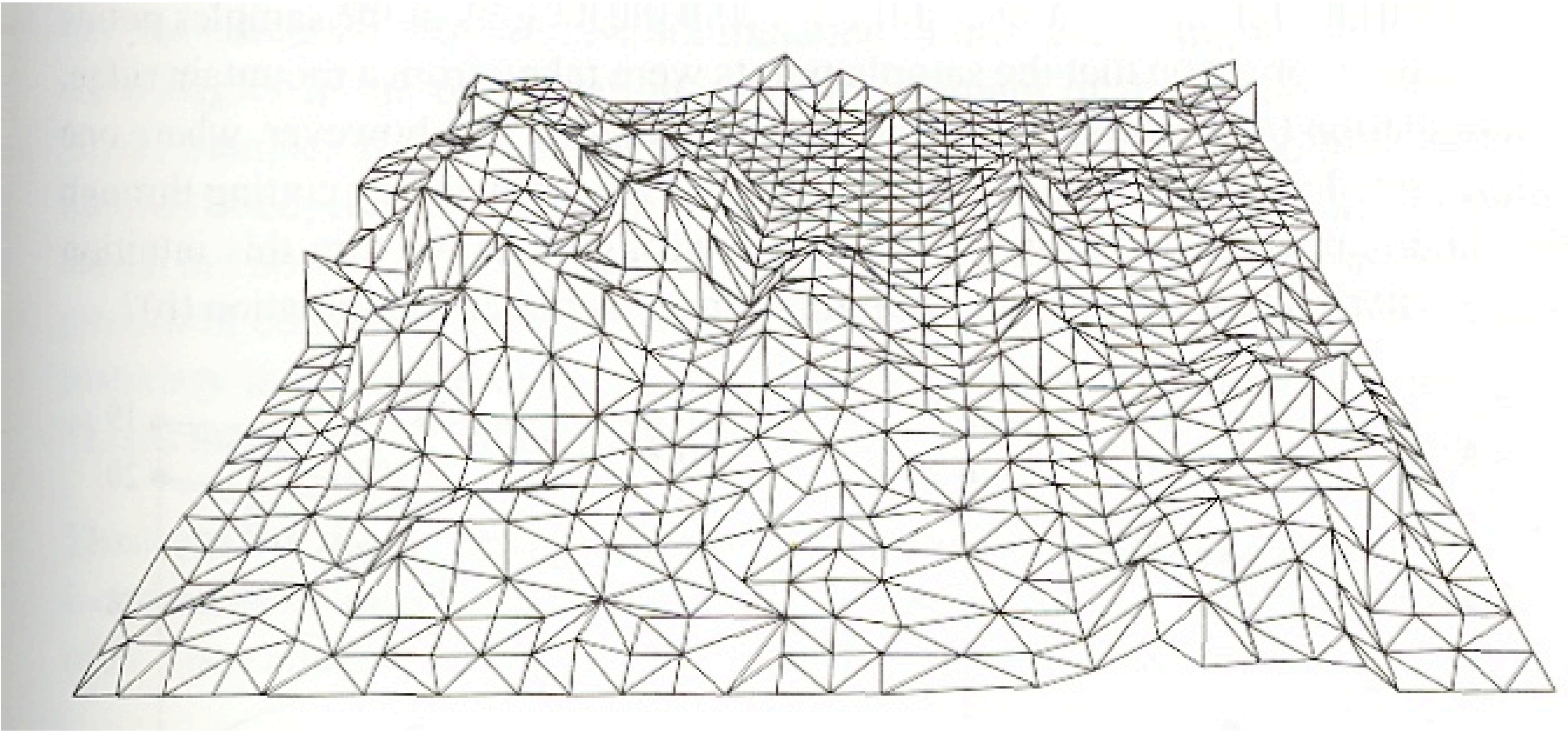
# Algorithmische Graphentheorie

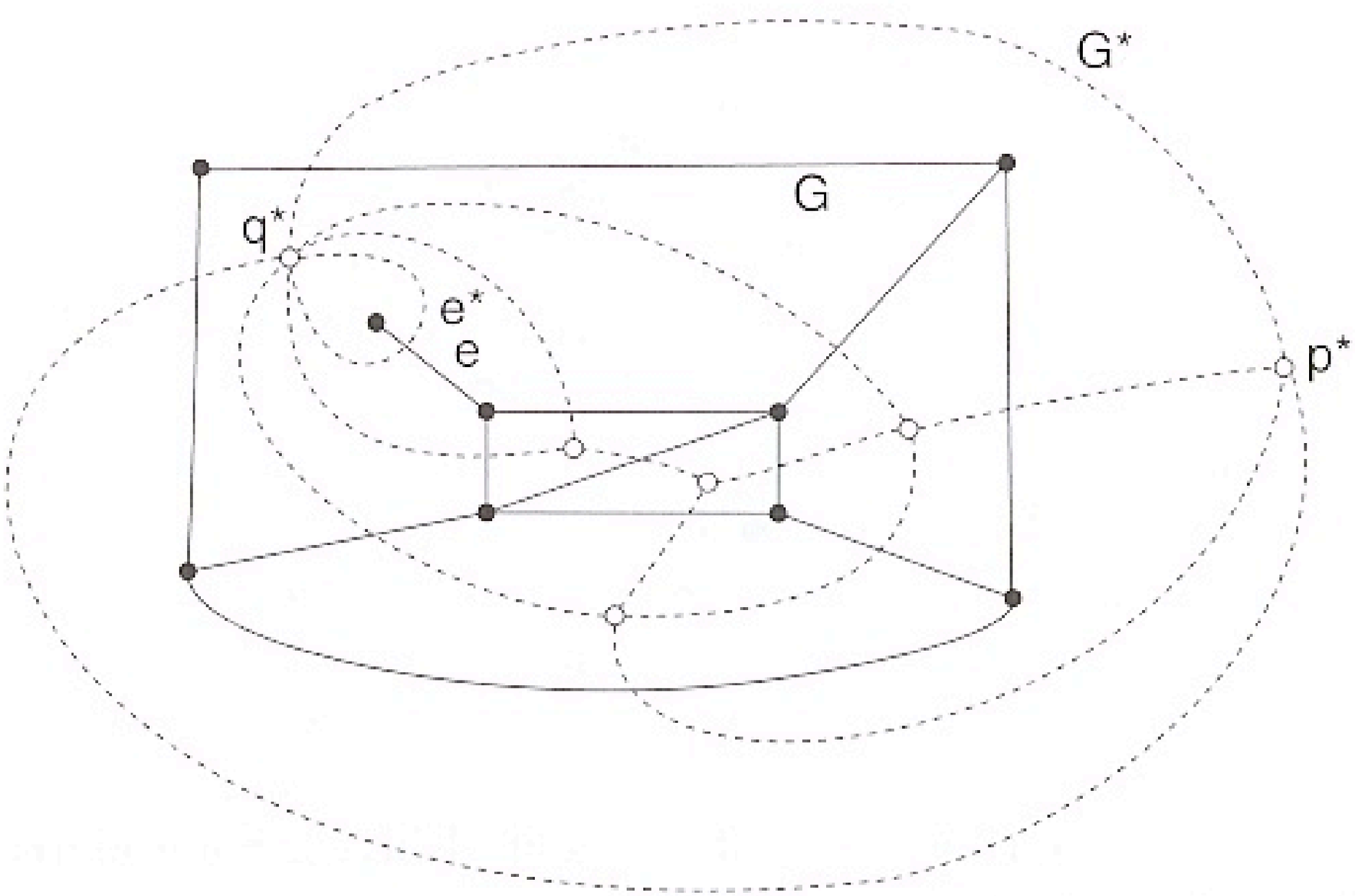
Teil 3

# *Delaunay -Triangulation*



# Dualer Graph

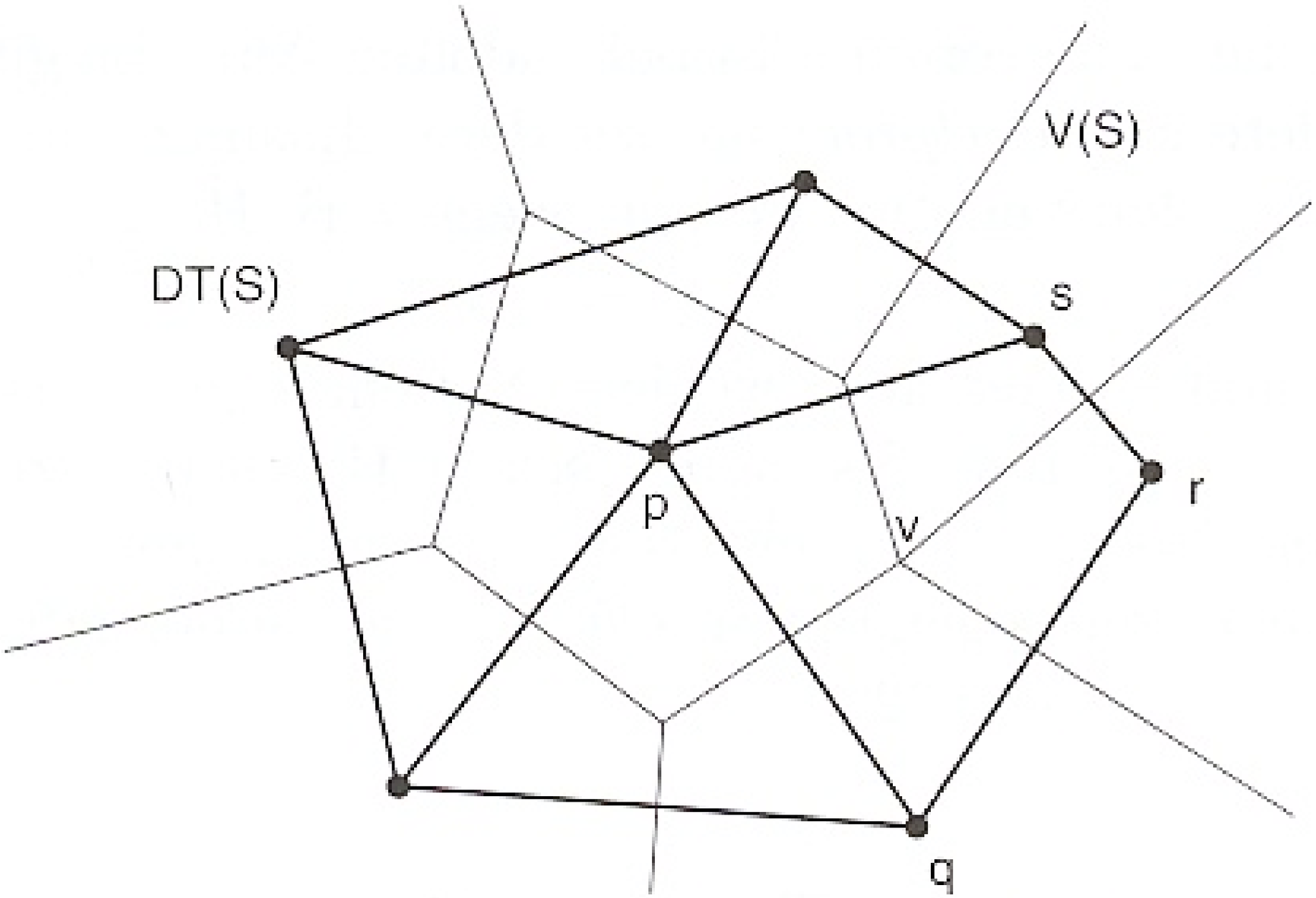
Zu einem kreuzungsfreien, nichtleeren zusammenhängenden Graphen $G$ auf der Kugeloberfläche ist der duale Graph $G^*$ einer, der zu jeder Fläche von $G$ einen Knoten besitzt, und eine (nicht notwendig gerade!) Kante von Knoten $p$ zu Knoten $q$ genau dann, wenn diese in $G$ eine einzig Kante kreuzt!



# ***Delaunay-Zerlegung*** ist dualer Graph des Voronoi Diagramms!

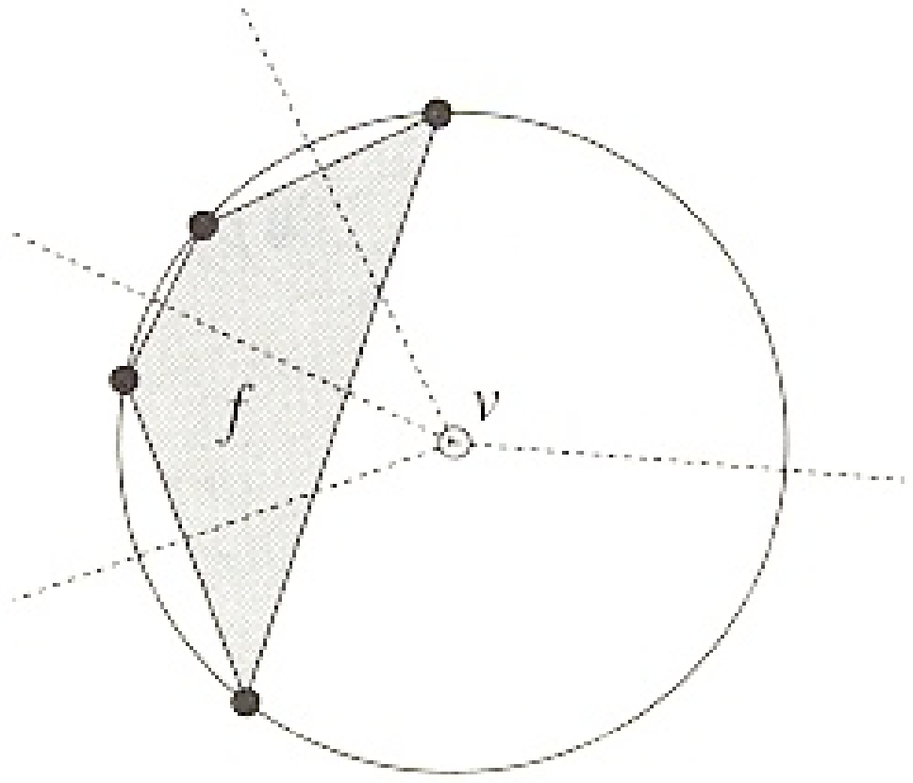
**Definition:**

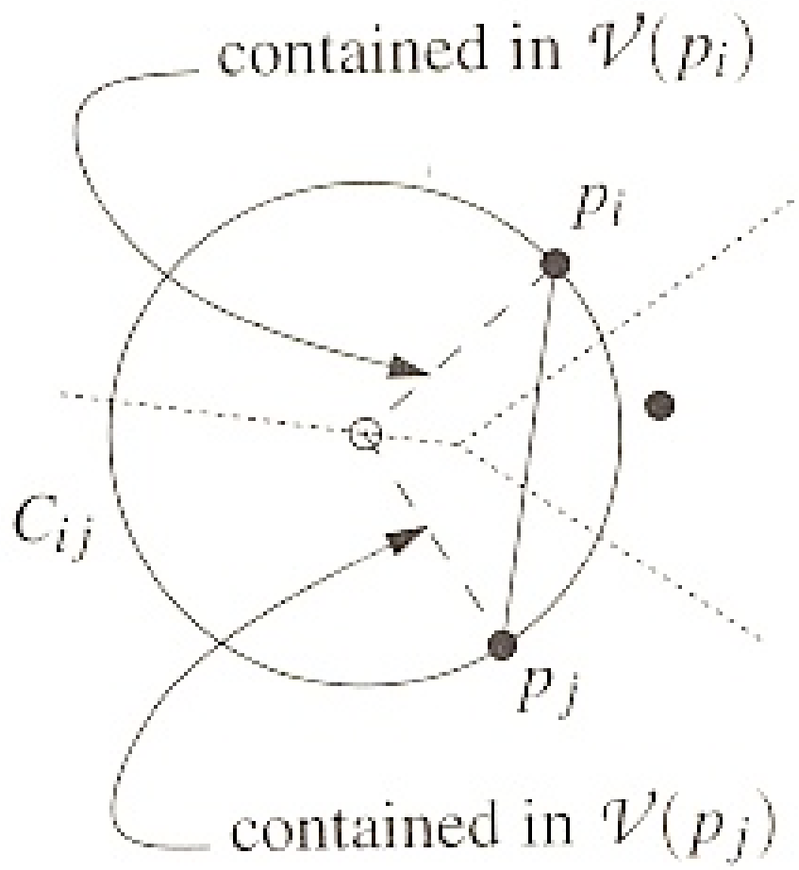
Die ***Delaunay-Zerlegung*** mit der Knotenmenge $S$ ist der duale Graph des Voronoi Diagramms zur selben Punktmenge.



# einfache Eigenschaften 1. Teil

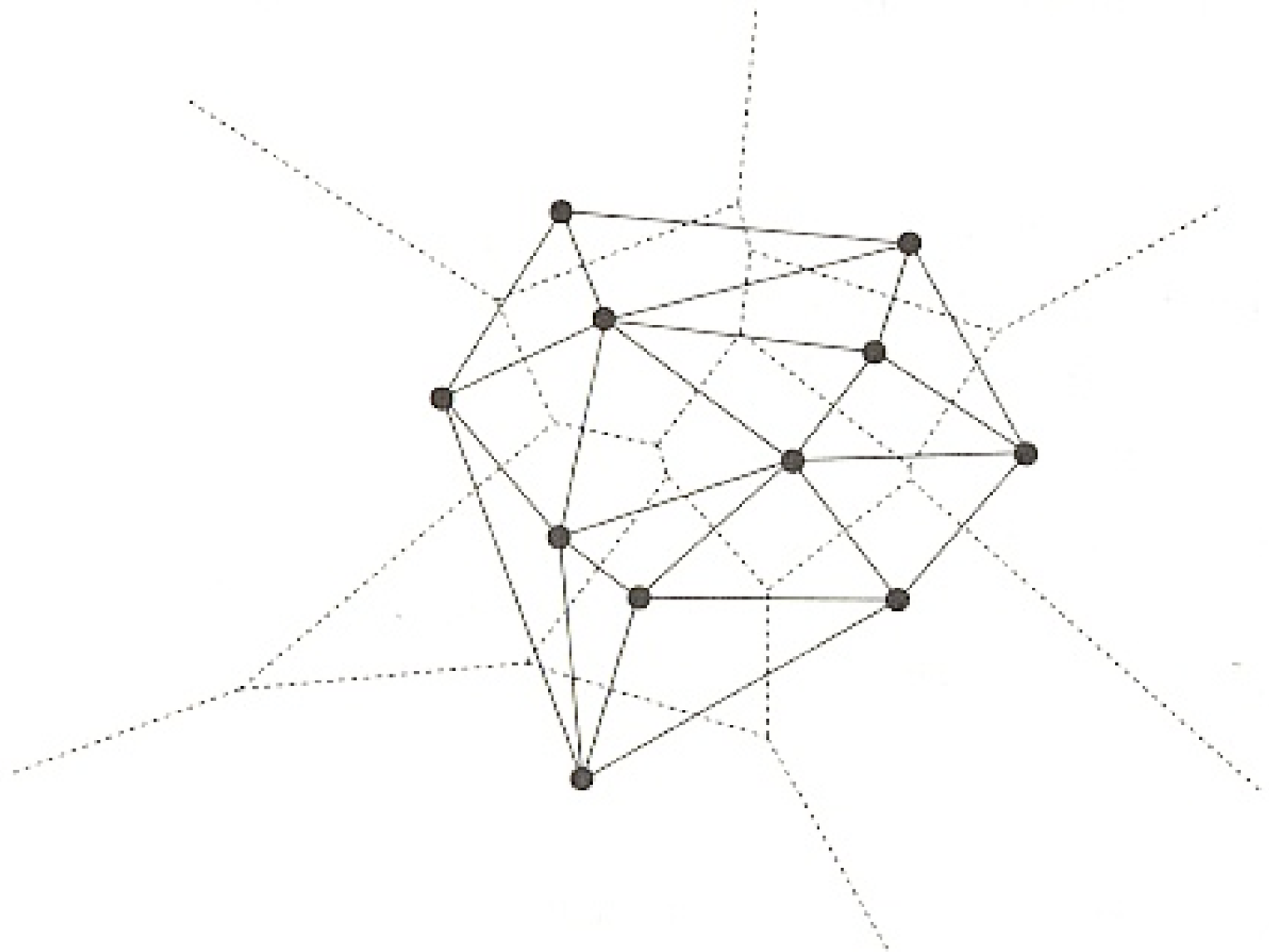
Zwei Punkte $p, q \in S$ sind genau dann durch eine Delaunay-Kante verbunden, wenn es einen Kreis gibt, auf dem ausschließlich $p$ und q liegen und keinen Punkt aus $S$ in seinem Inneren!




