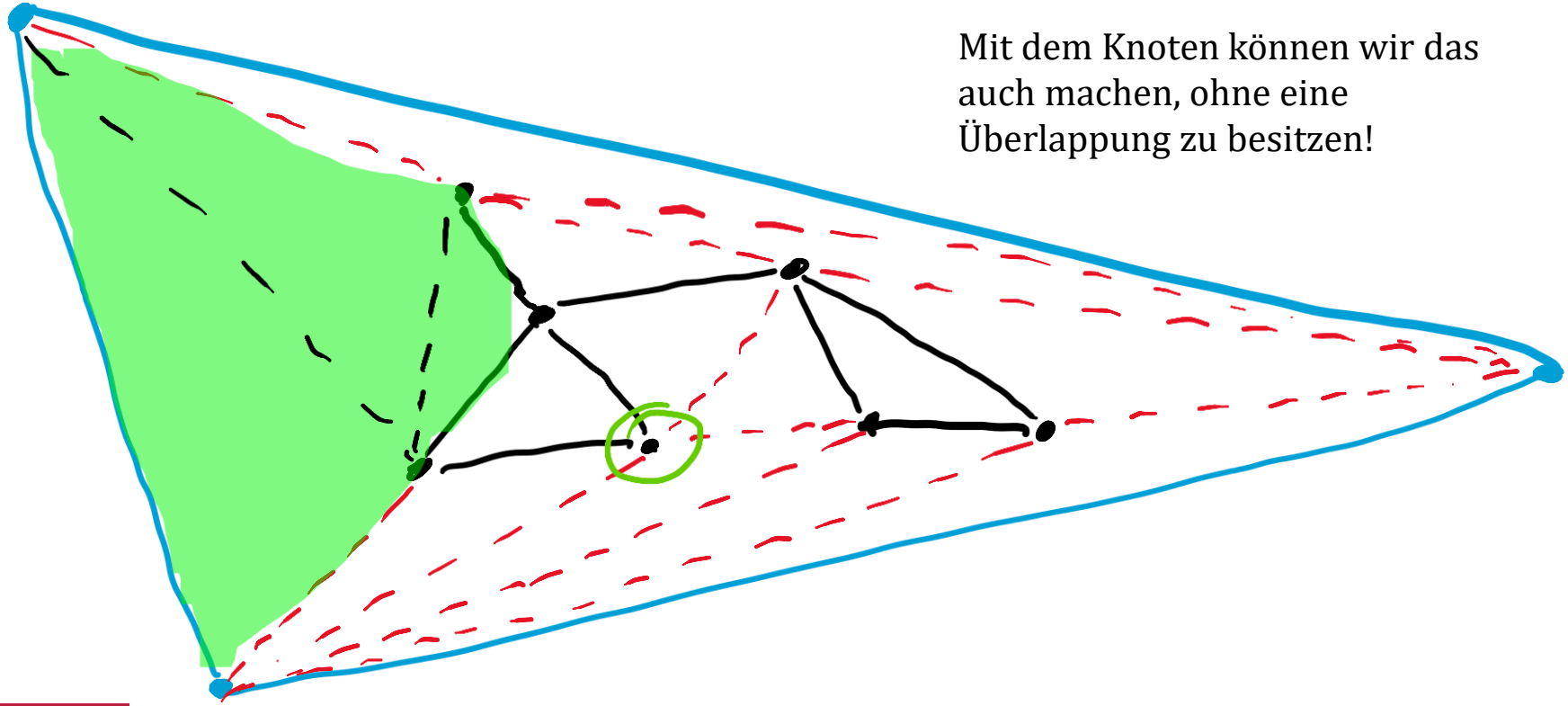
Wie teste ich, ob der Punkt in der grünen Fläche liegt?

# Simplifizieren



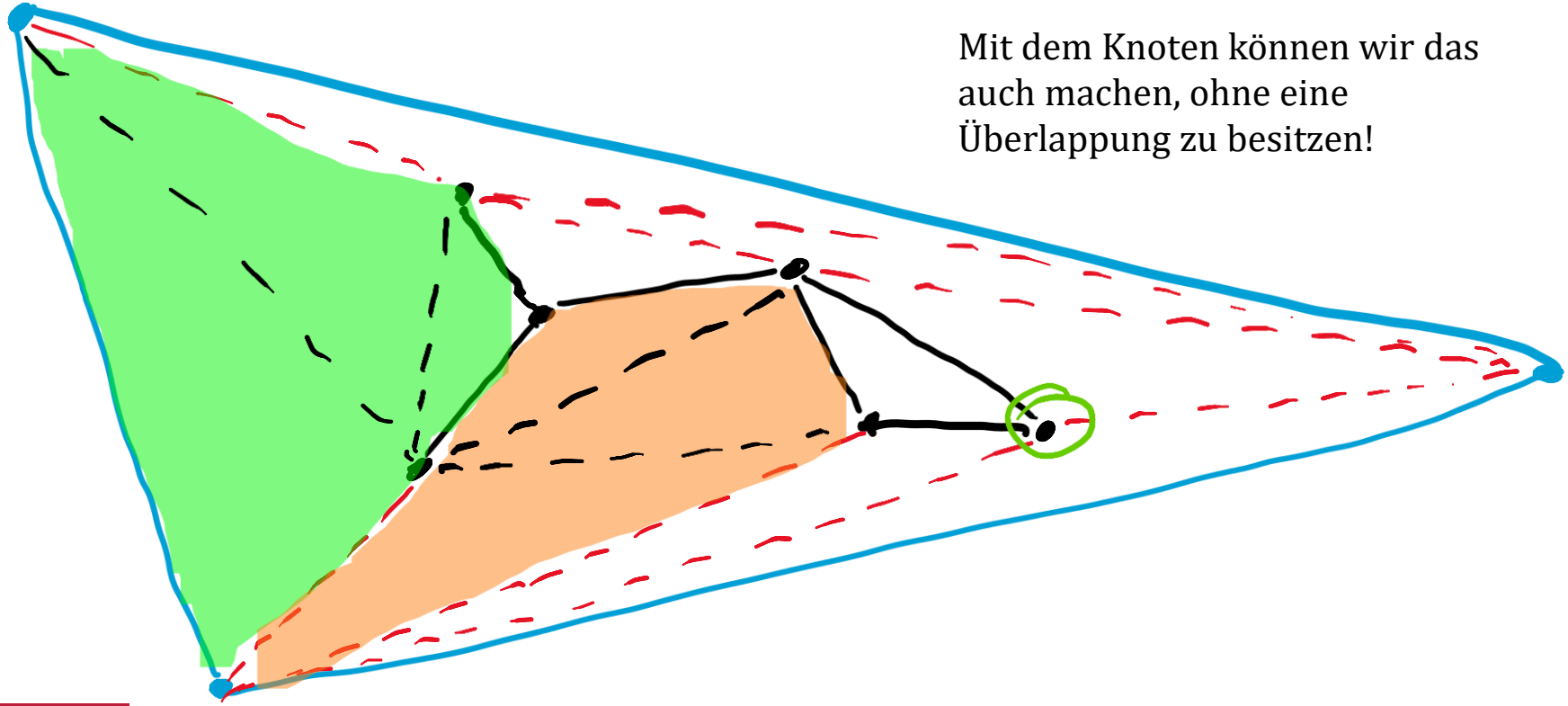
Wie teste ich, ob der Punkt in der grünen Fläche liegt?
→ Trianguliere erneut!