Der Mittelpunkt dieses Kreises liegt auf der Bisektionsgeraden von p und q, ist aber näher an diesen, als der Voronoi-Knoten, der zu den Voronoi-Regionen $V(p,S)$ und $V(q,S)$ gehört. Warum sind oben rechts $p_i$ und $p_j$ keine Voronoi-Knoten?

# einfache Eigenschaften 2. Teil



Wie man oben sieht, kann eine Delaunay-Kante mehr als nur eine Voronoi-Region durchqueren!

# Ist jede Delaunay-Zerlegung eine Triangulation?

Wenn die Voronoi-Punkte alle auf einem Kreis liegen, so ist der Voronoi-Knoten dessen Mittelpunkt, und die Delaunay-Zerlegung ist ein Polygon und keine Triangulation!

Wenn die Voronoi-Punkte alle auf einer Geraden liegen, so ist die Delaunay-Zerlegung ein Kantenzug und ebenfalls keine Triangulation!

Wenn von den Voronoi-Punkten der Menge $S$ keine $4$ auf einem Kreis und keine 3 auf einer Geraden liegen, so haben alle Voronoi-Knoten im Voronoi-Diagramm den Grad 3 und die Delaunay-Zerlegung ist tatsächlich eine Triangulation (die begrenzen Flächen sind Dreiecke). Diese nennen wir ***Delaunay-Triangulation***.

***Bemerkung:*** Delaunay ist nicht der französische Maler, sonder war der Mathematiker: Boris Nicolaevich Delone (Weil deutsch und französisch früher die Wissenschaftssprache war blieb der Name Delaunay bestehen)

# Konstruktion der Delaunay-Zerlegung(-Triangulation)

**Theorem:**
Zu einer Menge $S$ von $n := |S|$ Punkten der euklidischen Ebene kann eine Delaunay-Zerlegung in $O(n \cdot \ell og(n))$ Zeit- und O(n) Platzbedarf konstruiert werden.

**Beweis:**
Man konstruiere das Voronoi-Diagramm zu $S$ innerhalb dieser Komplexitätsschranken. Aus diesem kann die Delaunay-Zerlegung in linearem Aufwand generiert werden!

***Beobachtung:***
Wenn die konvexe Hülle von $S$ ($n := |S|$) genau $r$ Ecken hat, dann hat jede Triangulation von $S$ genau $2(n-1)-r$ viele Dreiecke.

Bitte zeigen Sie dies!

# Beweis der Beobachtung

***1. Version:***

Sei $d$ die Anzahl aller Dreiecke der Triangulation von $S$. Dann sind mit $3d$ alle Kanten im Inneren doppelt gezählt, nur die $d$ Stücke des äußeren Randes nicht, d.h $2e = 3d+r$.
Die Euler'sche Formel sagt aber

$$n-e+(d+1) = 2 \text{ oder } n-2+(d+1) = e$$

und Einsetzen ergibt:

$$3d+r = 2[n-2+(d+1)] = 2n-2+2d \text{ und}$$
$$d = 2(n-1)- r.$$

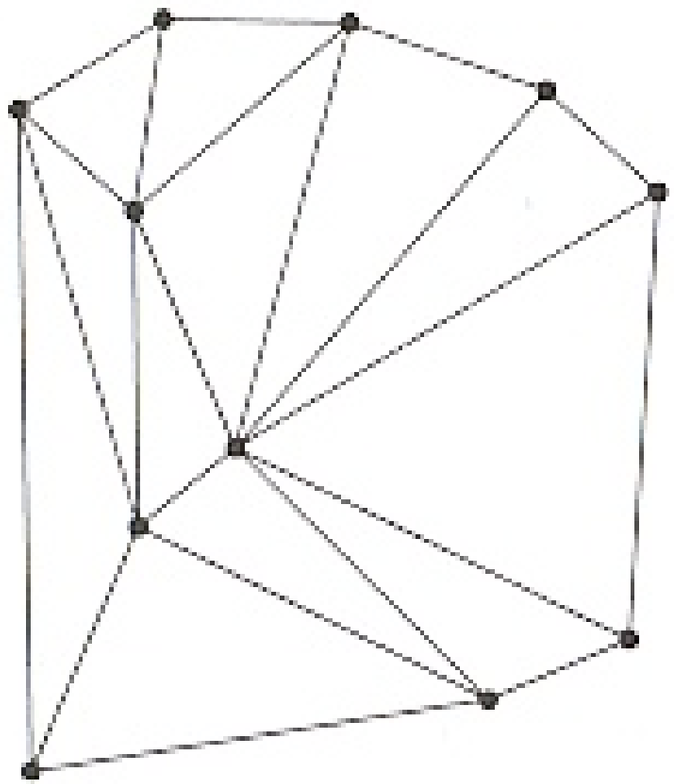
***qed.***

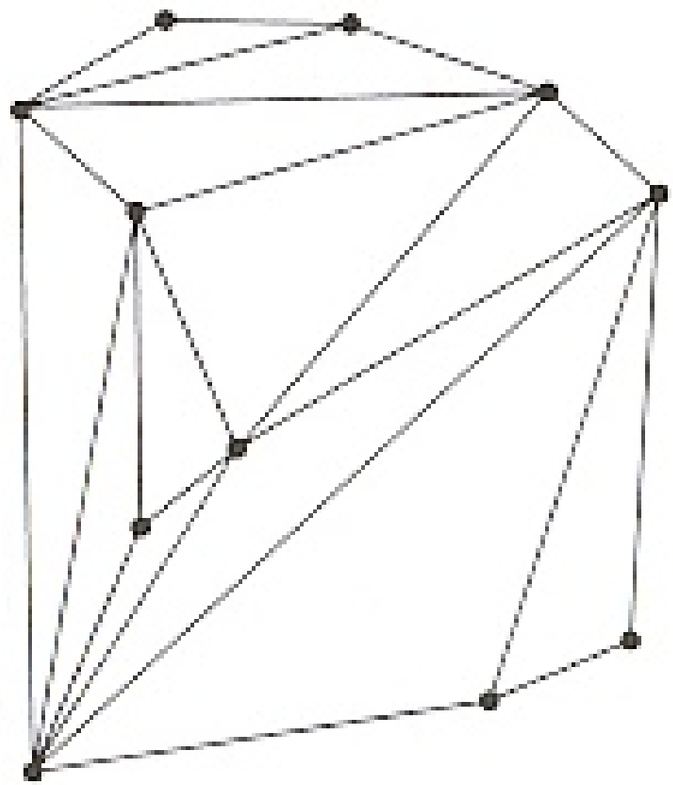
***2.Version:***

Induktion über $n \geq 3$:
Basis: $n= 3 \rightarrow r=3$ und $d=4- 3$■
Induktionschritt: $n \mapsto n+1$: a) Neuer Knoten im Inneren eines Dreieckes bedeutet: $r$ ändert sich nicht, $d \mapsto d+2 = [2(n-1)- r]+2 = 2n-r$.■
b) Neuer Knoten auf der konvexen Hülle, bedeutet $r \mapsto r+1$ und $d \mapsto d+1 = [2(n-1)-r]+1 = 2n-r-2+1 = 2n-(r+1)$.■

***qed.***

# Maximalität des kleinsten Winkels





Die rechte Triangulation hat sehr spitze Winkel, die linke ist dagegen “ausgeglichener”. Letzteres ist oft erwünscht und wird tatsächlich optimal durch die Delaunay-Triangulation gewährleistet!

**Definition:**
Für eine Triangulation $T$ einer Punktmenge $S$ sei $w(T) := (\alpha_1, \alpha_2, \ldots, \alpha_{3d})$ die aufsteigend sortierte Folge der Innenwinkel aller Dreiecke. Zwei solche Folgen werden bezüglich der lexikographischen Ordnung $<^{lex}$ miteinander verglichen.

# Maximalität des kleinsten Winkels

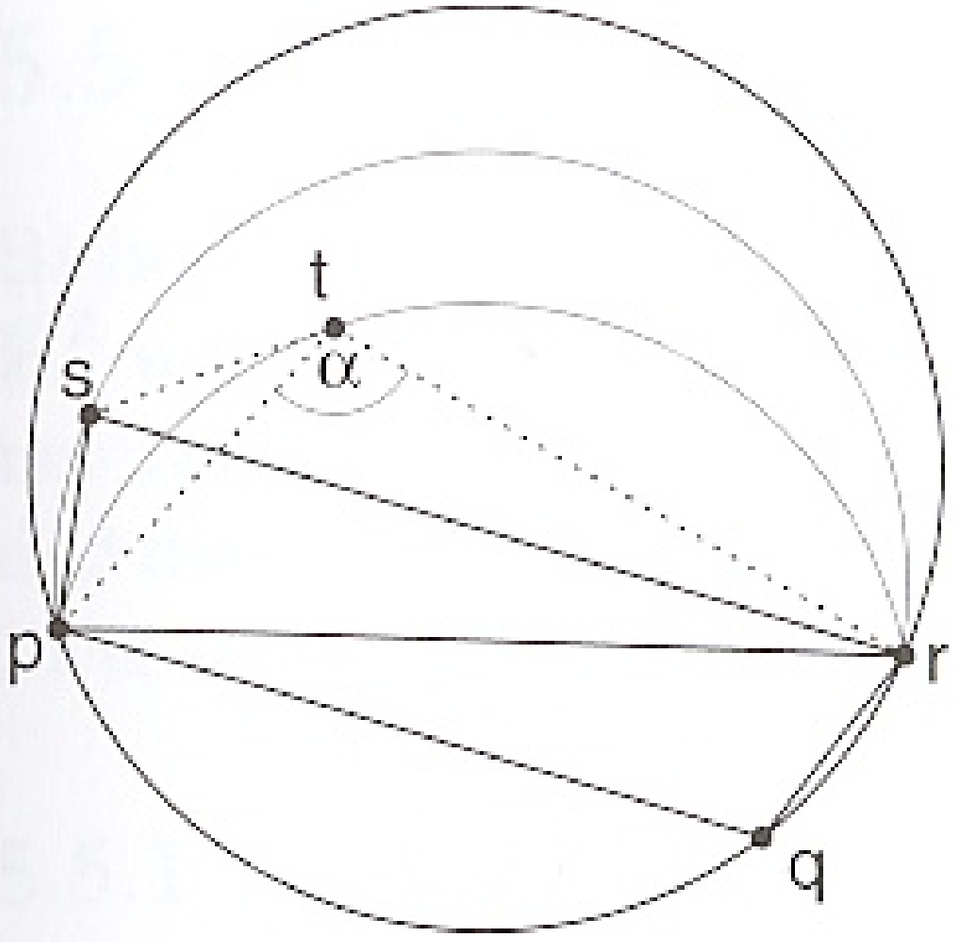
**Theorem:**
Sei $S$ eine Menge von $n := |S|$ Punkten der euklidischen Ebene, von denen weder 3 kollinear noch $4$ auf einem Kreis liegen. Dann hat die Delaunay-Triangulation unter allen Triangulationen von $S$ die größte Winkelfolge. Durch diese ist die Delaunay-Triangulation sogar eindeutig bestimmt.

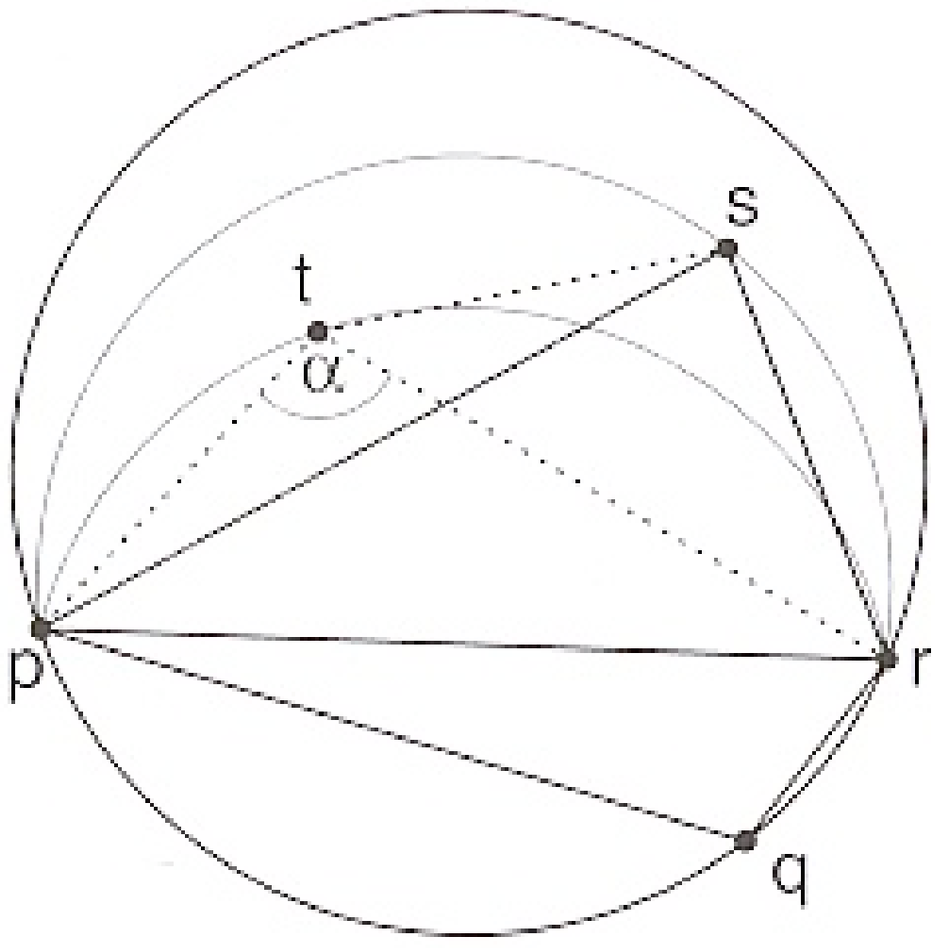
Zum Beweis wird die vorher bemerkte einfache Eigenschaft benötigt:

**Lemma:**
Drei Punkte $p, q, r \in S$ definieren genau dann ein Dreieck der Delaunay-Triangulation, wenn im Kreis durch diese $3$ Punkte kein Punkt aus $S$ liegt!

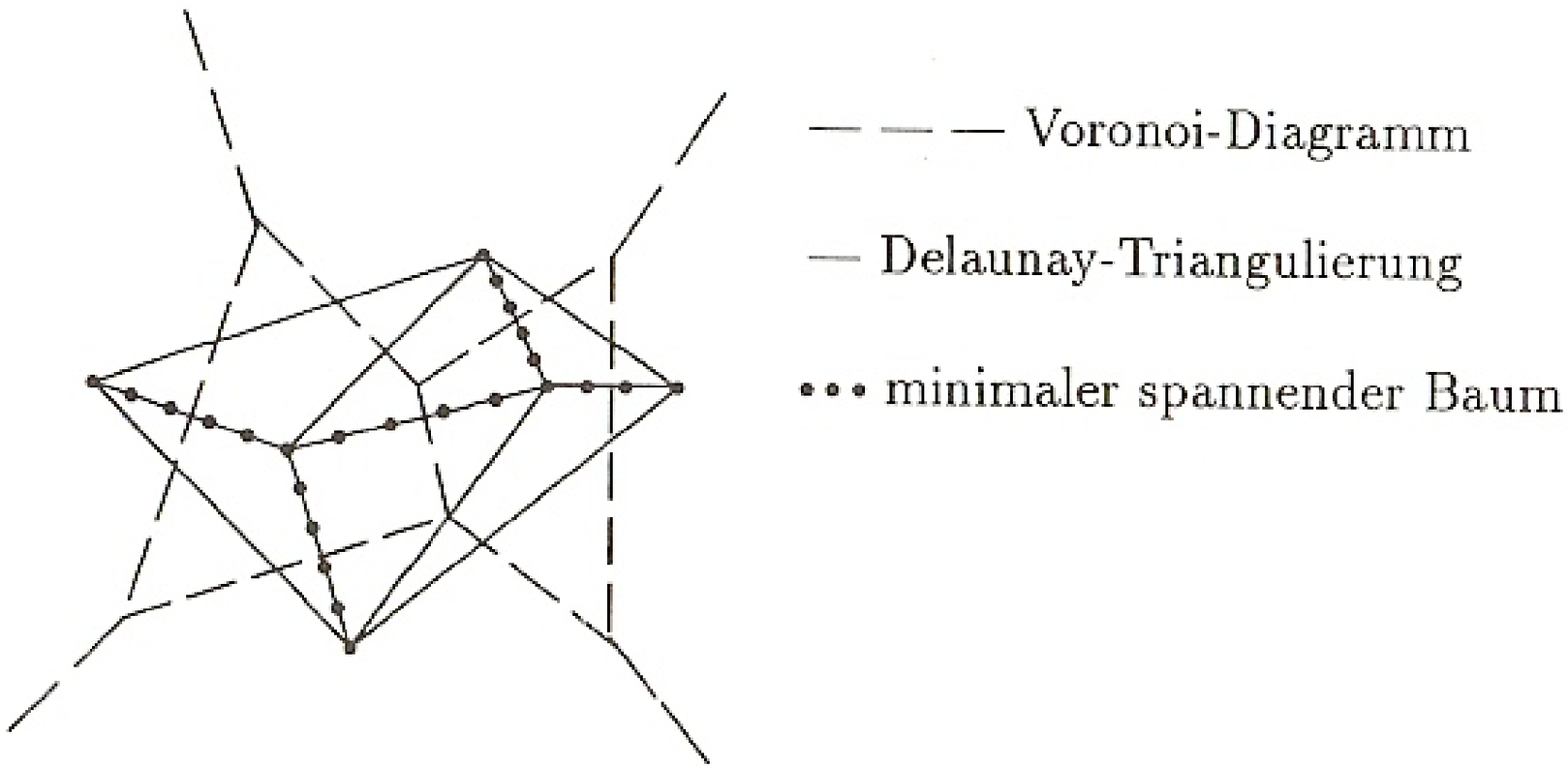
Der Beweis des Theorems benutzt gewöhnliche Geometrie (Satz von Thales: Die Winkelsumme im Dreieck zwischen einer Kreissehne und allen Punkten auf dem selben gegenüberliegenden Kreisbogen ist konstant). [Leicht verständliche und elementare Darstellung in Klein].