# Simplifizieren



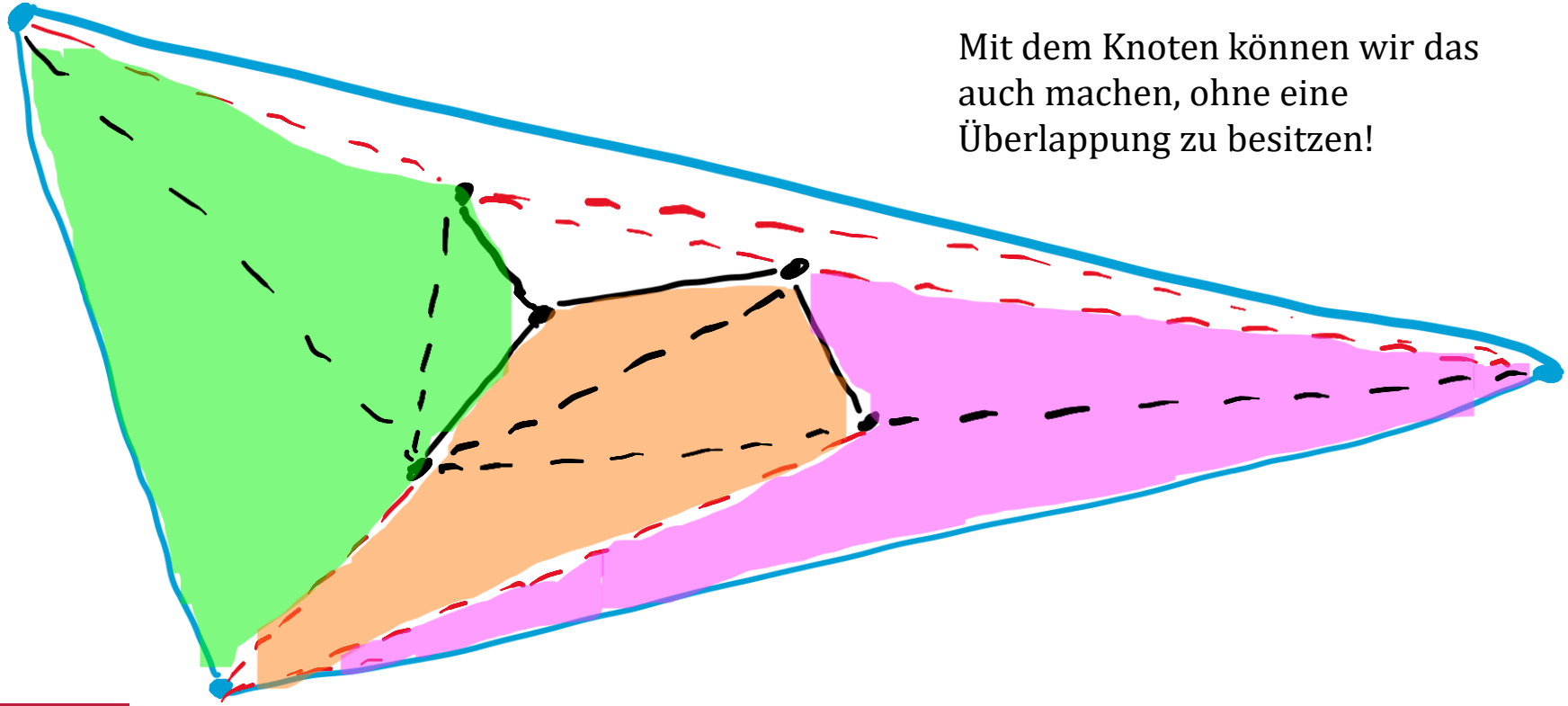
Mit dem Knoten können wir das auch machen, ohne eine Überlappung zu besitzen!

# Simplifizieren



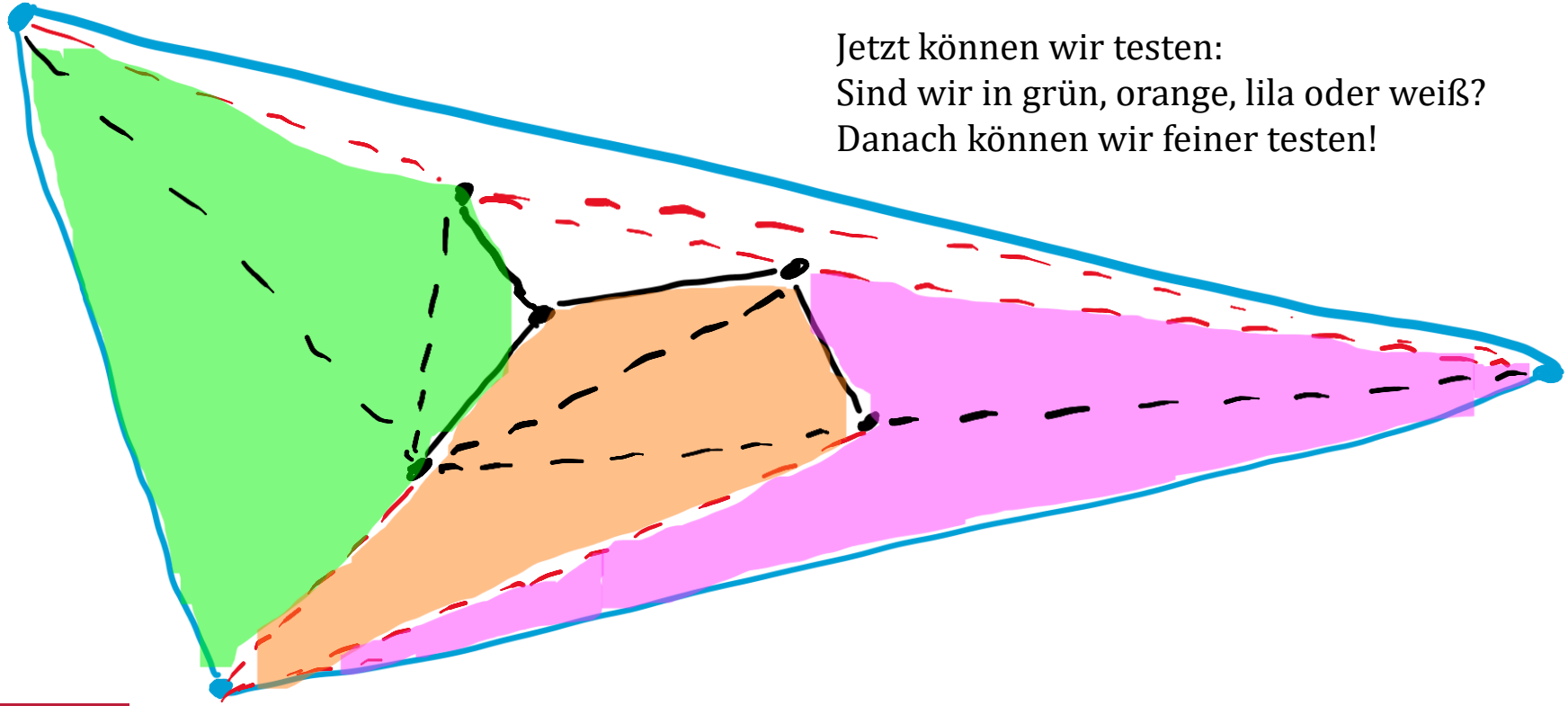
Mit dem Knoten können wir das auch machen, ohne eine Überlappung zu besitzen!

# Simplifizieren



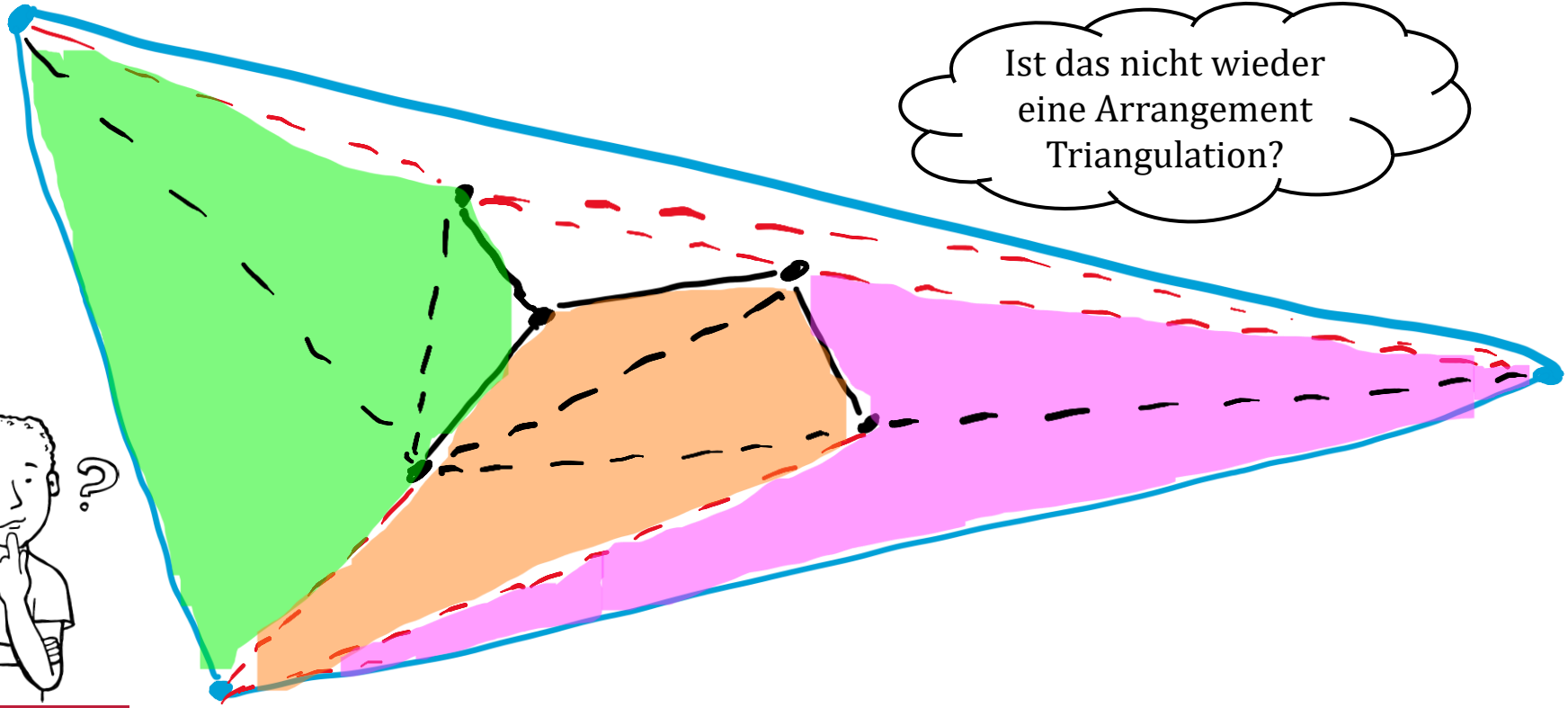
Mit dem Knoten können wir das auch machen, ohne eine Überlappung zu besitzen!