# Skizzen zum Beweis



# Delaunay-Triangulation und minimaler spannender Baum



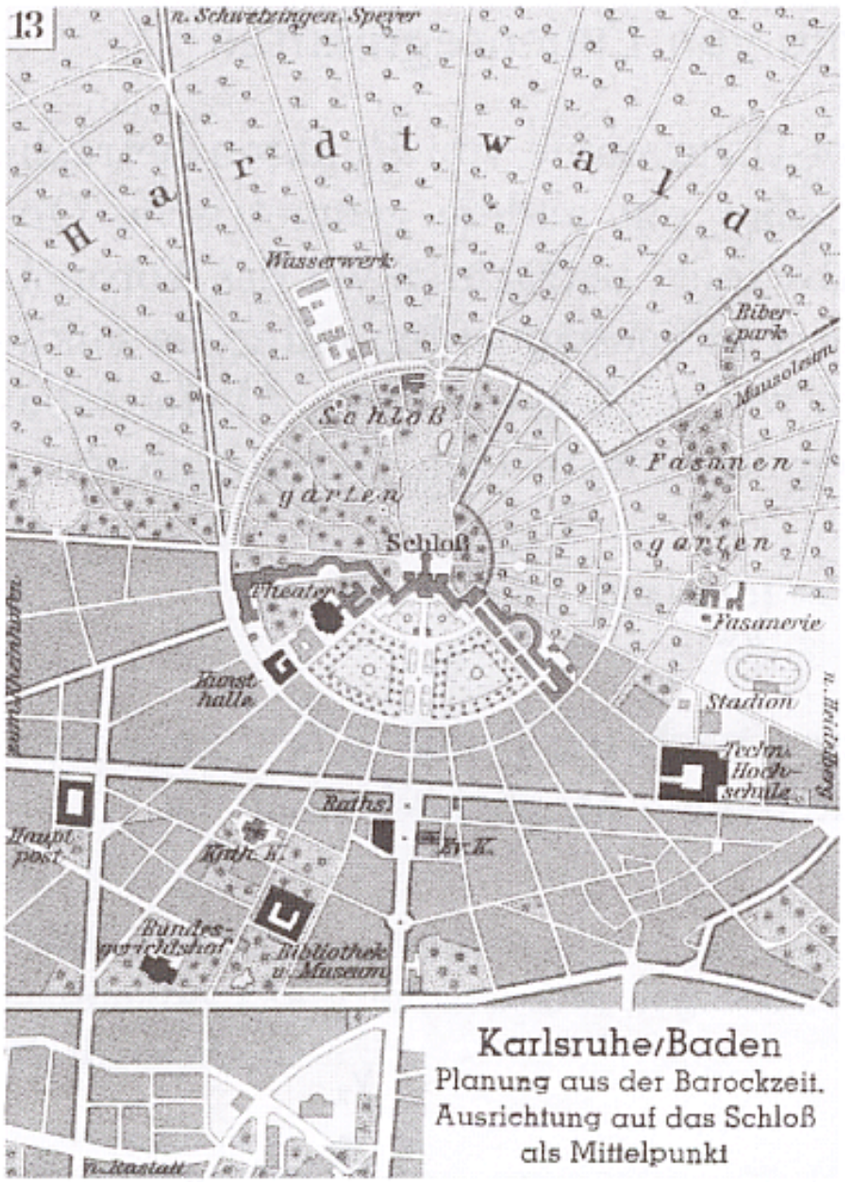

Die Delaunay-Triangulation kann benutzt werden, um einen minimalen spannenden Baum zu $n$ Punkten der Ebene in $O(n \cdot \ell og(n))$ Zeit zu finden:
Diejenigen Delaunay-Kanten die Voronoi-Kanten nur einmal schneiden sind die kürzesten Verbindungen zwischen den benachbarten Voronoi-Punkten!
Nach Erstellung der Delaunay-Triangulation muss man hier mit üblichen Verfahren das minimale Gerüst erstellen.

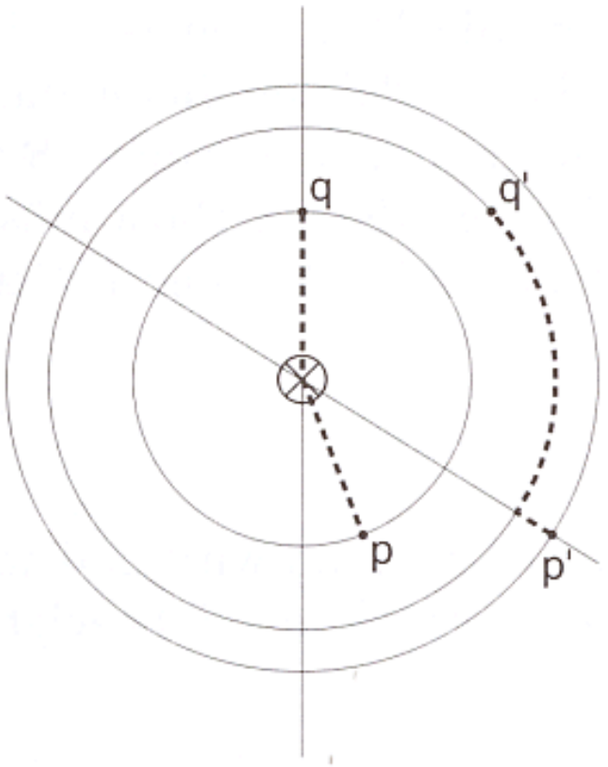
# Allgemeinerer Abstandsbegriff

Man kann neben den Minkowsky-Metriken:

$$L_i(p,q) := \sqrt[i]{|p_1 - q_1|^i + |p_2 - q_2|^i}$$

weitere konvexe Distanzfunktionen einführen. Man kann diese benutzen, wenn die euklidische Entfernung nicht passend ist:



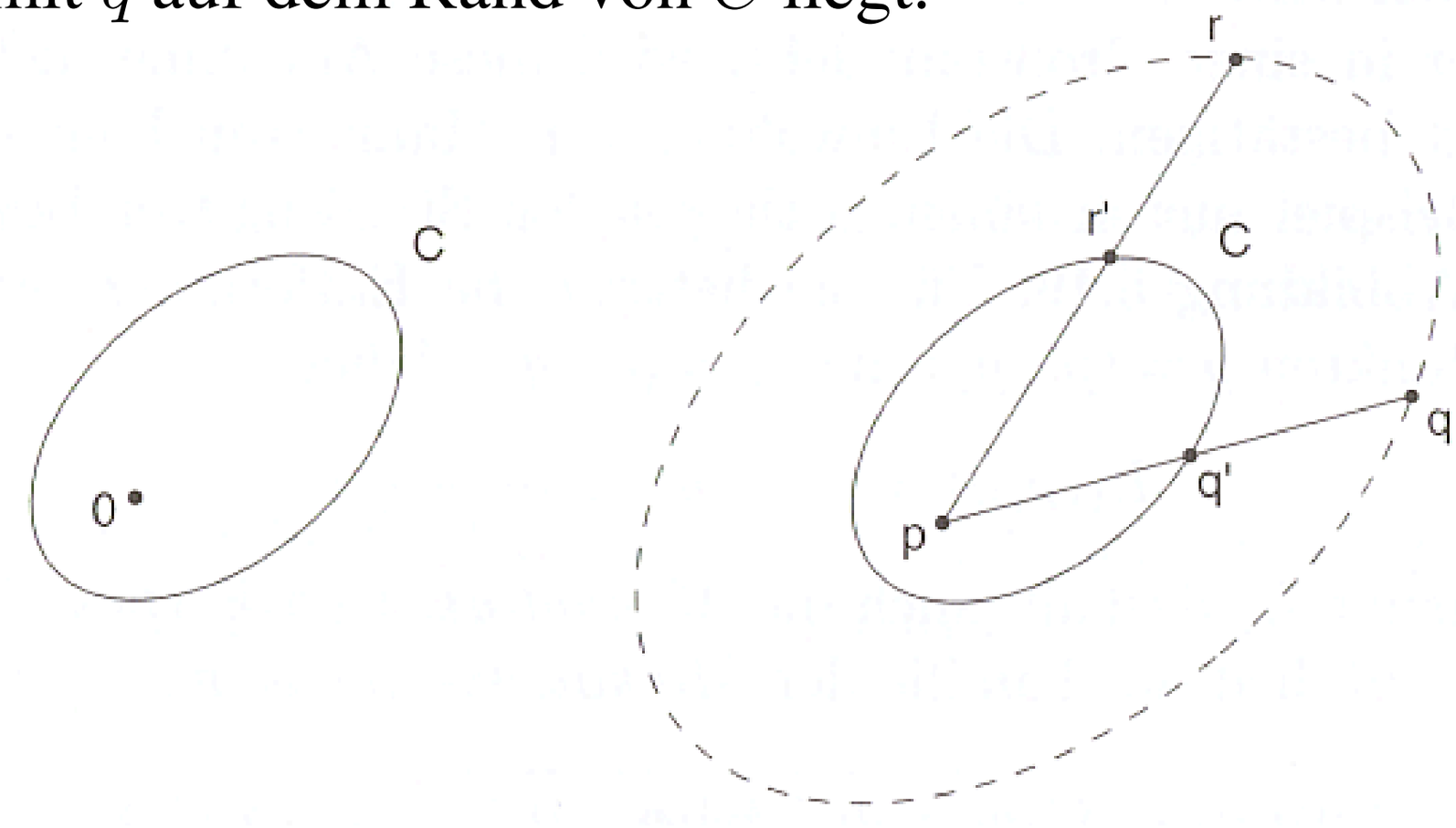

Zum Beispiel ist die Entfernung zwischen $p$ und $q$ in Karlsruhe nach Verschiebung an den Rand größer als in der Stadtmitte. Daher sind Varianten erforderlich.

# Konvexe Distanzfunktionen

Der Abstand $d_C(p,q)$ von $p$ nach $q$ bezüglich einer kompakten, konvexen Menge $C$ in der Ebene, die den Nullpunkt (das Zentrum von $C$) enthält, ist definiert als:

$$d_C(p,q) := \frac{|pq|}{|pq'|}$$

Das ist der Faktor, um den die nach $p$ verschobene Kopie von $C$ skaliert werden muss, damit $q$ auf dem Rand von $C$ liegt.



# Eigenschaften konvexer Distanzfunktionen

Es gilt immer:

- $d_C(p,q) \geq 0$,
- $d_C(p,q) = 0$ genau dann, wenn $p = q$, und
- $d_C(p,r) \leq d_C(p,r) + d_C(r,q)$.

Mit etwas Vorsicht kann man auch bei konvexen Distanzfunktionen Voronoi-Diagramme erklären und auch finden! Dort zeigt man:

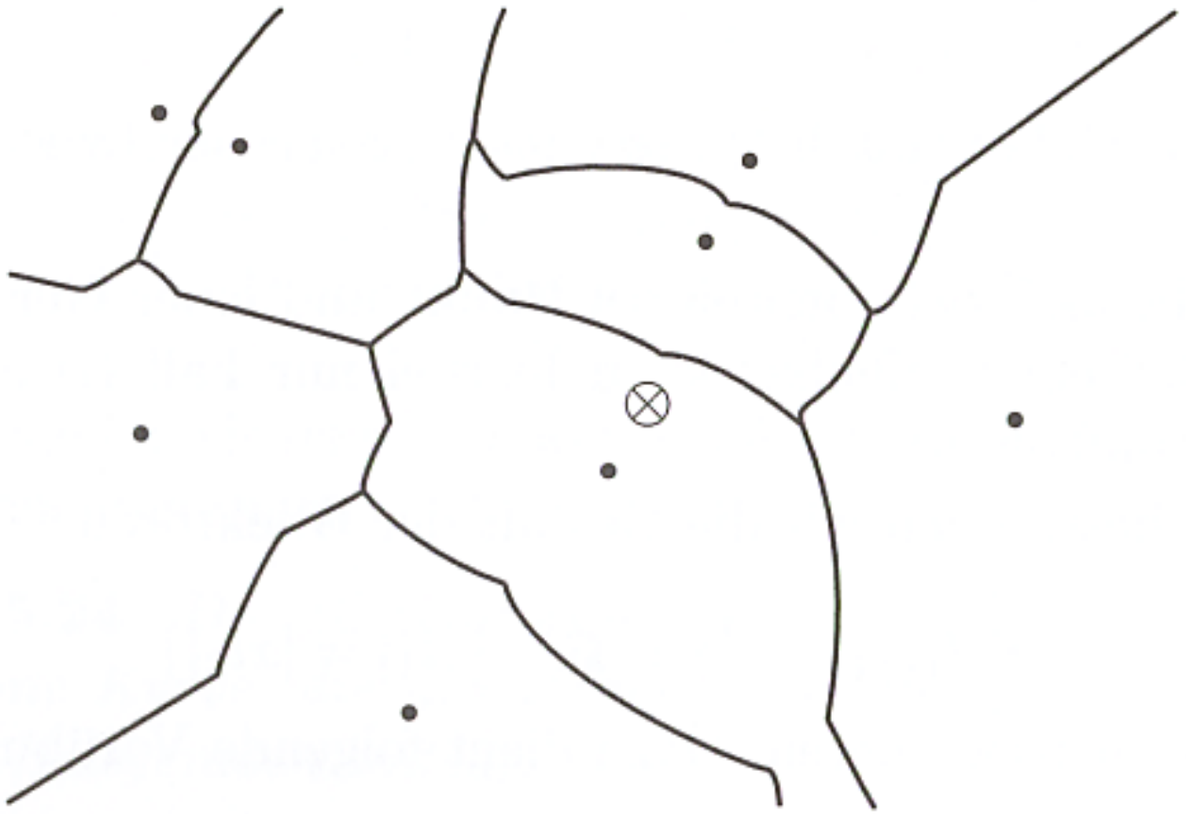
- Die Voronoi-Regionen sind nicht mehr konvex sondern "Sternförmig" und zusammenhängend,
- Zwei Voronoi-Regionen haben höchstens eine gemeinsame Kante,
- Bisektoren sind in der Regel gekrümmte Kurven

# Nicht immer taugen konvexe Distanzfunktionen

Die Ähnlichkeit zu den Voronoi-Diagrammen der euklidischen Ebene wird in höheren Dimensionen bei konvexen Distanzfunktionen FALSCH!

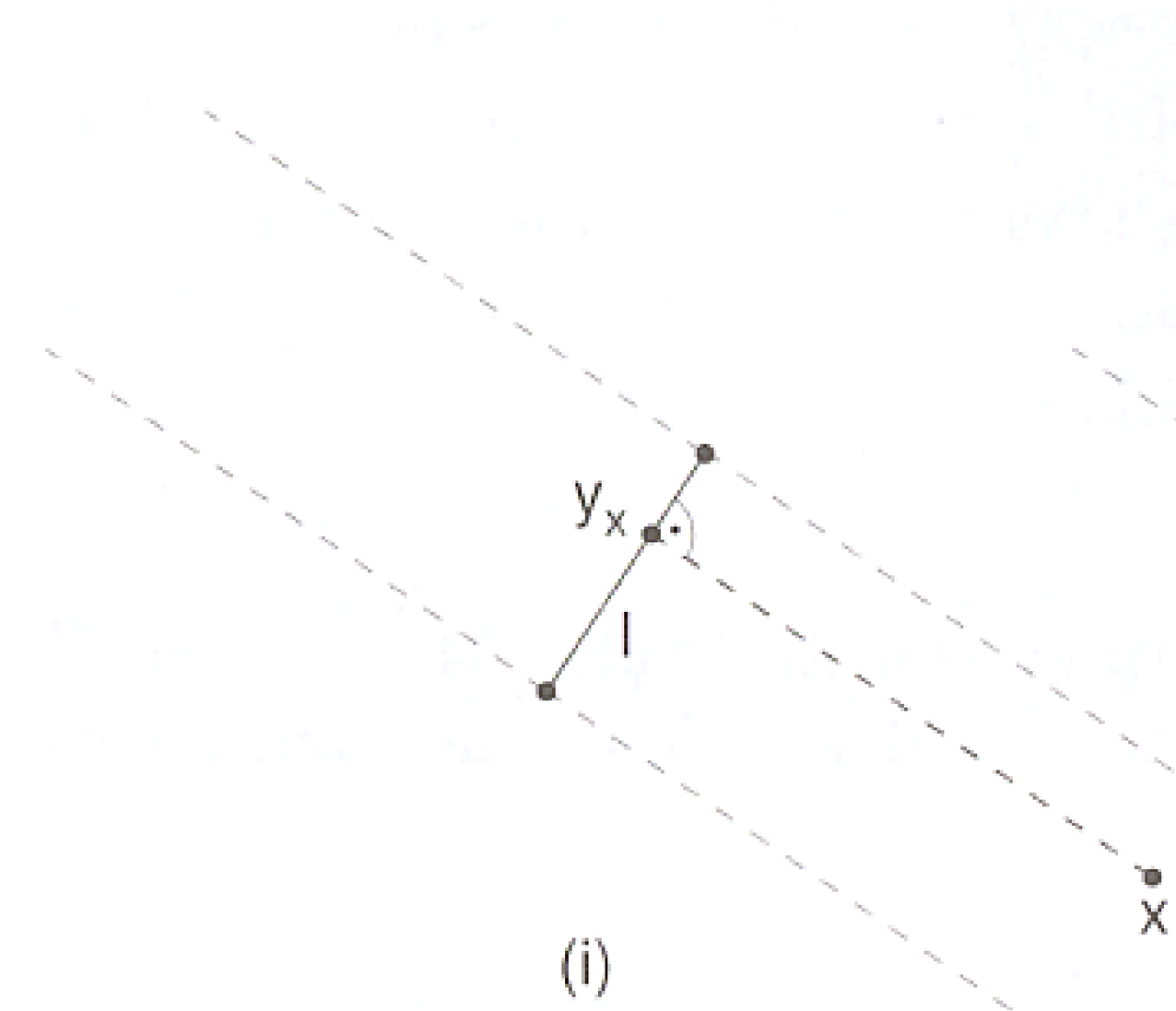
Die Karlsruhe-Metrik ist ebenfalls keine Metrik mit konvexer Distanzfunktion.

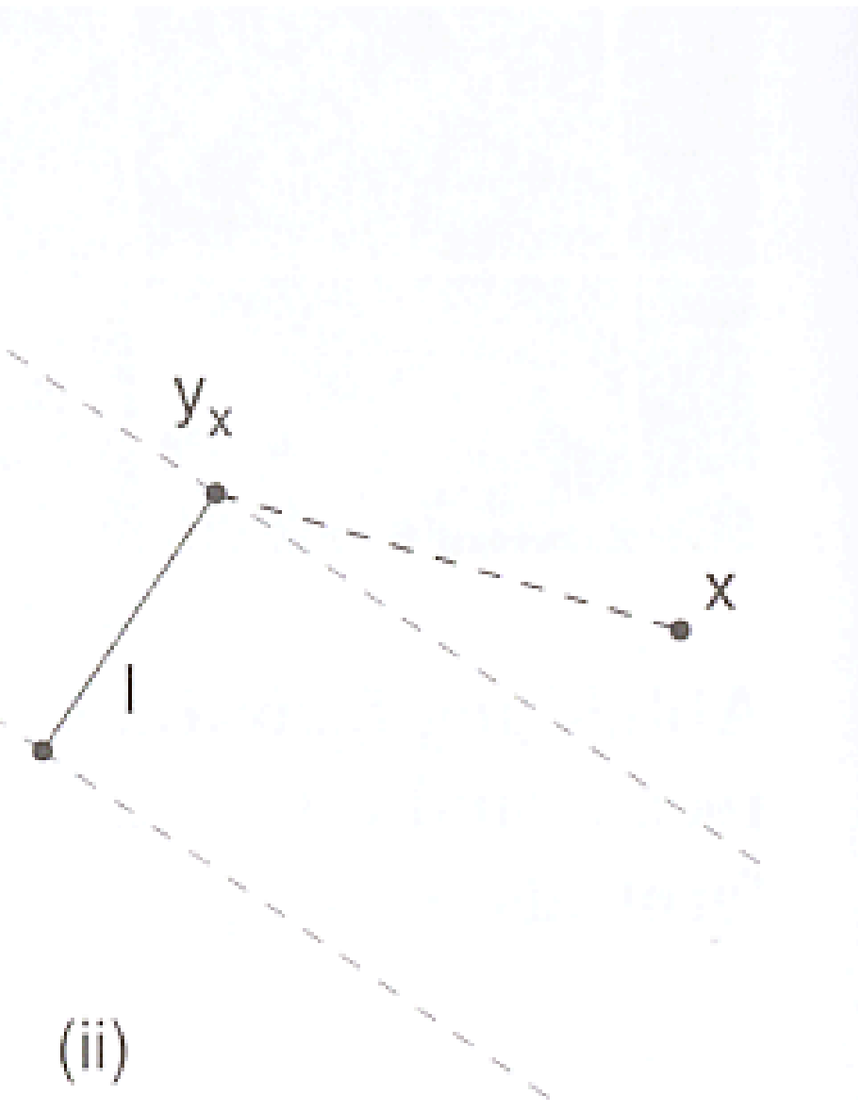
Ein Voronoi-Diagramm zu 8 Punkten in der Karlsruhe-Metrik:



# Voronoidiagramme zu Liniensegmenten (in der Ebene)

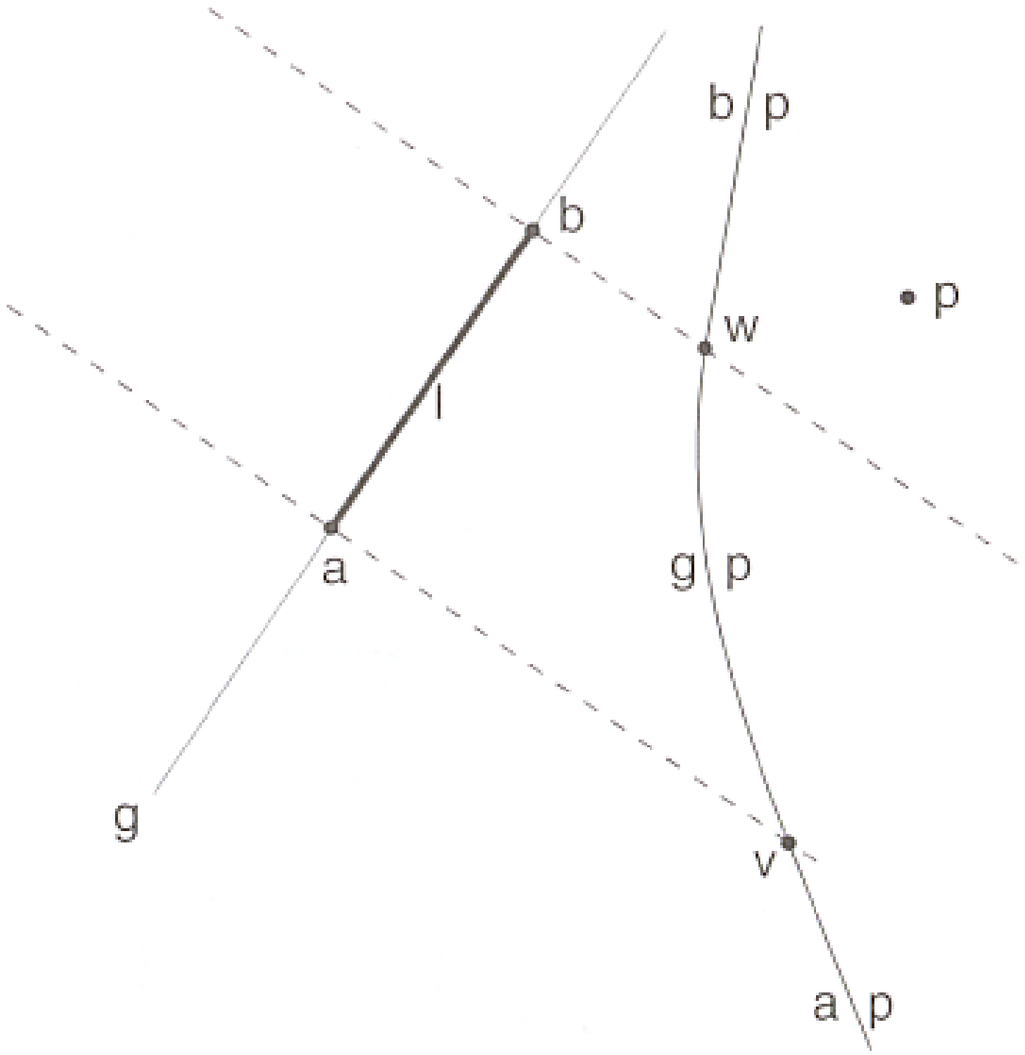
Der Abstand eines Punktes $p$ zum Liniensegment $\ell$ ist die kürzeste Verbindung von $p$ zu einem Punkt von $\ell$.



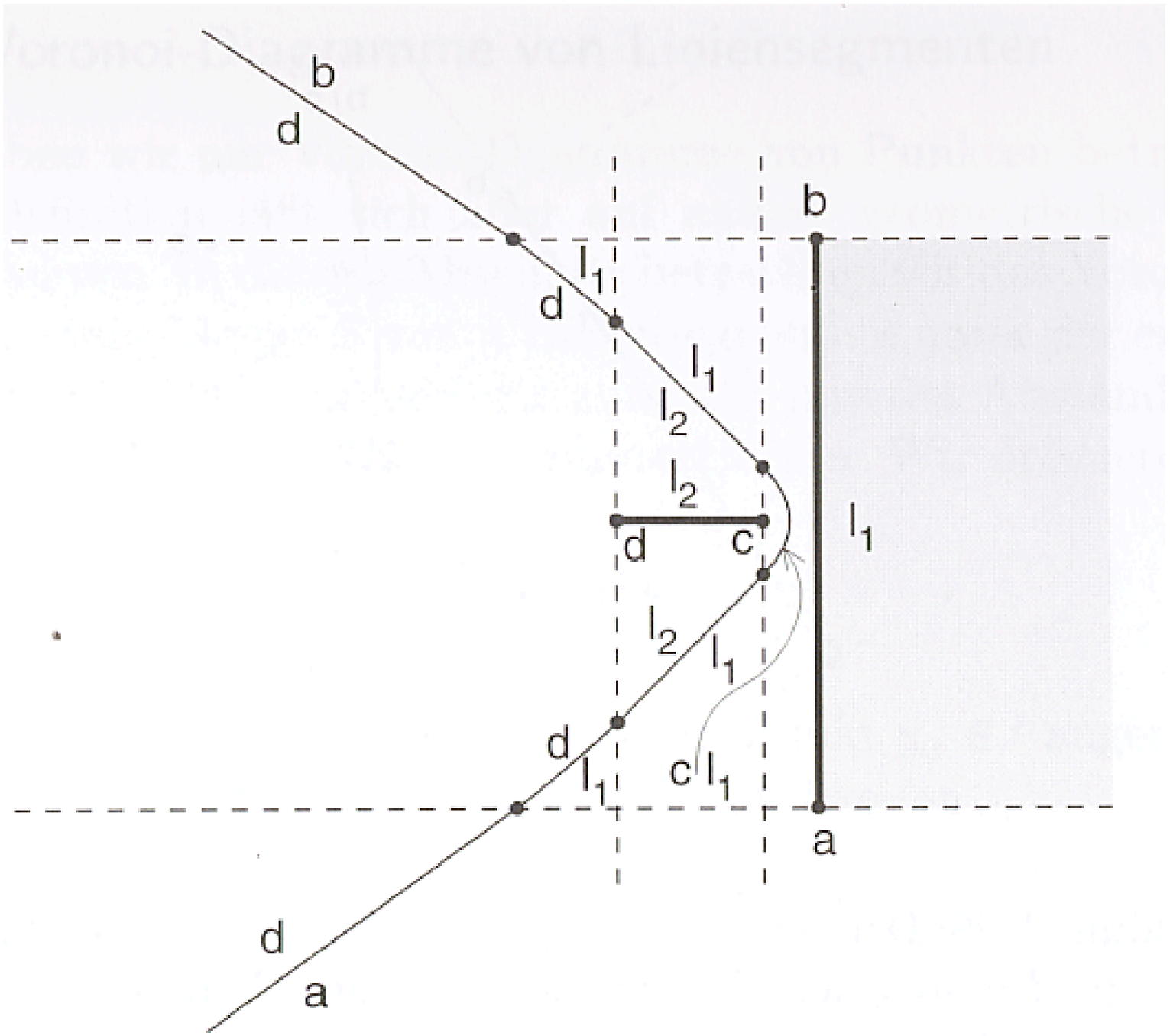
# Bisektoren Punkt-Liniensegment

Der Bisektor $B(\ell,p)$:



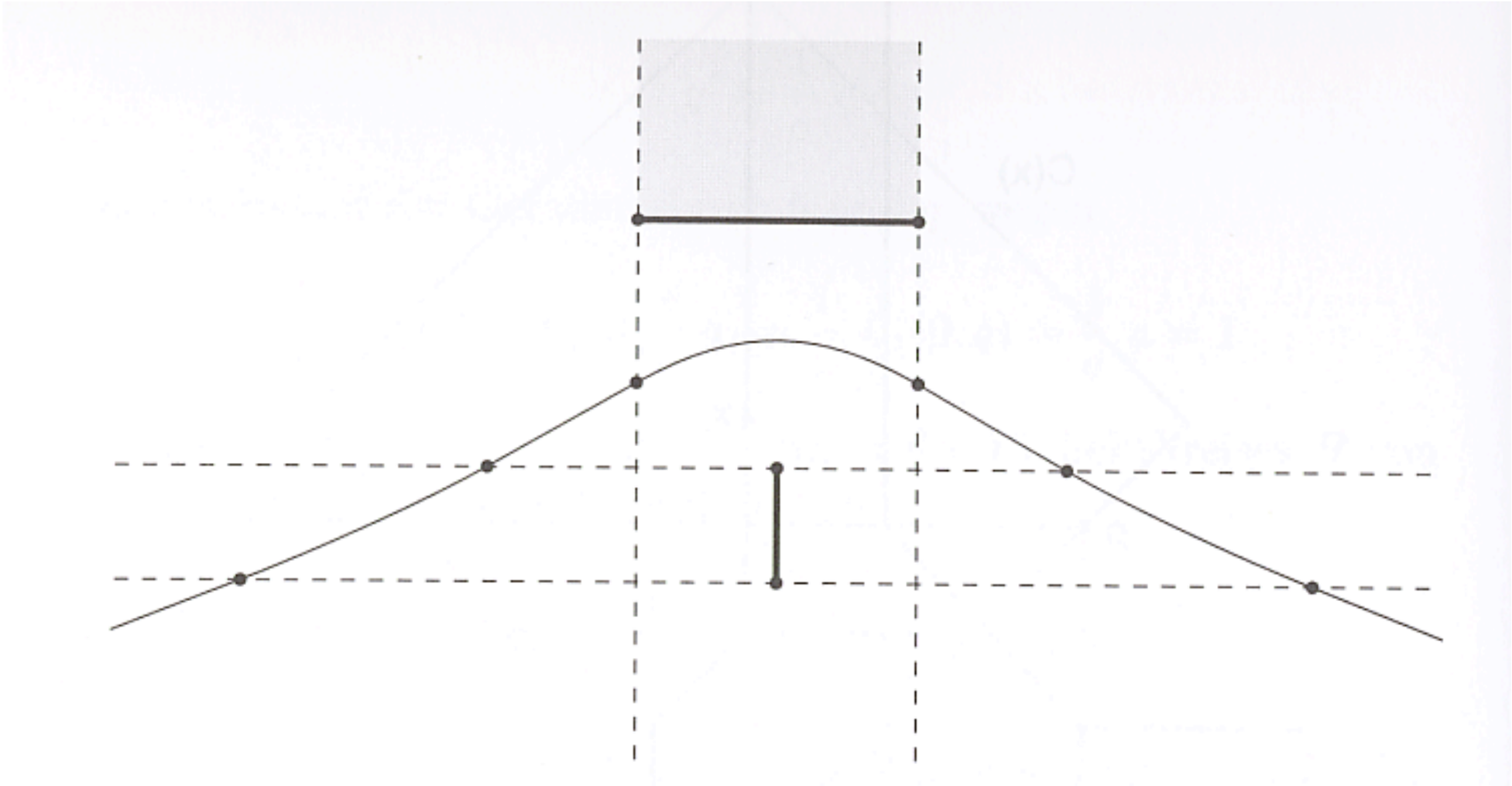

Jeder Bisektor $B(\ell,p)$ zwischen einem Punkt $p$ und einem Liniensegment $\ell$ besteht aus 2 Halbgeraden und einem Parabelstück.

# Bisektor Liniensegment-Liniensegment




Jeder Bisektor $B(\ell_1,\ell_2)$ zwischen zwei Liniensegmenten $\ell_1$ und $\ell_2$ besteht aus 2 Halbgeraden, Liniensegmenten und Parabelsegmenten, deren Gesamtzahl 7 nicht übersteigt.

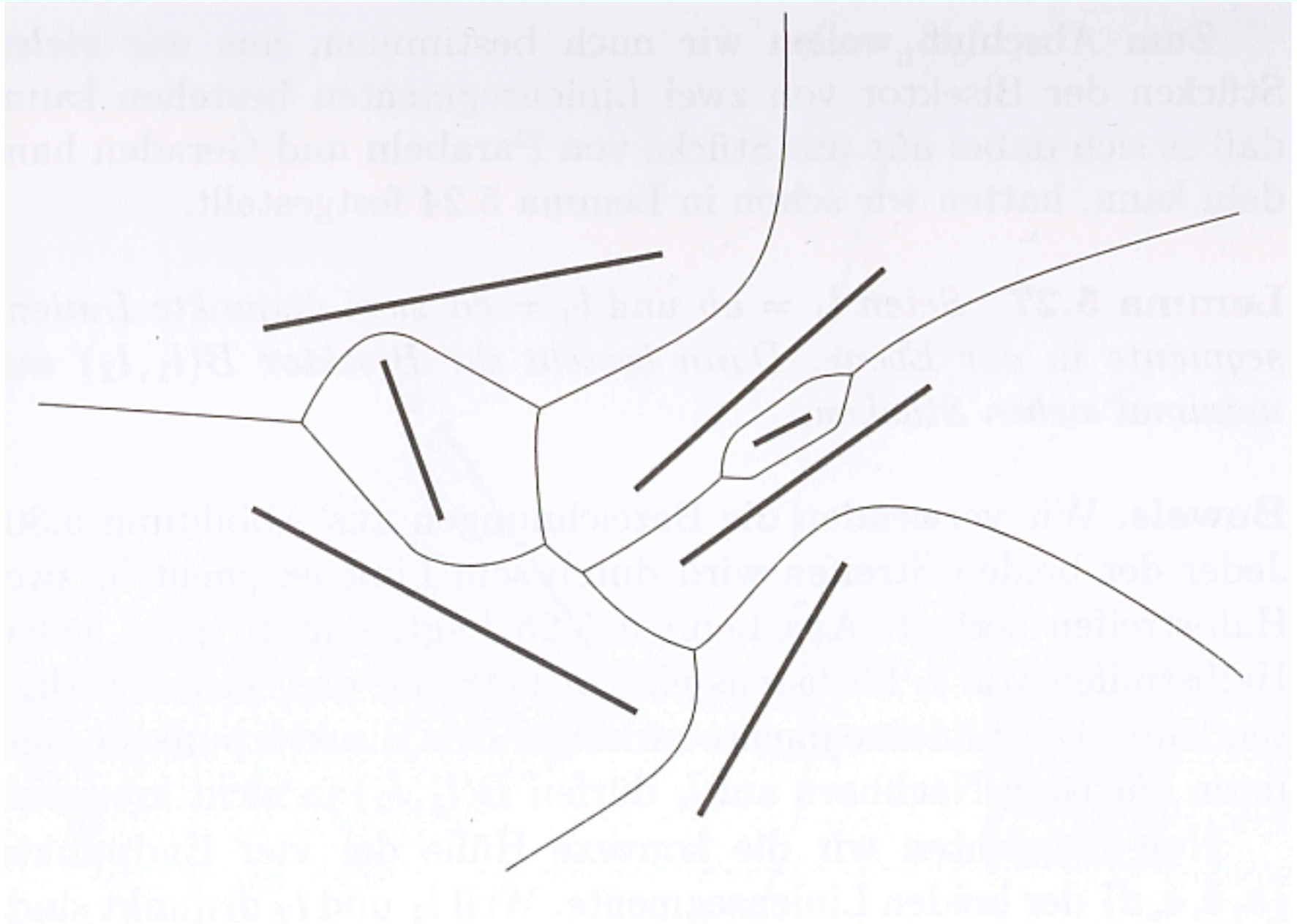
# maximal 7 Bisektorstücke



**Satz:**

Das Voronoi-Diagramm von $n$ disjunkten Liniensegmenten in der Ebene hat $O(n)$ viele Voronoi-Knoten und $O(n)$ viele Voronoi-Kanten. Jede Kante besteht aus $O(1)$ vielen Stücken und der Rand einer Voronoi-Region enthält im Mittel höchstens $6$ Kanten.

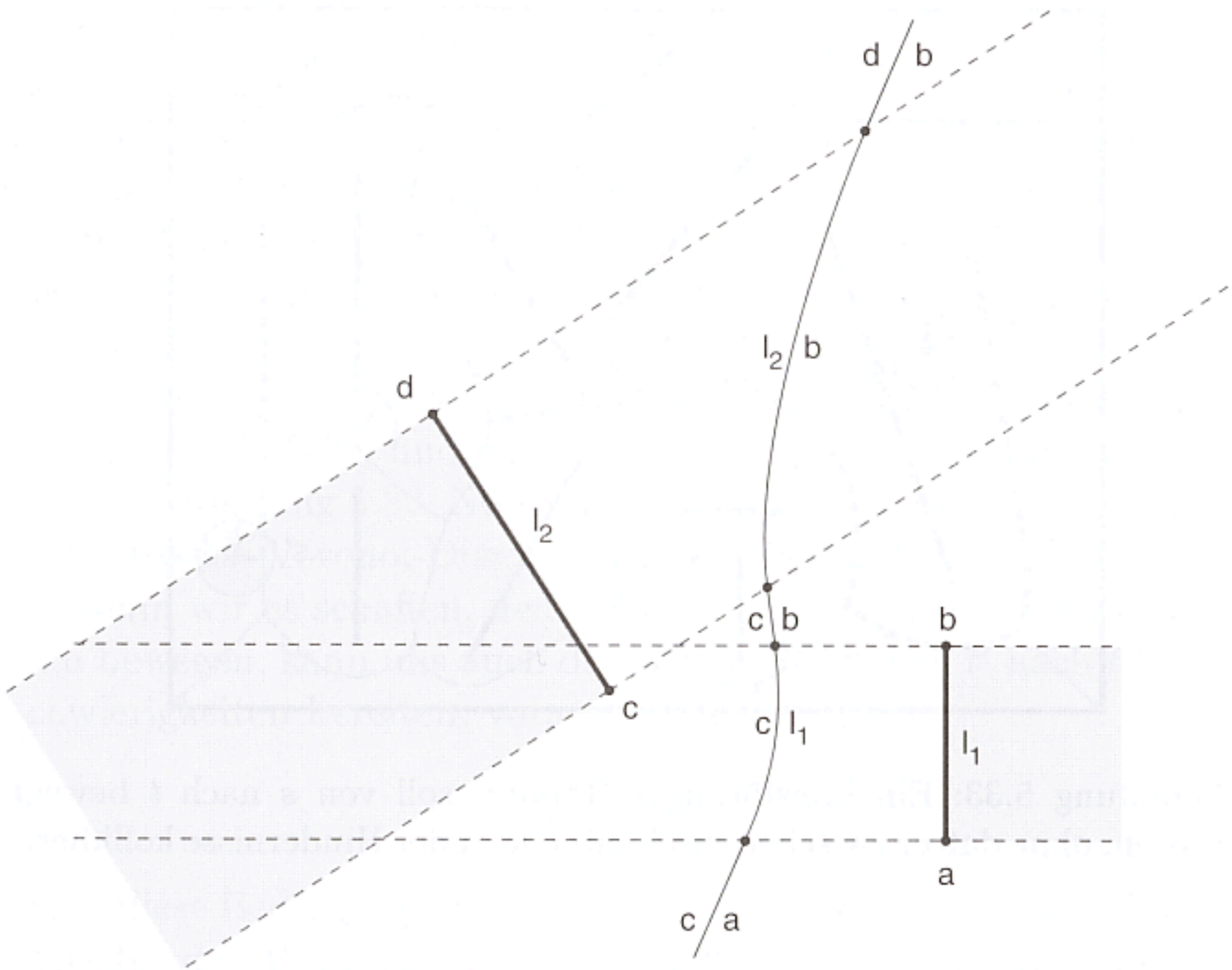
# Voronoi-Diagramm zu Liniensegmenten



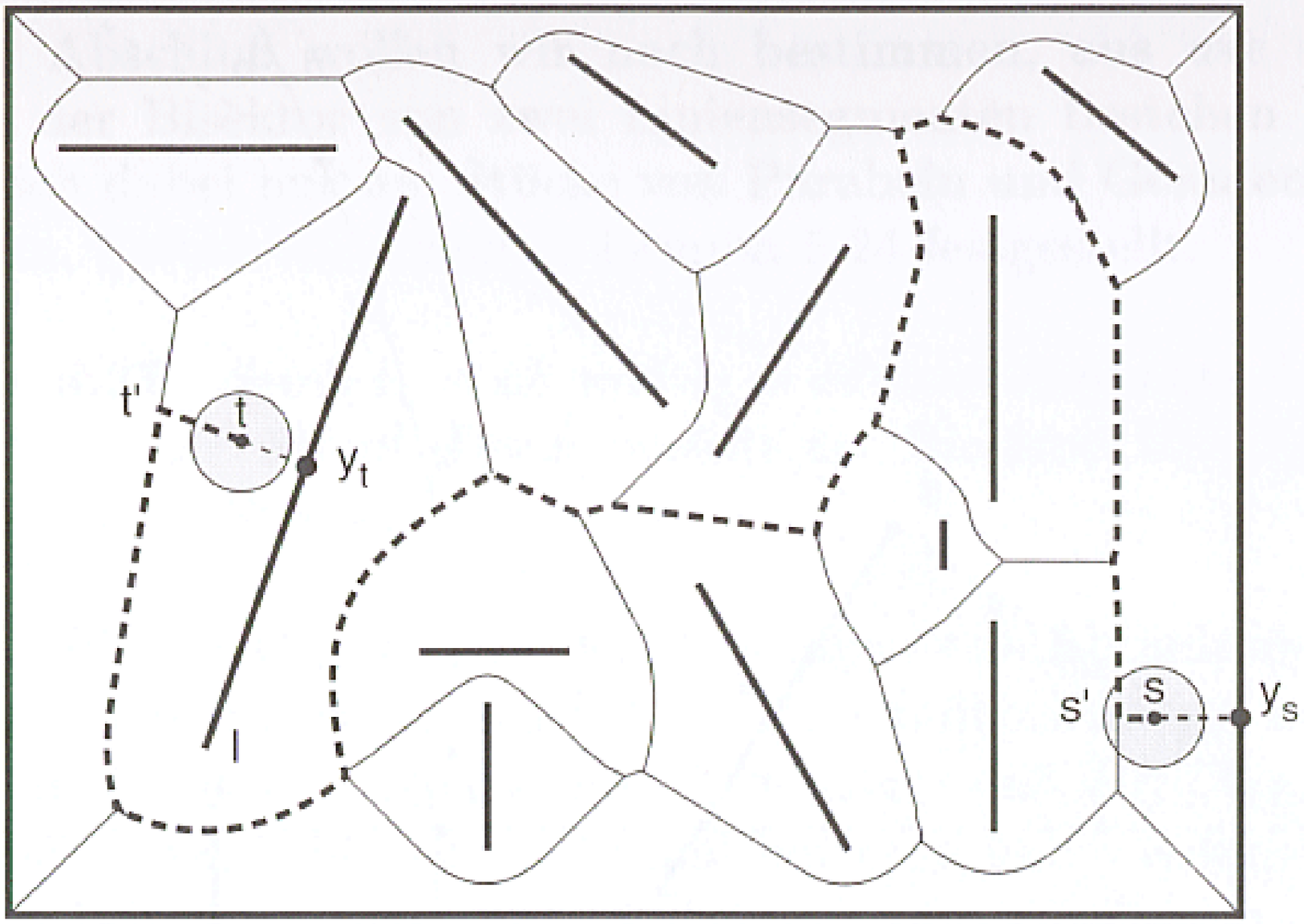
Die Voronoi-Regionen von Liniensegmenten sind zusammenhängende Mengen.

Zwei Voronoi-Regionen können  mehr als eine gemeinsame Randkante Besitzen!

# 5 Bisektorenstücke



# Roboterbewegung auf den Bisektorstücken




**Satz:**

Genau dann kann sich der Roboter kollisionsfrei von $s$ nach $t$ bewegen, wenn sein Radius die Abstände $|sy_s|$ und $|ty_t|$ nicht übersteigt, und wenn es eine kollisionsfreie Bewegung von $s'$ nach $t'$ entlang der Voronoi-Kanten gibt.

# Kollisionsfreie Bewegung eines runden Roboters

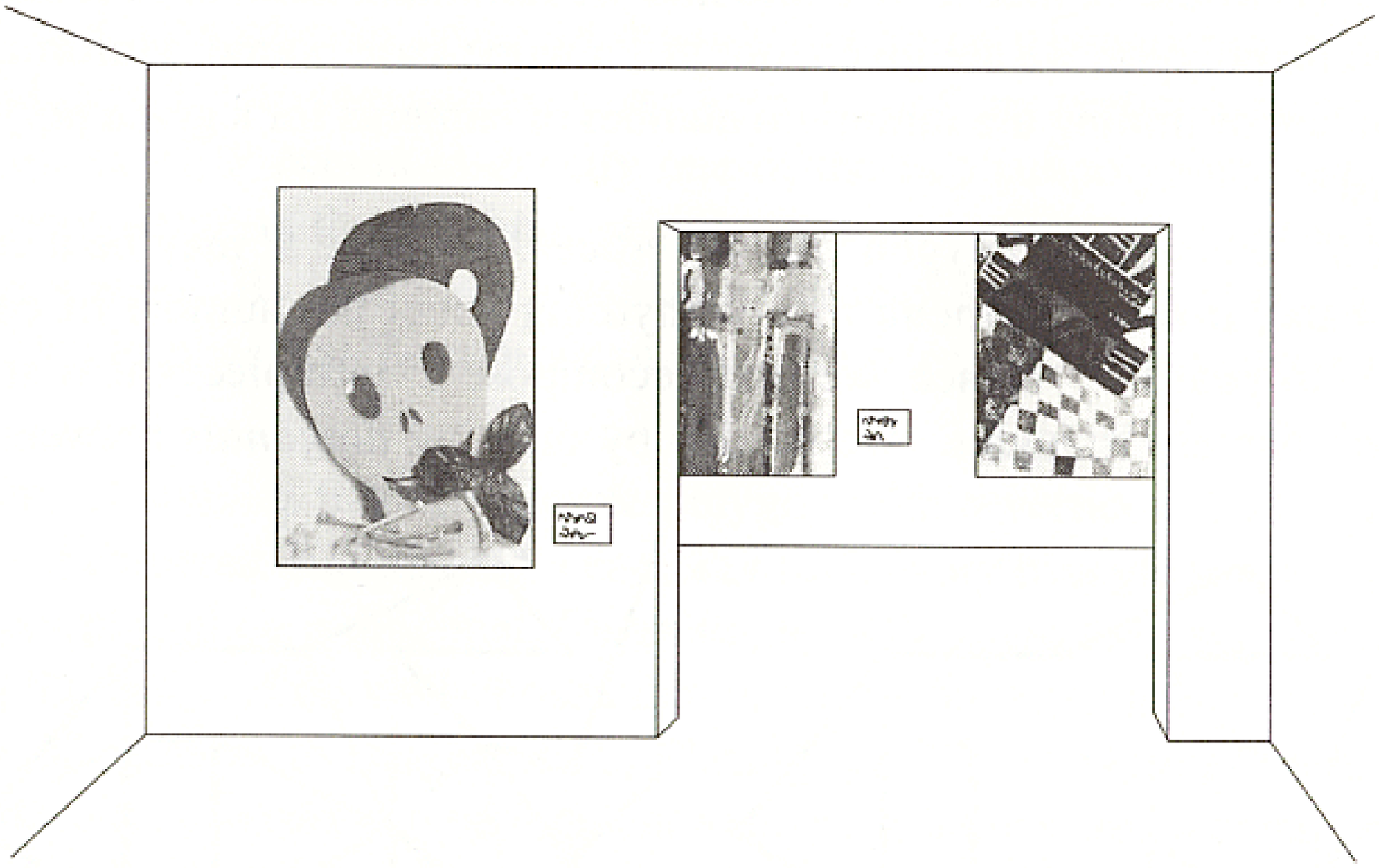
**Satz:**

Ist das Voronoi-Diagramm der $n$ Liniensegmente vorhanden, lässt sich für einen beliebigen Roboterradius $r$ und beliebige Punkte $s$ und $t$ in Zeit $O(n)$ ein kollisionsfreier Weg von $s$ nach $t$ bestimmen; oder aber feststellen dass es keinen gibt.

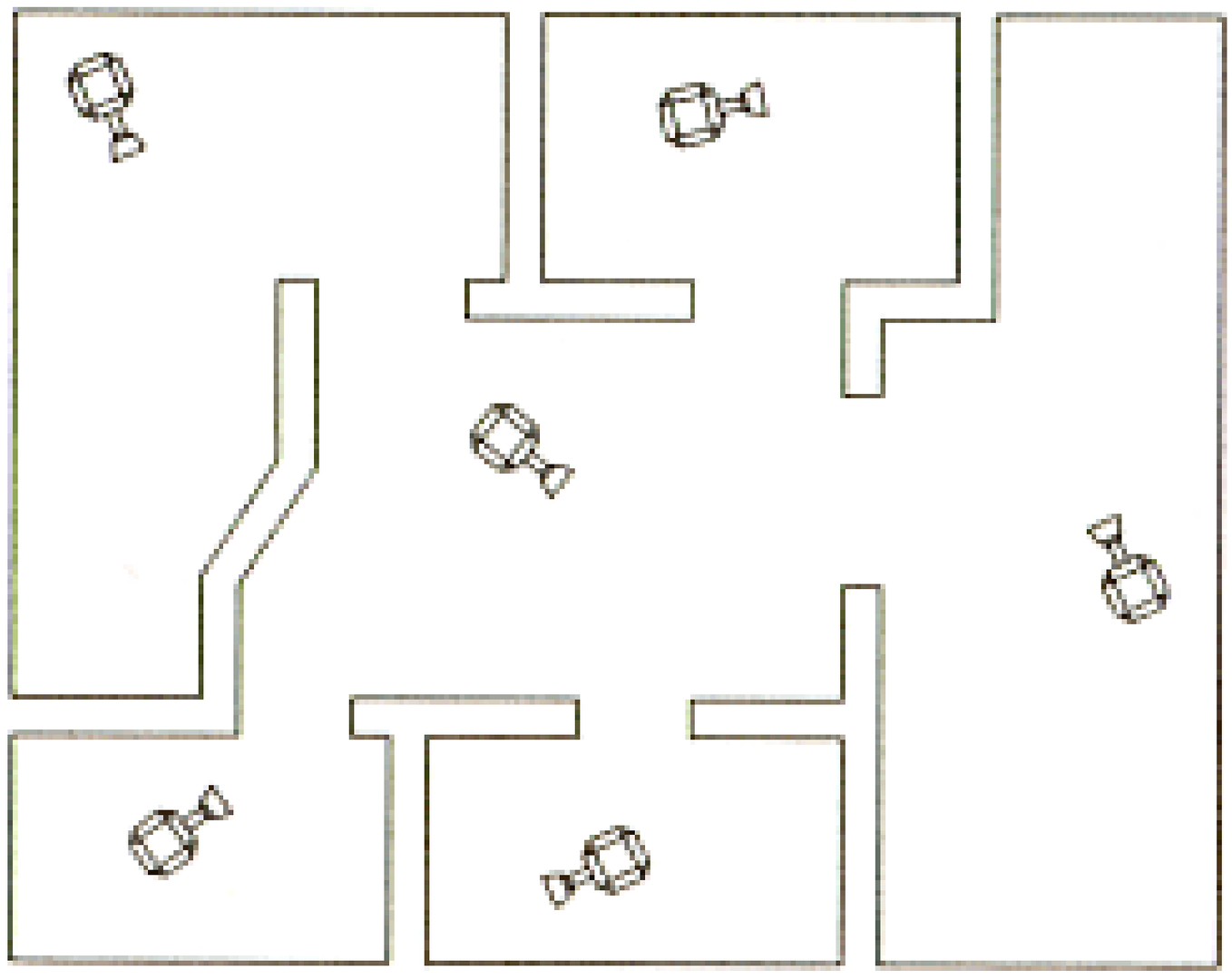
**Beweis:**

1. alle Kanten entfernen, deren Abstände für Roboterradius zu eng beieinander stehen.
2. Start- und Endpunkte $s'$ und $t'$ auf Voronoi-Kanten bestimmen.
3. In Breitensuche einen Weg auf verbliebenen Voronoi-Kanten von $s'$ nach $t'$ suchen.

# Kunstgalerie-Überwachung

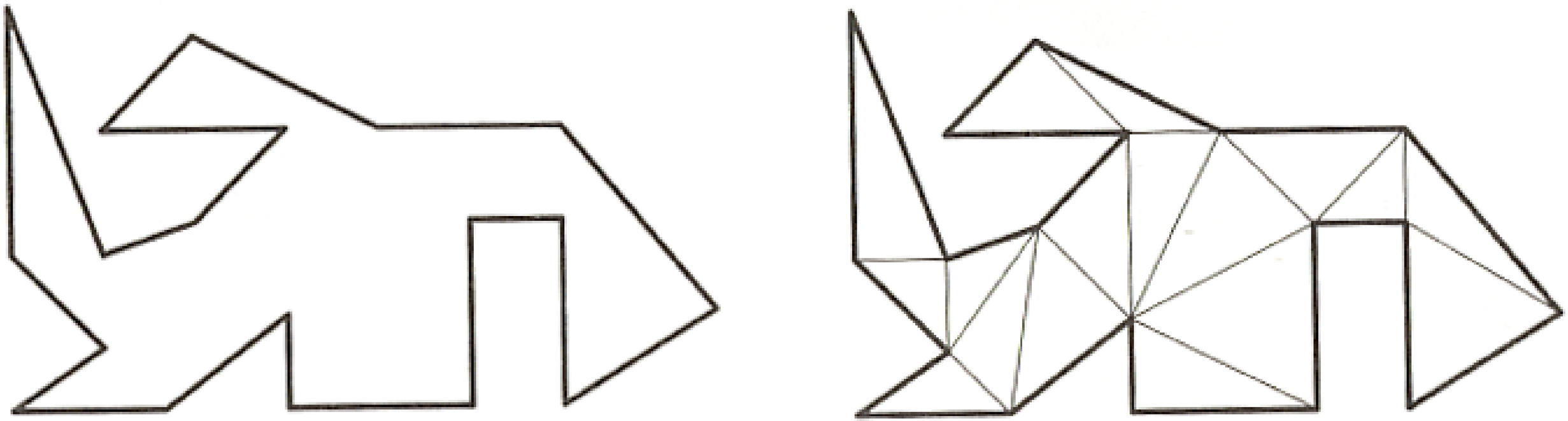


# Kunstgalerie-Überwachung



# Triangulationen von Polygonen

**Feststellung:**
Wenn ein einfaches Polygon $n$ Ecken hat, dann hat seine Triangulation $n-2$ Dreiecke.



***Induktions-Beweis:***

**Basis:**

$n=3$ hat 1 Dreieck.

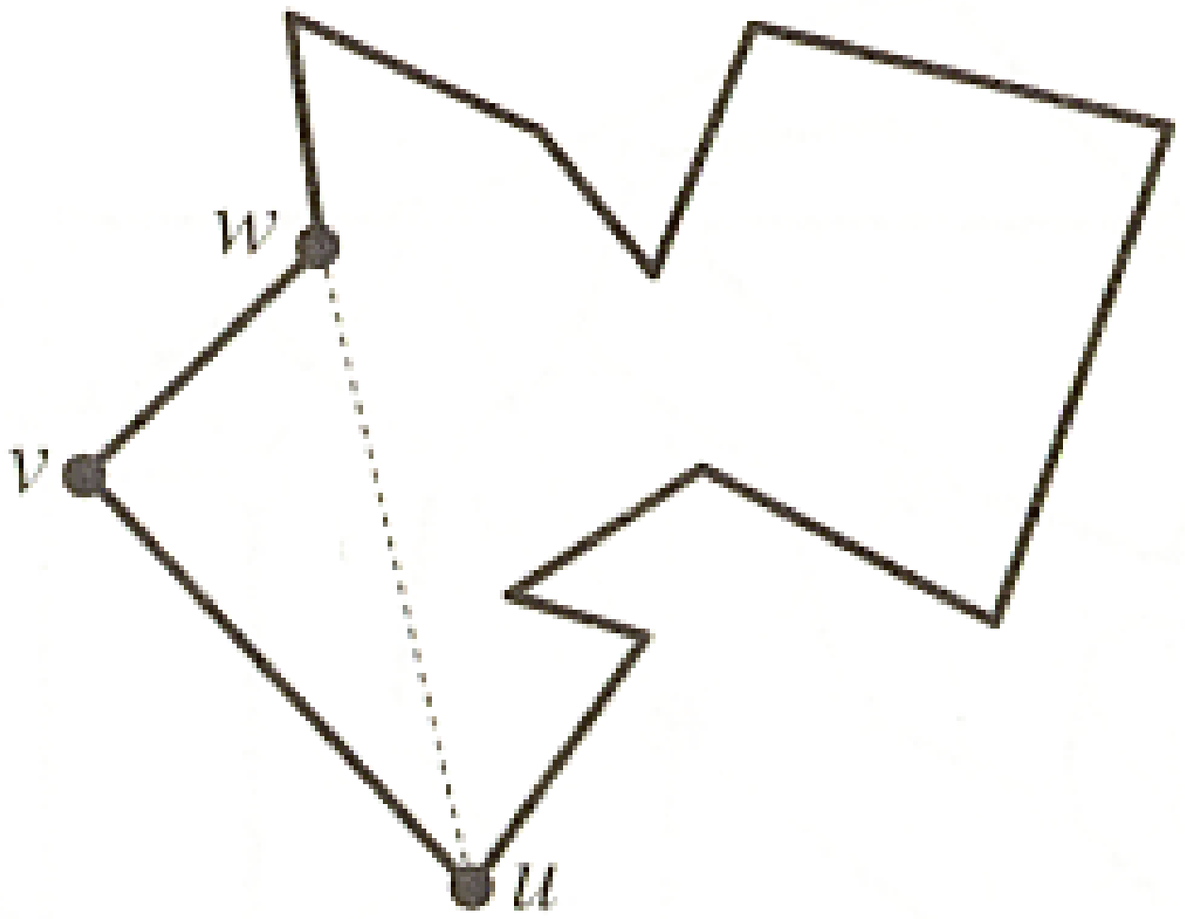
**Induktions-Annahme:**

Behauptung stimmt für ein festes $m$ und alle $n \leq m$.