# Simplifizieren



Jetzt können wir testen:
Sind wir in grün, orange, lila oder weiß?
Danach können wir feiner testen!

# Simplifizieren



# Wrap-Up

Nach dem **Simplifizieren** haben wir wieder eine Triangulation → Wiederhole Vorgehen!

Wir können **„unabhängige“** Teile gleichzeitig simplifizieren

Wenn wir ein Dreieck in der Simplifizierung identifiziert haben…

…in welchen Dreiecken müssen wir in der feineren Auflösung weiter suchen?

# Unabhängige Mengen

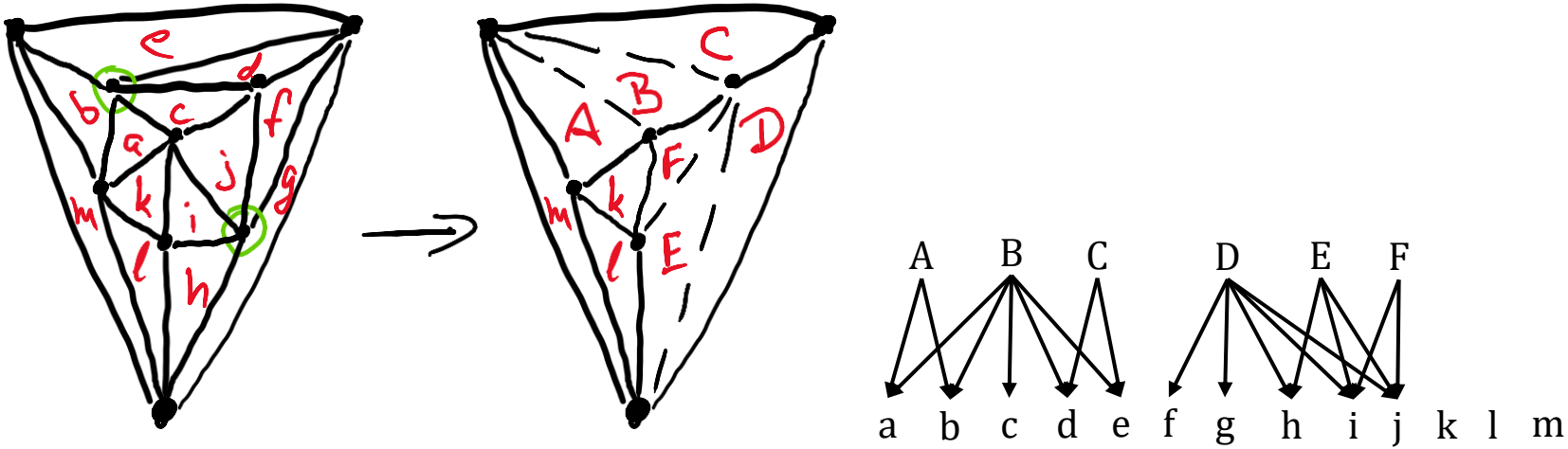
Beobachtung:
Wir können die Triangulation parallel vereinfachen, wenn die zugehörigen Knoten nicht verbunden sind.

**Definition 2.31**
Für ein Graph $G = (V, E)$ ist $I \subseteq V$ genau dann eine unabhängige Menge, wenn $\{u, v\} \notin E$ für je zwei Knoten $u, v \in I$ gilt.