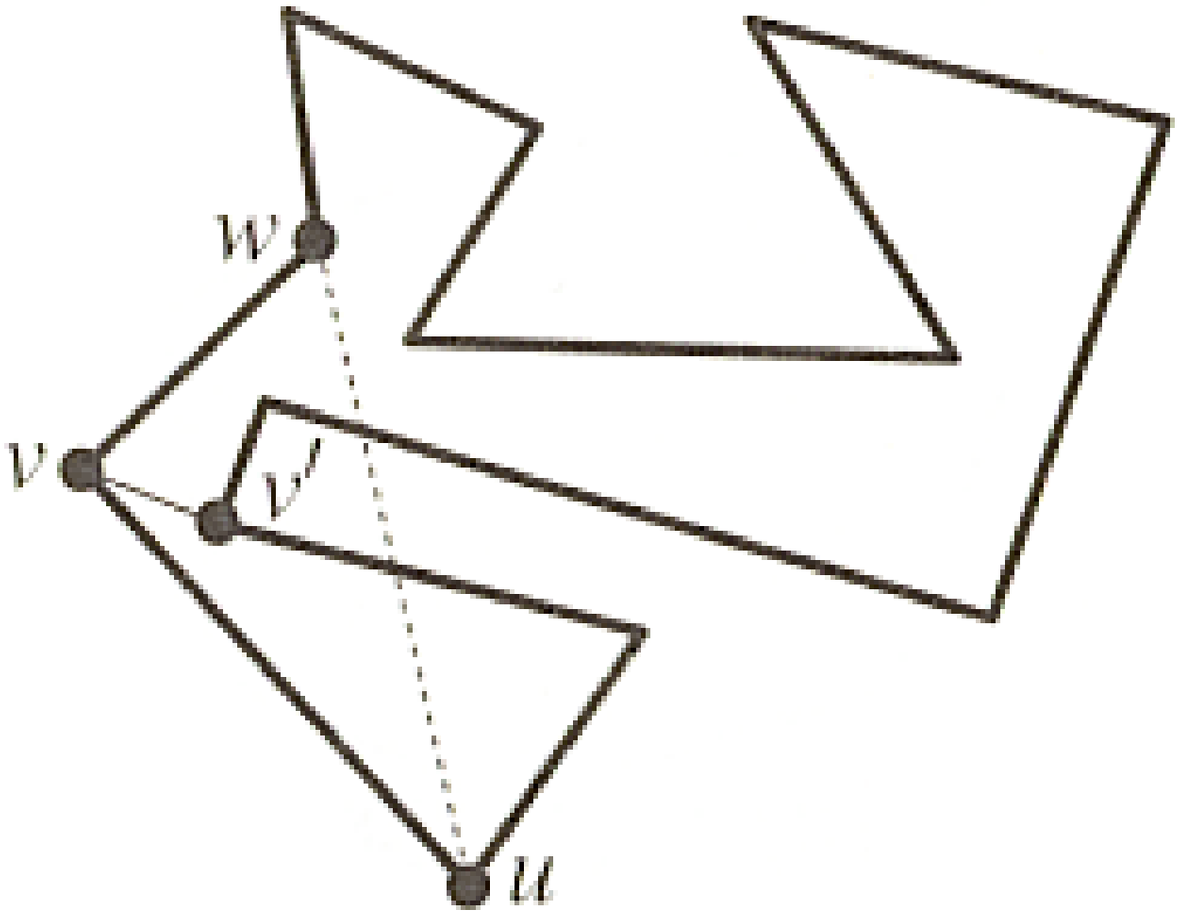
# Induktions-Schritt

**Induktions-Schritt:**

Das Polygon habe $m+1$ Ecken. Wir wählen eine nicht einspringende Ecke $v$ aus und betrachten die Gerade, die ihre Nachbarn $u$ und $w$ verbindet. Ist diese vollständig im inneren des Polygones, so gibt es ohne $v$ ein trianguliertes Polygon mit $m$ Ecken und nun ein Dreieck und eine Ecke mehr, d.h. die Formel stimmt.



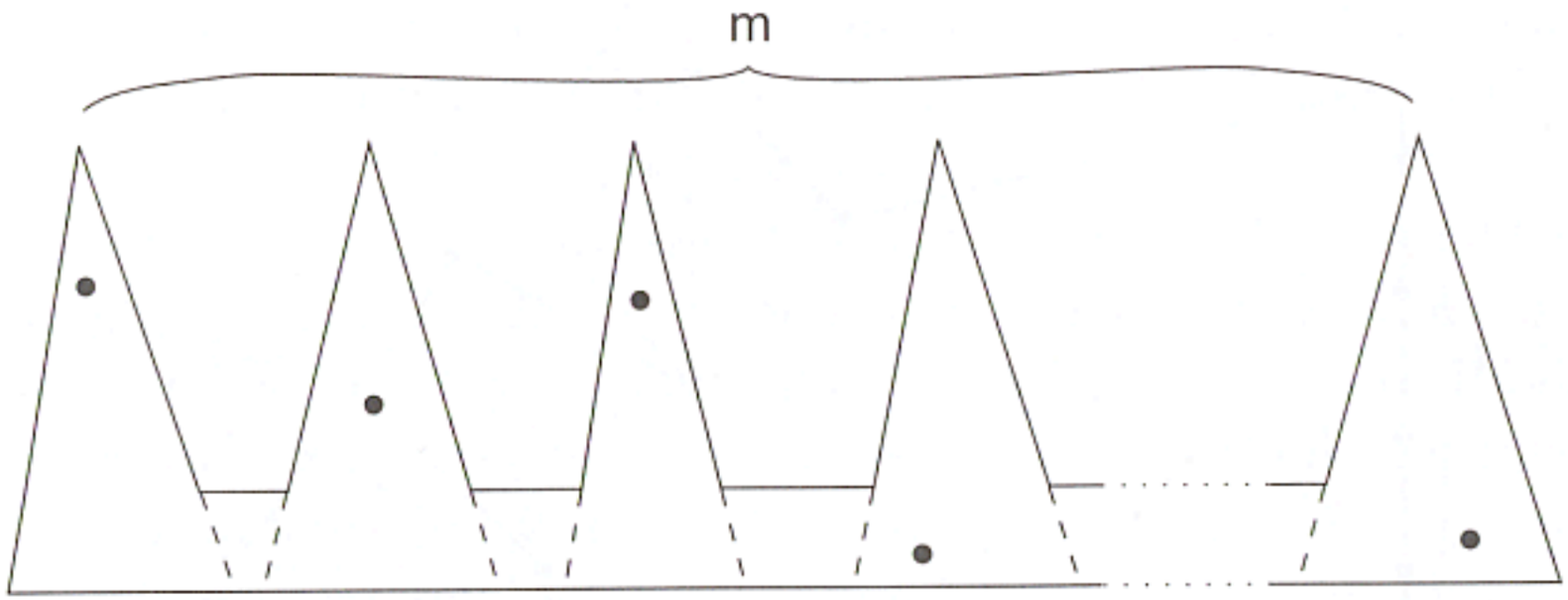
**Induktions-Schritt Fortsetzung:**
Ist die Kante $vw$ vollständig im inneren des Polygones, so gibt es zwischen oder auf der Geraden $uw$ und der Ecke $v$ eine weitere Ecke $v'$ des Polygons. Die Gerade $vv'$ zerlegt das Polygon in zwei kleinere Polygone mit $m_1$ (bzw. $m_{2)}$ Ecken und $m_1 - 2$ (bzw $m_2 - 2$) Dreiecken. Zusammen sind dies $m_1 + m_2 - 4$ Dreiecke und $m_1 + m_2 - 2$ Ecken, denn $v$ und $v'$ werden beim Zusammenfügen nicht mehr doppelt gezählt! Die zu beweisende Formel stimmt auch hier.



# Obere und untere Schranken für Anzahl nötiger Wächter

Natürlich reicht es, in jedes Dreieck der Triangulation einen Wächter zu stellen, d.h. mit $n-2$ Wächtern kommt man immer aus!

Und $n/3$ ist eine unter Schranke, wie das Polygon mit $3m$ Ecken und $m$ erforderlichen Wächtern zeigt.

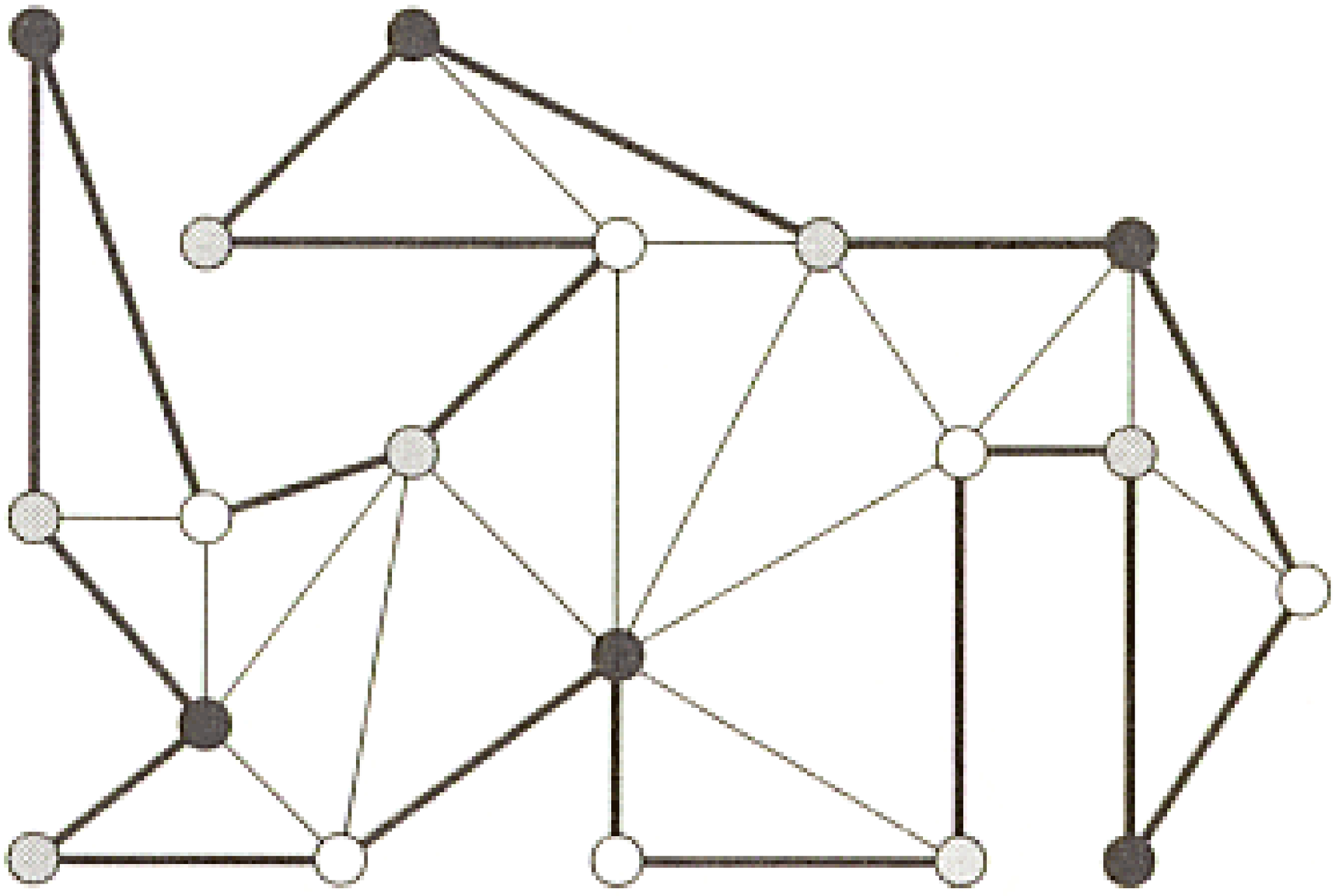



Die Bestimmung der kleinsten notwendigen Zahl von Wächtern ist *NP*-schwer! [O'Rourke: Art Gallery Theorems and Algorithms, Oxford Univ. Press, New York (1987)]

# 3-Färbung

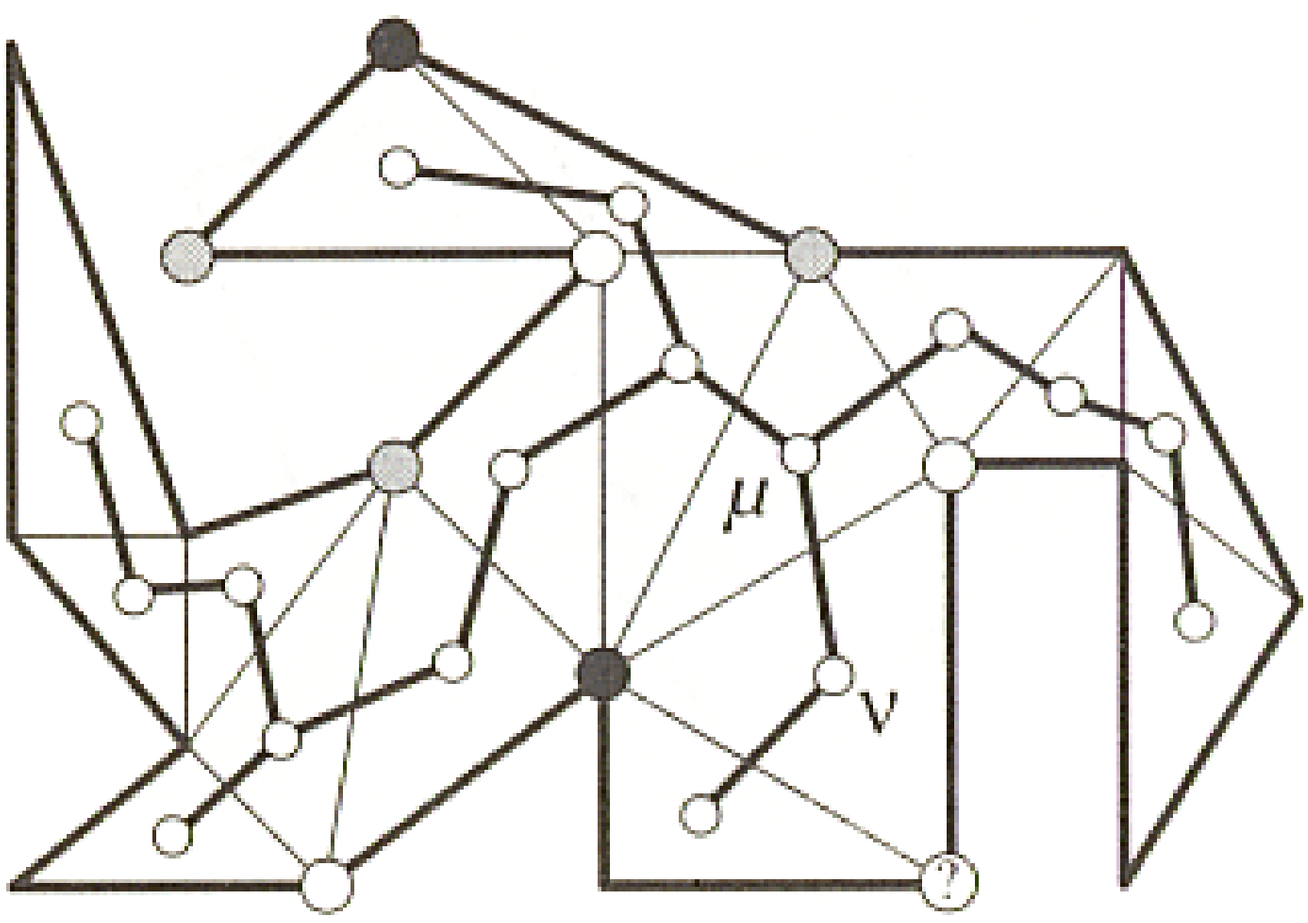
Gibt es bessere Obere Schranke? ***Ja!***

Wenn an jeder Ecke mit gleicher Farbe bei einer 3-Färbung ein Wächter postiert ist, so ist jedes Dreieck überwacht! Also reichen $\lfloor m/3 \rfloor$ viele Wächter, sofern wir eine 3-Färbung sicher stellen können.



# Beweis für Existenz von 3-Färbung einer Triangulation

Der duale Graph einer Triangulation eines Polygons (ohne Löcher) ist ein Baum: Jede seiner Kanten kreuzt eine Dreieckskante, diese aber teilen das Polygon in zwei getrennte Teile, und  das Entfernen des Astes teilt den dualen Graphen in 2 Komponenten.
Beginnt man mit dem 3-gefärbten Dreieck das zu einem Blatt des dualen Baumes gehört, so ist die dritte Farbe im angrenzenden Dreieck, welches man entlang des Baumes erreicht, eindeutig bestimmt. Wegen der Baumstruktur kann es nirgendwo einen Konflikt geben.

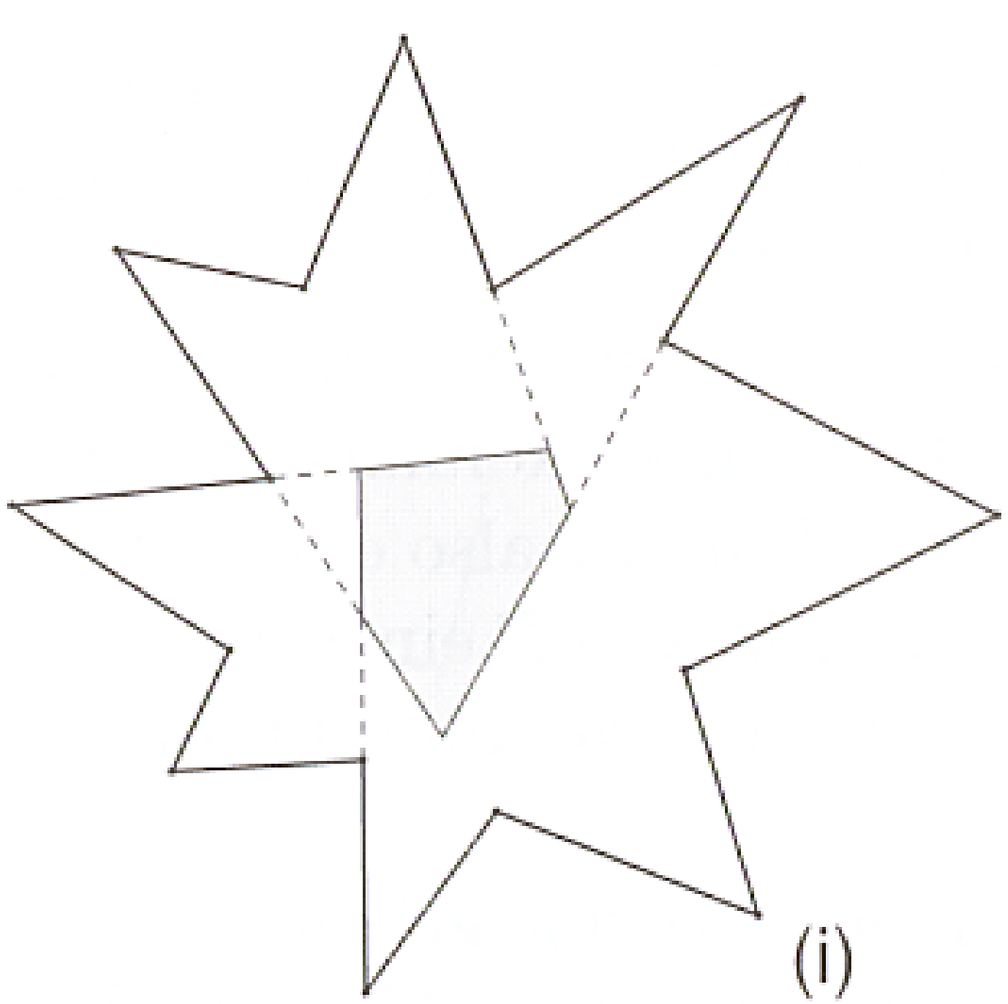


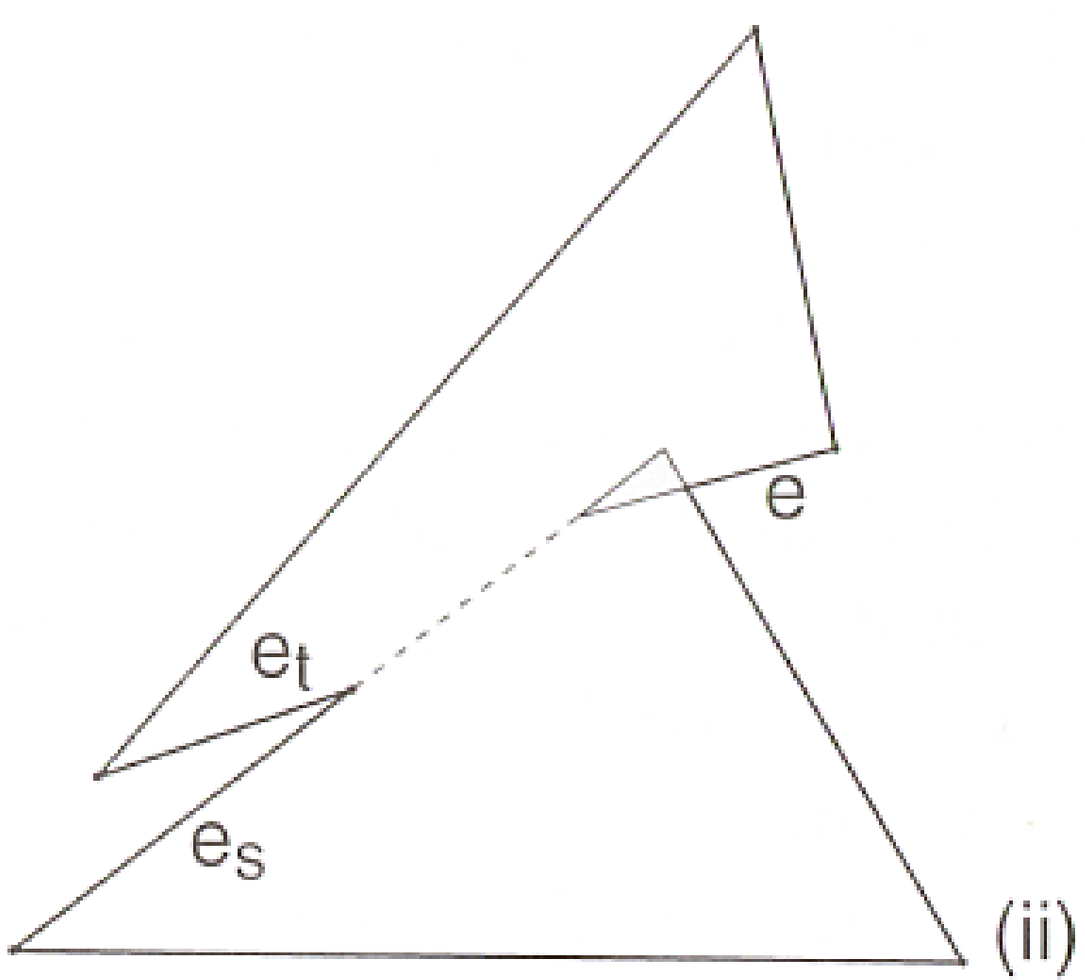
# Der Kern eines Polygons

Der Kern eines Polygons, mit $ker(P)$ bezeichnet, ist der Bereich, von dem aus das gesamte Polygon sichtbar ist!