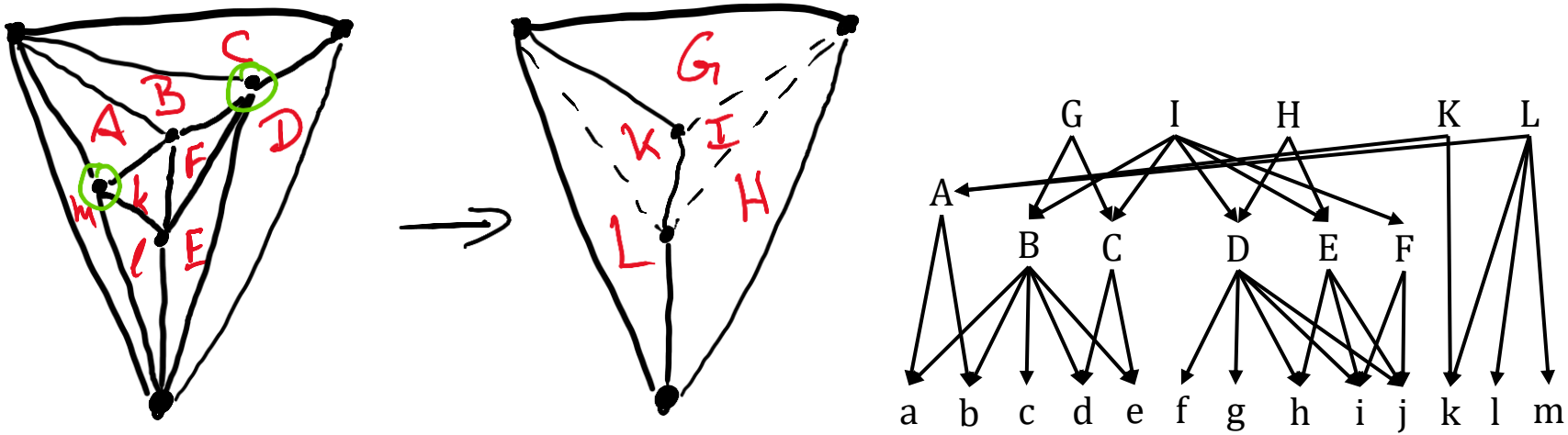
Das INDEPENDENT SET PROBLEM sucht nach einer größten unabhängigen Menge.

Das Problem ist allerdings NP-schwer! Wir werden später analysieren, dass wir effizient "genügend große" Mengen finden können.