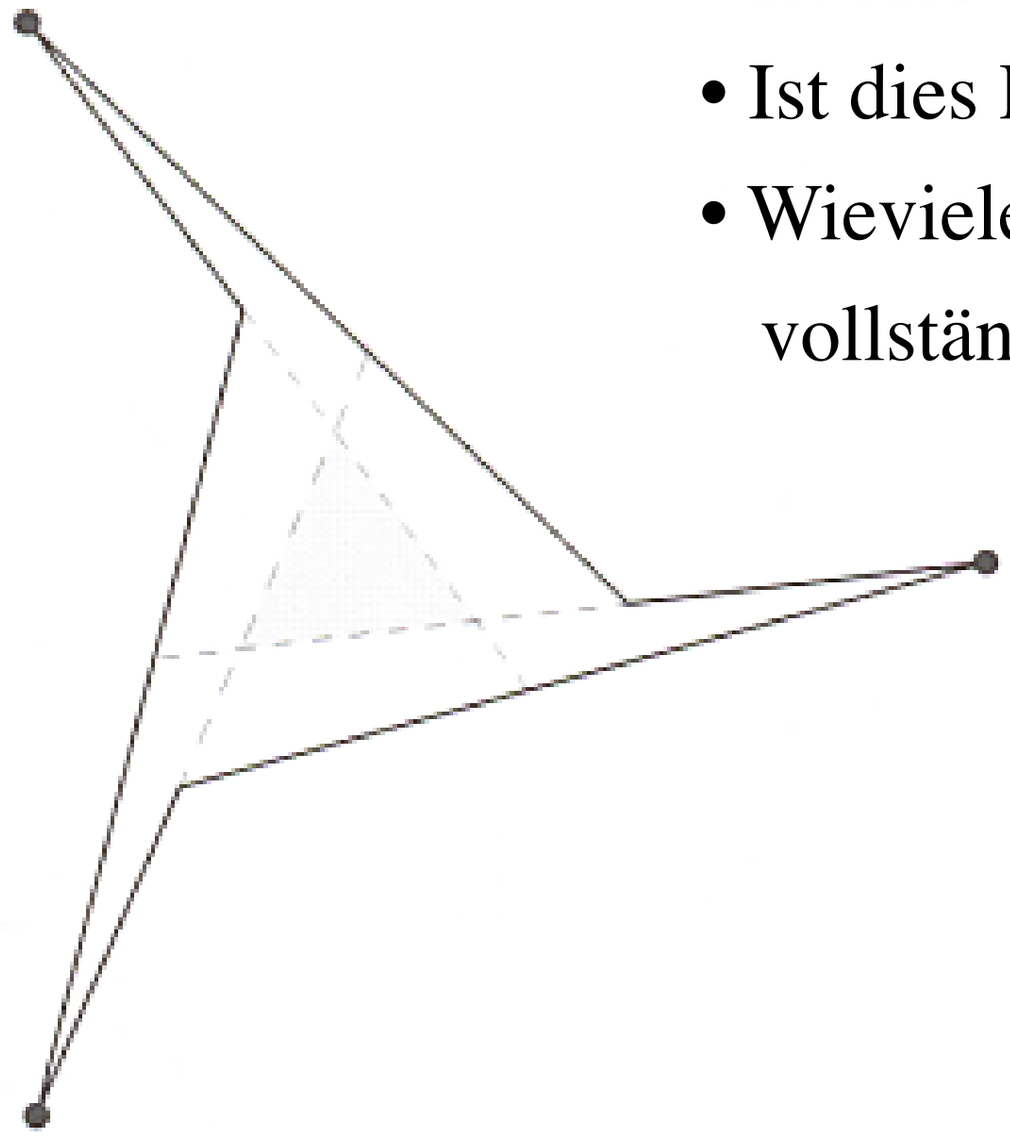
Ein Polygon heißt sternförmig, wenn sein Kern nicht leer ist.

Jedes konvexe Polygon ist natürlich sternförmig!



# Stern oder nicht?

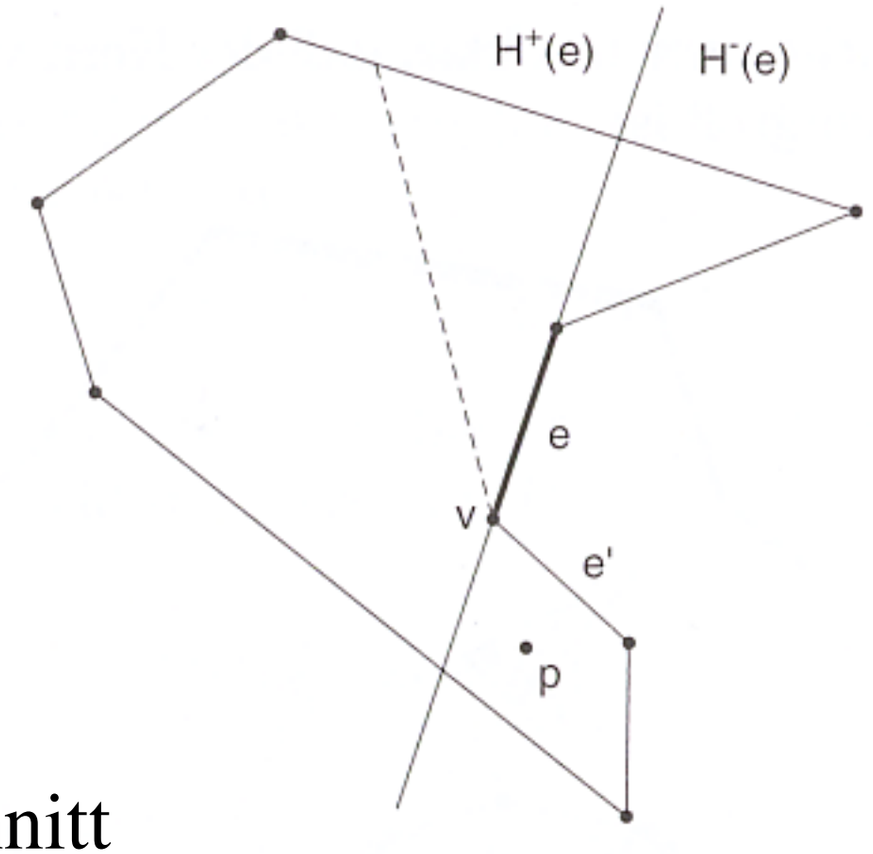


Beispiel eines Polygons, bei dem 3 Wächter den gesamten Rand bewachen, aber das Innere nicht vollständig sehen.

- Ist dies ein Stern?
- Ist dies Polygon konvex?
- Wieviele Wächter benötigt man zur vollständigen Überwachung?

# Kern als Durchschnitt von Halb-Ebenen

Der Rand eines Polygones wird entgegen der Uhrzeigerrichtung durchlaufen.
Mit $H^+(e)$ (bzw. $H^-(e)$) einer Kante $e$ wird die linke (rechte) Halb-Ebene bezeichnet. Offensichtlich sieht man die Kante $e$ nie von aus $H^-(e)$ aus, bestenfalls von $H^+(e)$ aus.




**Lemma:**

$$ker(P) = \bigcap_{e \text{ ist Kante von } P} H^+(e)$$

Wir brauchen also nur noch den Durchschnitt aller Halb-Eben zu bestimmen, um den Kern eines Polygons zu finden.

# Durchschnitt von Halb-Ebenen berechnen

Man zerlegt die Menge aller Halb-Ebenen in zwei etwa gleich große Teilmengen und berechnet deren Durchschnitte rekursiv.
Das Ergebnis werden konvexe Mengen sein, die durch polygonale Ketten mit höchstens $n$ Kanten berandet sind. Deren Durchschnitt berechnet man mit einem sweep-Verfahren in $O(n)$ Zeit.

Daraus ergibt sich Laufzeit von $O(n \ \ell og \ n)$.

Es gibt ein etwas umständlicheres Verfahren, das unter Ausnutzung des Drehwinkels der Polygon-Kanten tatsächlich eine bessere Laufzeit von $O(n)$ besitzt:

**Theorem:**

Der Kern eines einfachen Polygons mit $n$ Ecken kann in $O(n)$ Zeit und mit linearem Speicherplatz berechnet werden.

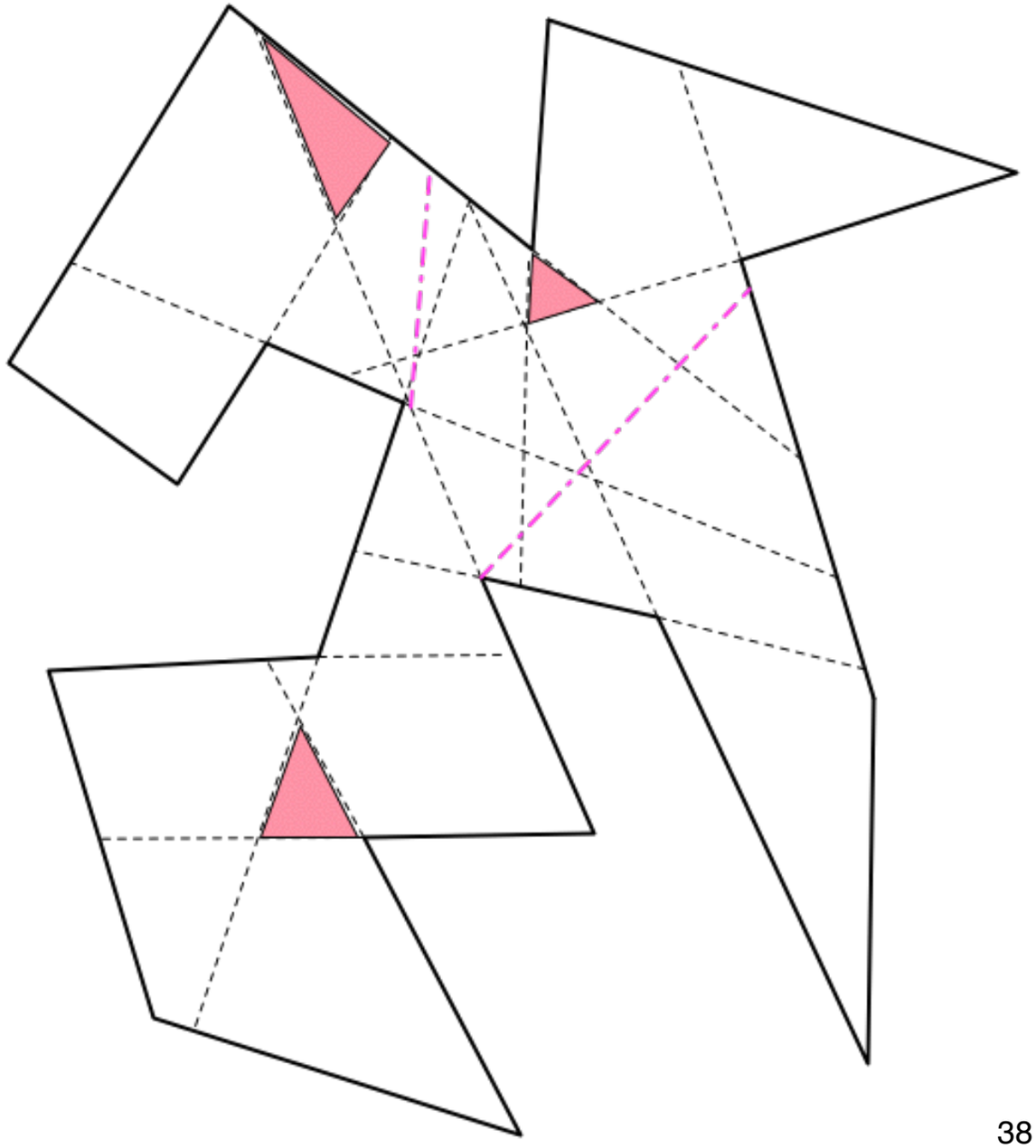
# Lösen Kerne das komplizierte Art-Gallery Problem?

Wenn das Polygon sternförmig ist, dann reicht ein Wächter im Kern $ker(P)$ aus.

Wenn das zu beobachtende Gebiet anders ist???

[dazu steht in R.Klein, Algorithmische Geometrie, nicht mehr viel!]

Einfache Polygone sind Polygone ohne Löcher oder Überschneidungen des Randes. Aber wie sieht dort der Kern aus? Er ist i.A. nicht zusammenhängend!

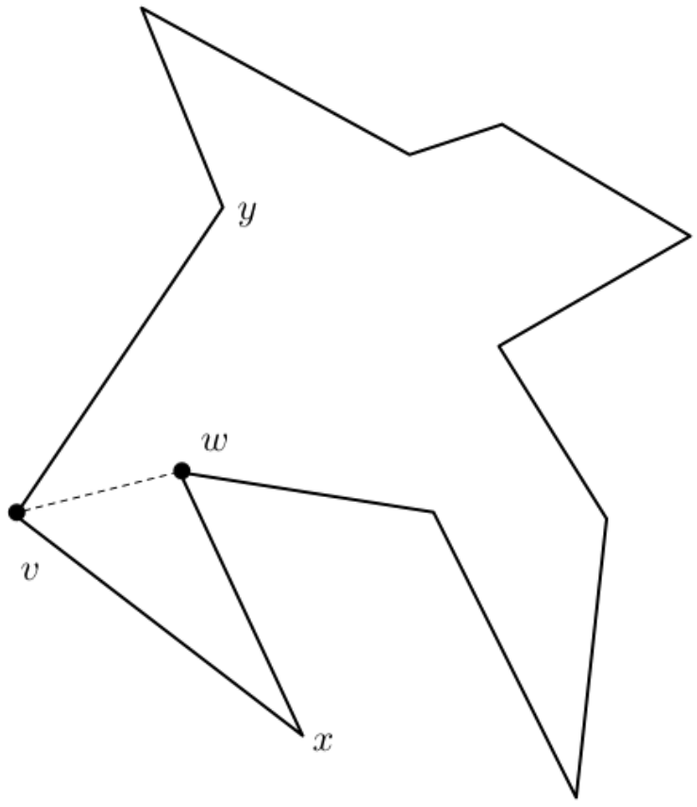


# Triangulation beliebiger Polygone: zunächst in $O(n^2)$ Zeit

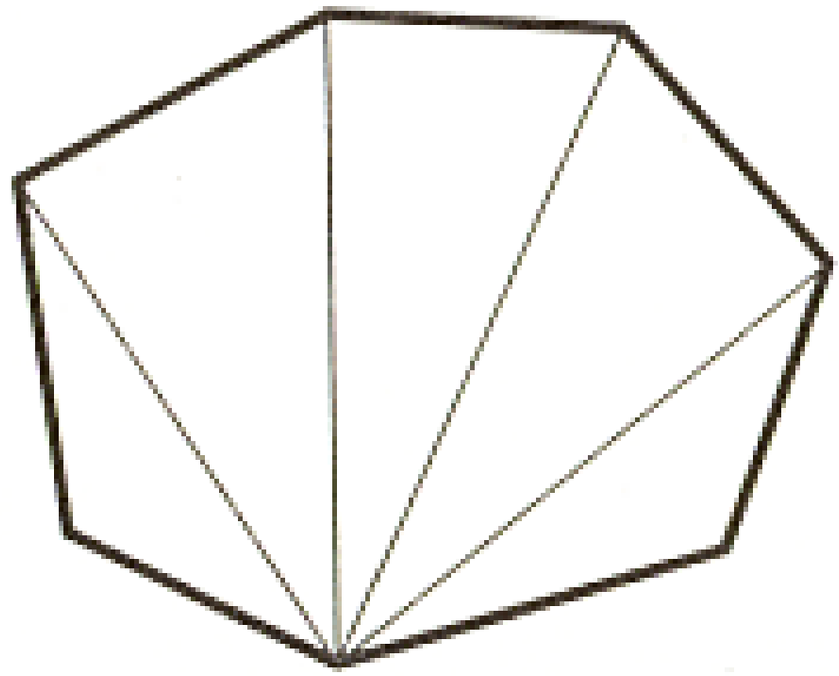
Um ein einfaches Polygone zu triangulieren, könnte man so vorgehen:

Finde eine Diagonale und trianguliere beide Teilpolygone links und rechts davon.
Dazu wähle den von links gesehen ersten Knoten $v$ und versuche dessen Nachbarn $x$ und $y$ zu verbinden. Falls das nicht geht, wähle Knoten $w$ innerhalb des Dreiecks $v-x-y$ der zu $xy$ größten Abstand hat und verbinde ihn mit $v$.

Auf diese Weise findet man stets eine Diagonale in $O(n)$ Zeit und das gesamte Verfahren benötigt $O(n^2)$ Zeit im schlechtesten Fall. Da wären wir mit der Delaunay-Triangulation in $O(n\ \ell og\ n)$ schon besser! Aber bei welchem Programmieraufwand!