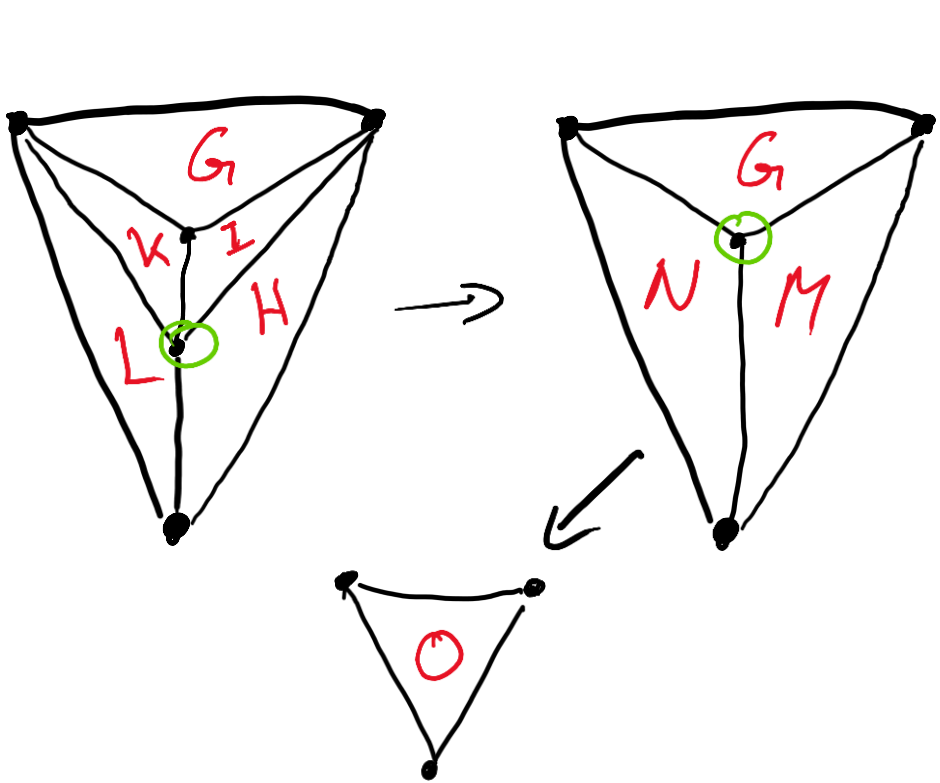
# Hierarchie von Simplifizierungen

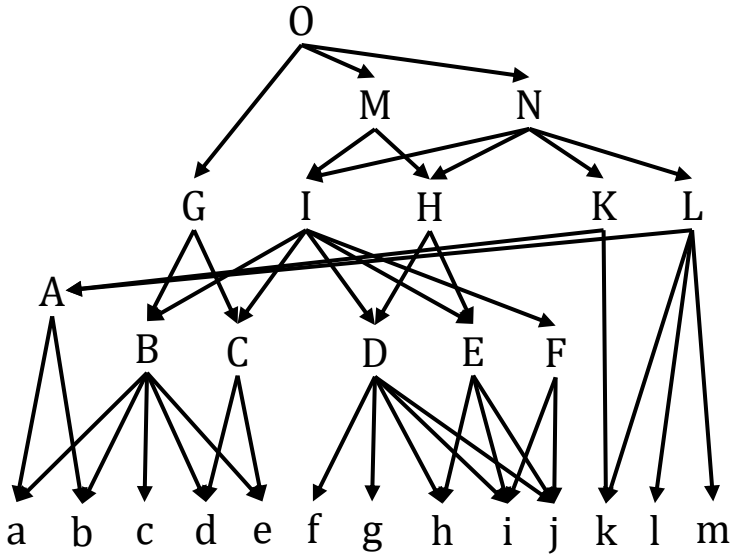


# Hierarchie von Simplifizierungen



# Hierarchie von Simplifizierungen



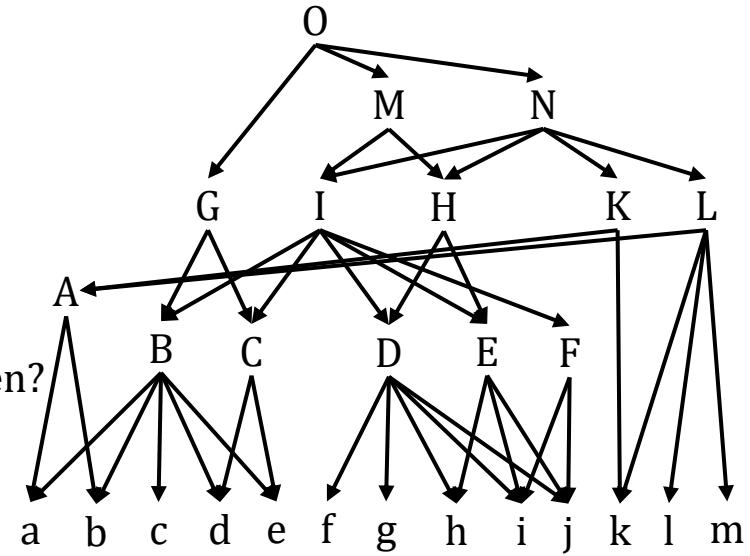



# Hierarchie von Simplifizierungen

Wir müssen bei einer Query nur von oben einen Pfad zu einem Originalen Dreieck finden!
Alle Tests sind mit Dreiecken, also sehr einfach.

Was müssen wir jetzt noch tun?

- Wie können wir **logarithmische Höhe** garantieren?
- Reicht das für logarithmische Laufzeit?
- **Verzweigungsgrad** ist auch entscheidend!




Also:
Wie definieren wir die Hierarchie und wie können wir sowohl logarithmische Höhe und konstanten Verzweigungsgrad garantieren?

# Datenstruktur der Hierarchie

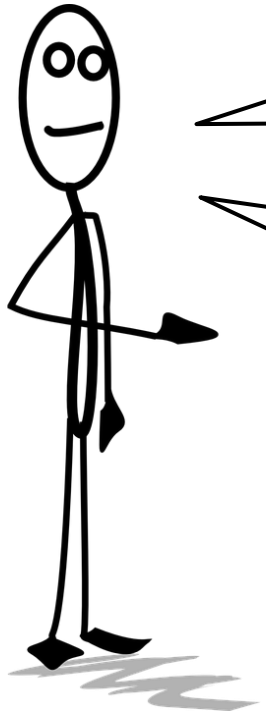
**Definition 2.32 (Kirkpatricks Hierarchie)**
Sei $\mathcal{A}$ ein Arrangement, $T(\mathcal{A})$ dessen Triangulation und $T_0, T_1, \dots, T_k$ eine Folge von Triangulationen mit folgenden Eigenschaften.

- $T_0 \coloneqq T(\mathcal{A})$
- $T_{i+1}$ entsteht aus $T_i$ durch Löschen unabhängiger Knoten und Triangulieren freien Flächen.
- $T_k$ ist ein Dreieck

Gespeichert wird die Hierarchie mit einem gerichteten, azyklischen Graphen (kurz **DAG**):

- Jedes Dreieck aus $T_i$ entspricht einem Knoten.
- Es existiert eine Kante von einem Dreieck $\Delta_1 \in T_{i+1}$ zu einem Dreieck $\Delta_2 \in T_i$, wenn $\Delta_1 \cap \Delta_2 \neq \emptyset$

# Effiziente Umsetzung



Was wir jetzt zeigen werden:

- Es existiert eine unabhängige Menge der Größe $\frac{n}{18}$, in welchem jeder Knoten maximal Grad 8 besitzt.

Mit dieser Eigenschaft folgt:

- Höhe des DAGs ist $O(\log n)$.
- Eine Ebene zu bearbeiten, dauert $O(1)$ Zeit.

# Größe der Datenstruktur

**Lemma 2.33**
Jeder triangulierte planare Graph mit $n \geq 4$ Knoten besitzt eine unabhängige Knotenmenge der Größe $\frac{n}{18}$, in welchem jeder Knoten Grad $\leq 8$ besitzt. Eine solche Menge kann in Zeit $O(n)$ berechnet werden.

Als Algorithmus genügt ein einfacher Greedy-Algorithmus:

- Betrachte jeden Knoten mit Grad maximal 8.
- Falls Knoten aufgenommen werden kann, nimm ihn auf.

Für jeden triangulierten planaren Graphen auf $n$ Knoten ist $|E| = 3n - 6$.
→ die Gradsumme aller Knoten $6n - 12 < 6n$.
→ Es existieren mind. $\frac{n}{2}$ viele Knoten mit Grad $\leq 8$.
Im Algorithmus reduziert sich die Knotenmenge immer um $\leq 9$ pro aufgenommenen Knoten.
→ $\frac{n}{18}$ viele Knoten können aufgenommen werden.

# Analyse

**Theorem 2.34**
Sei $\mathcal{A}$ ein Arrangement. Dann gelten folgende Aussagen.
1. Die Platzkapzität der Hierarchie ist $O(n)$.
2. Die Query-Zeit für das Punktlokalisierungsproblem mit der Hierarchie benötigt $O(\log n)$ Zeit.
3. Die Kirkpatrick Hierarchie kann in Zeit $O(n \log n)$ konstruiert werden.

**Größe Datenstruktur:**
#Knoten + #Kanten.
Da Verzweigungsgrad maximal 8 ist, reicht die Analyse der Knotenzahl:
Jedes Simplifzieren reduziert die Knotenzahl im Arrangement um $\frac{n}{18}$, d.h. wir besitzen insgesamt:

$$O\left(n + \frac{17}{18}n + \left(\frac{17}{18}\right)^2 n + \ \ldots\right) \subseteq O\left(\frac{1}{1 - \frac{17}{18}}n\right) = O(n)$$

# Analyse

**Theorem 2.35**
Sei $\mathcal{A}$ ein Arrangement. Dann gelten folgende Aussagen.
1. Die Platzkapzität der Hierarchie ist $O(n)$.
2. Die Query-Zeit für das Punktlokalisierungsproblem mit der Hierarchie benötigt $O(\log n)$ Zeit.
3. Die Kirkpatrick Hierarchie kann in Zeit $O(n \log n)$ konstruiert werden.

**Query-Zeit**
Es kann nur $O(\log n)$ Ebenen in der Datenstruktur geben. Durch konstanten Verzweigungsgrad folgt die angegebene Laufzeit.

# Analyse

**Theorem 2.36**
Sei $\mathcal{A}$ ein Arrangement. Dann gelten folgende Aussagen.
1. Die Platzkapzität der Hierarchie ist $O(n)$.
2. Die Query-Zeit für das Punktlokalisierungsproblem mit der Hierarchie benötigt $O(\log n)$ Zeit.
3. Die Kirkpatrick Hierarchie kann in Zeit $O(n \log n)$ konstruiert werden.

**Pre-Processing**
Das Triangulieren eines Arrangements auf $n$ Knoten benötigt $O(n \log n)$ Zeit.
Damit ist die Zeit zum Triangulieren jeder Ebene insgesamt

$$\sum_{i=1}^{k} n_i \log n_i \leq \log n \sum_{i=1}^{k} n_i \in O(n \log n)$$

# Nächste Woche

Gibt es andere Möglichkeiten?

Was passiert, wenn wir mehr Informationen haben wollen?

Wo im outer face sind wir? Eher links / unten / rechts / oben?