# Triangulation convexer Polygone: geht in $O(n)$ Zeit

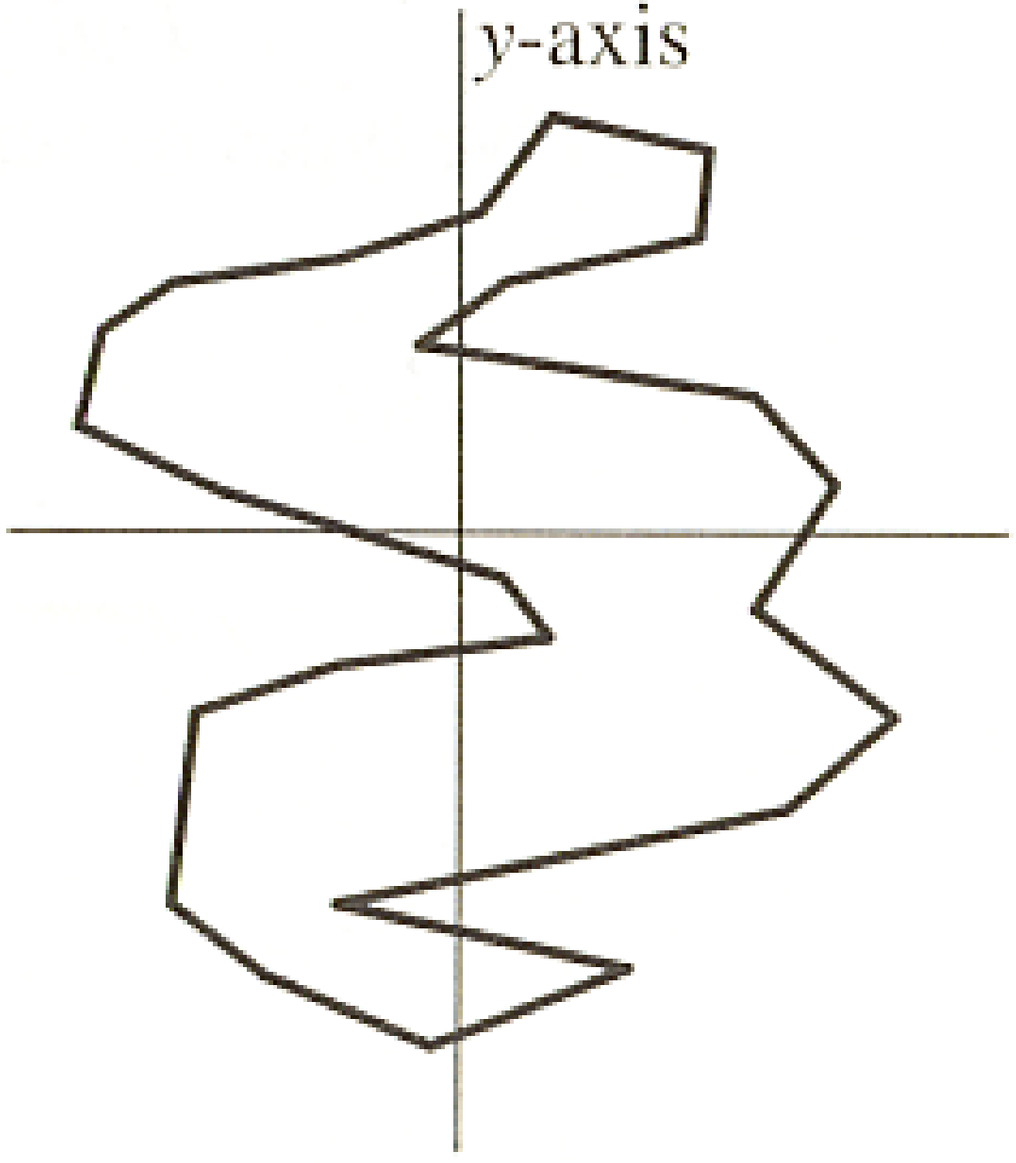


Das diese Triangulationen in linearer Zeit zu finden sind ist klar: Man beginnt an irgend einem Knoten und verbindet stes mit dem nächsten, noch nicht besuchten Nachbarn.

Zuerst eine Zerlegung des Polygons in konvexe Teilgebiete zu versuchen und dann in linearer Zeit weiter zu arbeiten, funktioniert leider nicht:
Es ist genauso schwer ein Polygon in seine konvexen Teile zu zerlegen, wie es zu triangulieren!

# Triangulation monotoner Polygone ist in $O(n)$ Zeit möglich

Ein Polygon ist *monoton*, wenn
der Weg vom obersten zum
untersten Punkt auf jeder Seite
stets nur abwärts oder waagerecht
verläuft, niemals jedoch aufwärts!

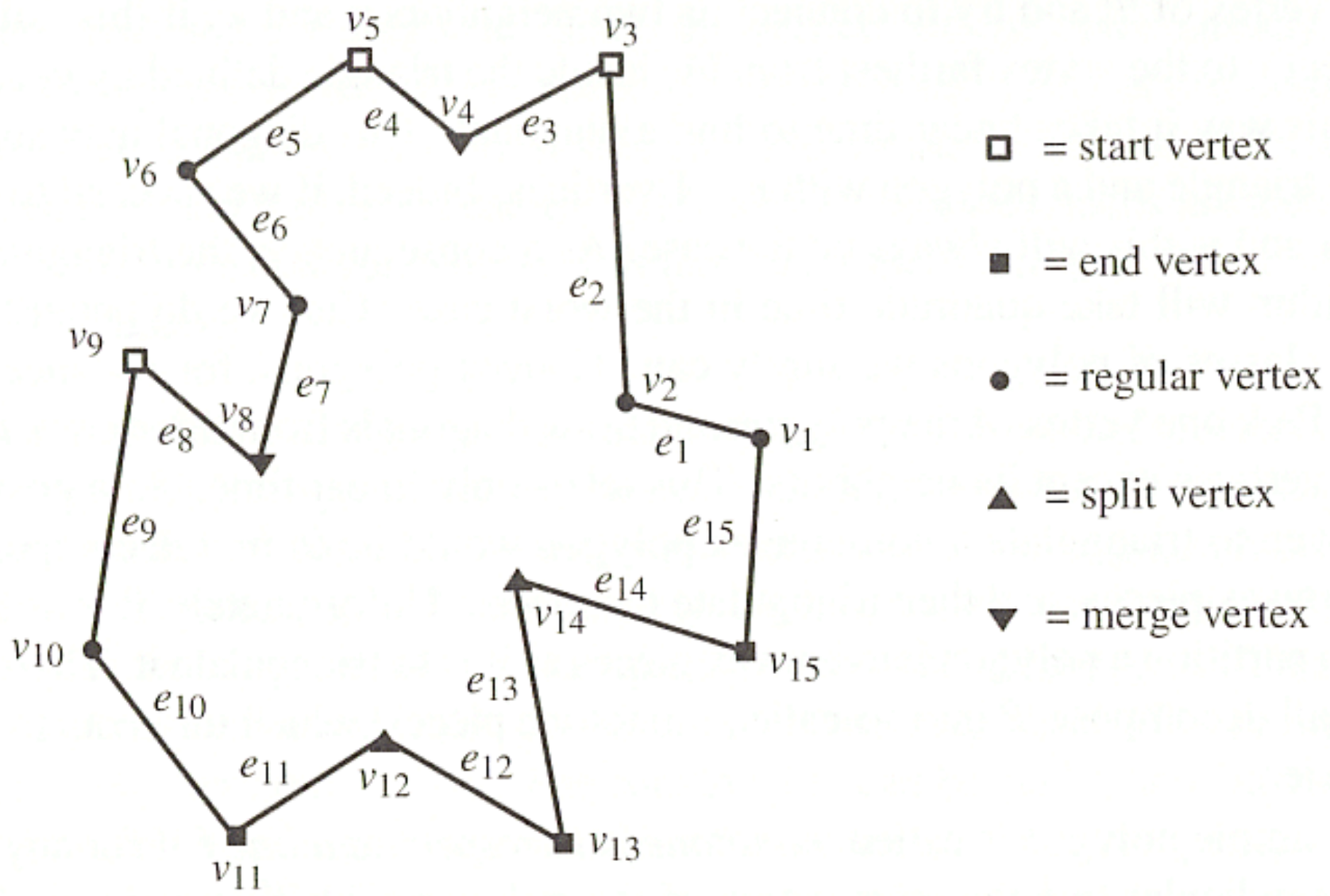



Man kann jedes Polygon in $O(n \ \ell og \ n)$
Zeit in monotone Teile zerlegen, und
monotone Polygone in linearer Zeit
triangulieren.

Also kann jedes einfache Polygon in
$O(n \ \ell og \ n)$ Zeit trianguliert werden.

Wir sehen uns einige Details an:

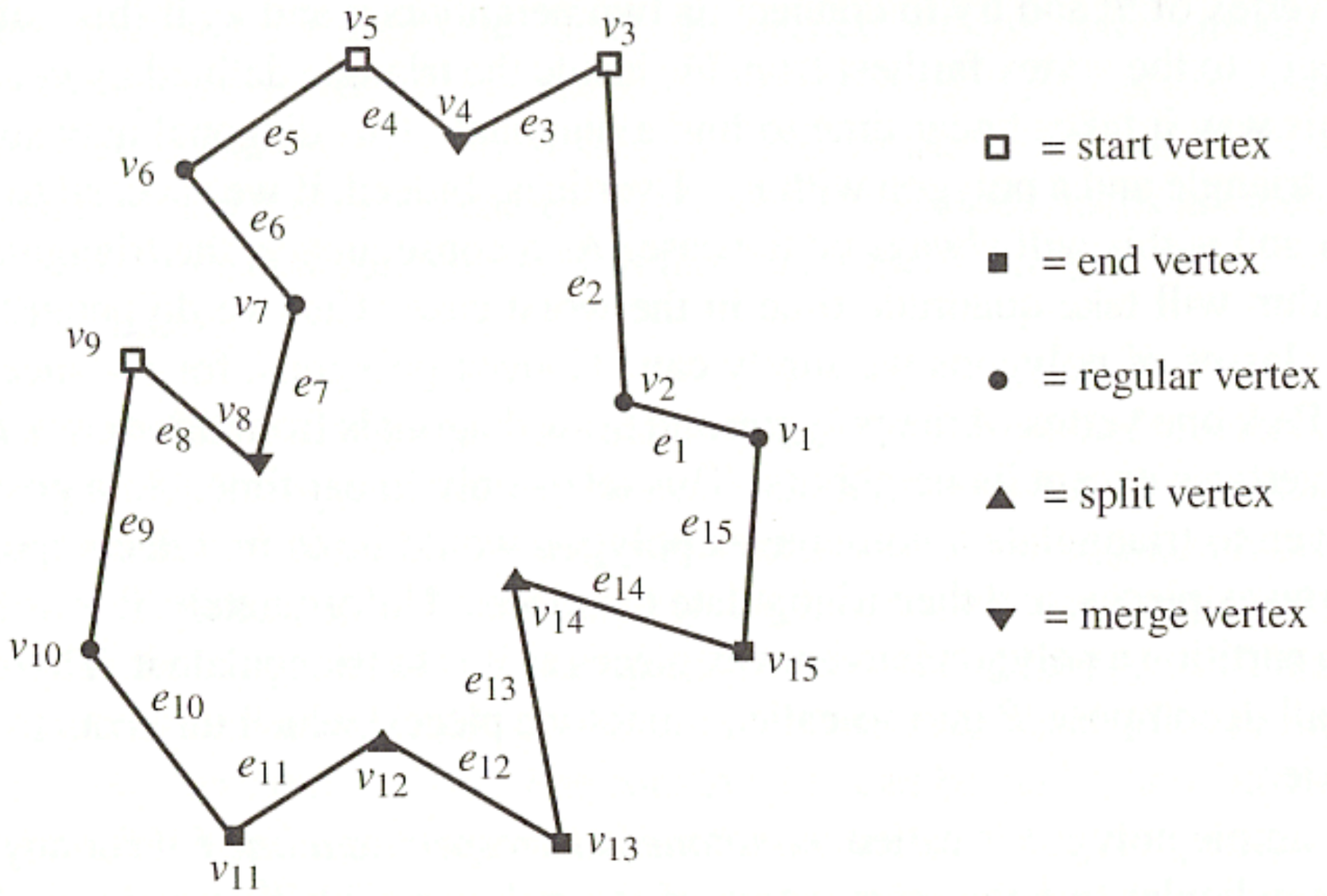
# Zum Finden der Partitionierung




Für Ecken $p := (p_x, p_y)$ und $q := (q_x, q_y)$ sei $p < q$ genau dann, wenn $(p_y < q_y)$ oder ($p_y = q_y$ und $p_x > q_x$)!

Im Beispiel gilt also $v_{13} < v_{14}$ und $v_{15} < v_{14}$.

# Zum Finden der Partitionierung




$p$ ist ***start***-Ecke, wenn für die direkten Nachbarn $x,y$ gilt: $x<p,\ y<p$ und der Innenwinkel ist kleiner als $\pi=180^\circ$, ist der Innenwinkel größer als $\pi$, so ist dies eine ***split***-Ecke. (andersherum bei ***end***-Ecken und ***merge***-Ecken). Diese vier Eck-Typen sind ***turn***-Ecken, alle anderen ***reguläre*** Ecken.

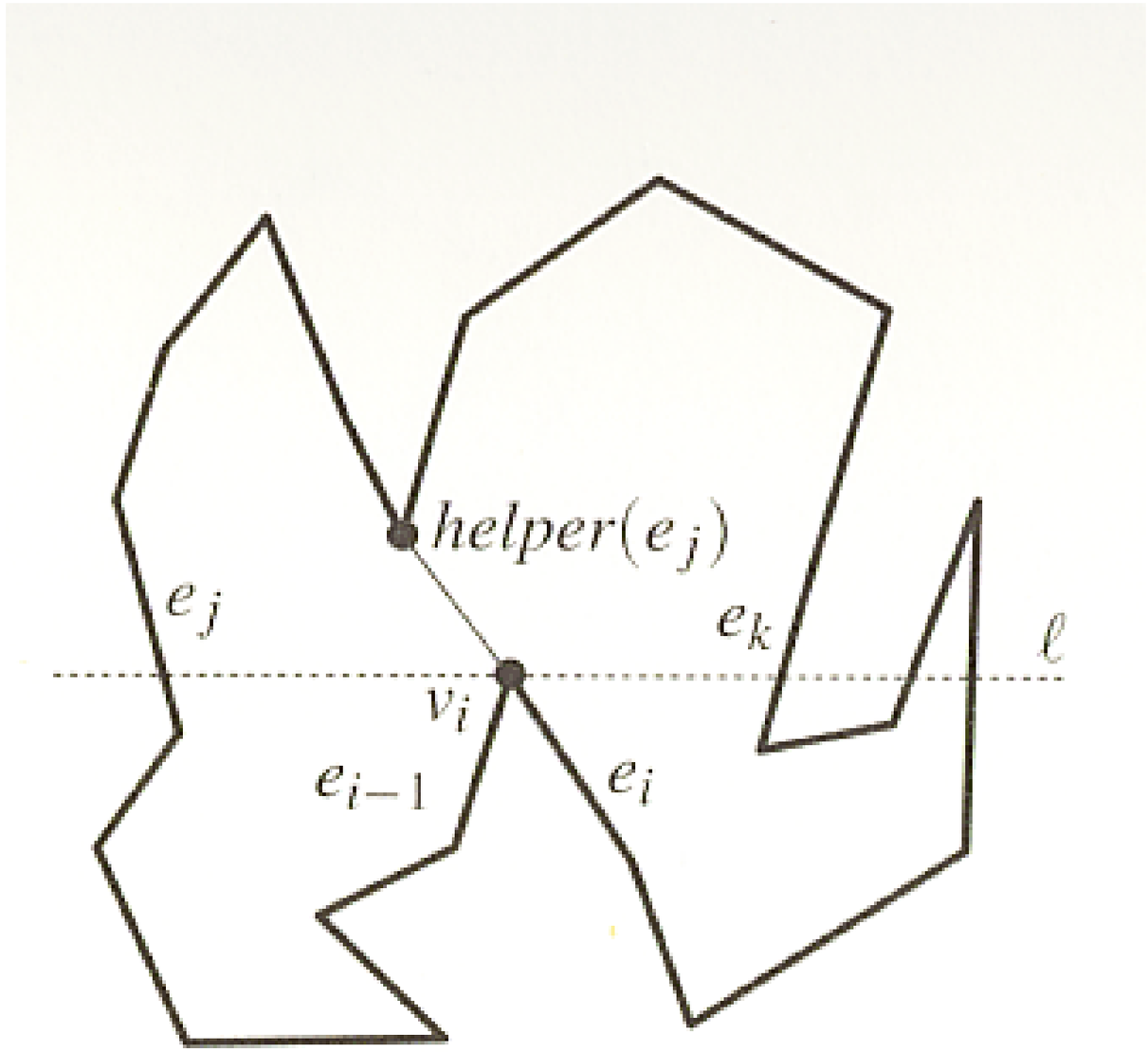
# Vermeiden von ***split***-Ecken und ***merge***-Ecken

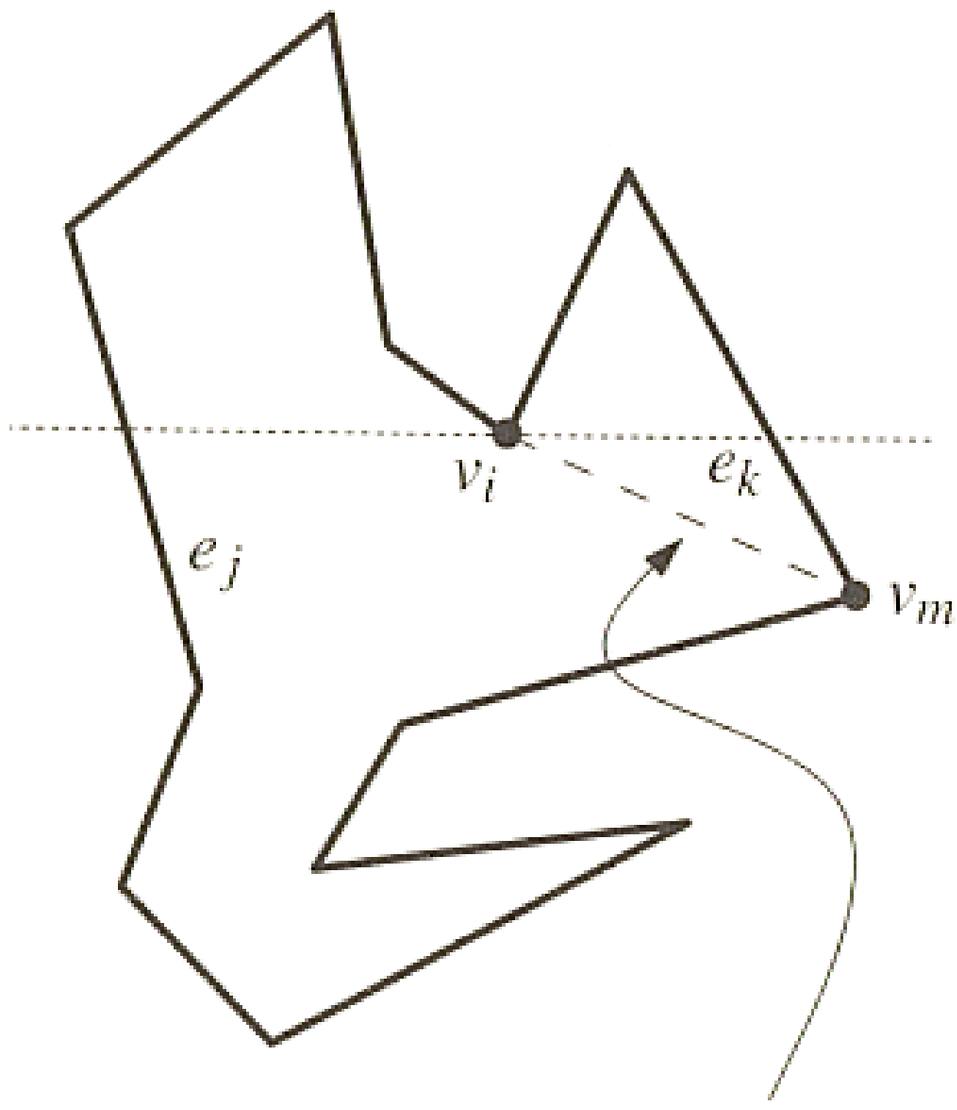
**Lemma:**
Ein Polygon ist monoton, wenn es weder ***split***- noch ***merge***-Ecken besitzt.

Um die Zerlegung in monotone Teile zu erhalten, muss man von jeder ***split***-Ecke eine Diagonale nach oben, und von jeder ***merge***-Ecke eine Diagonale nach unten finden.
Diese dürfen sich nicht kreuzen!

In einem scan von oben nach unten, werden alle Ecken in einer Prioritäts-Warteschlange (*priority-queue*) von Ereignis-Punkten gespeichert. Dadurch können die nächsten Ereignispunkte in $O(\ell og\ n)$ Zeit gefunden werden. Beim Besuch einer ***split***-Ecke, wird die am nächsten gelegene Ecke als Verbindungspunkt der Diagonalen gewählt. (Das vermeidet Überkreuzungen!)

# Diagonalen mit einem *scan-line*- oder *sweep*-Verfahren finden



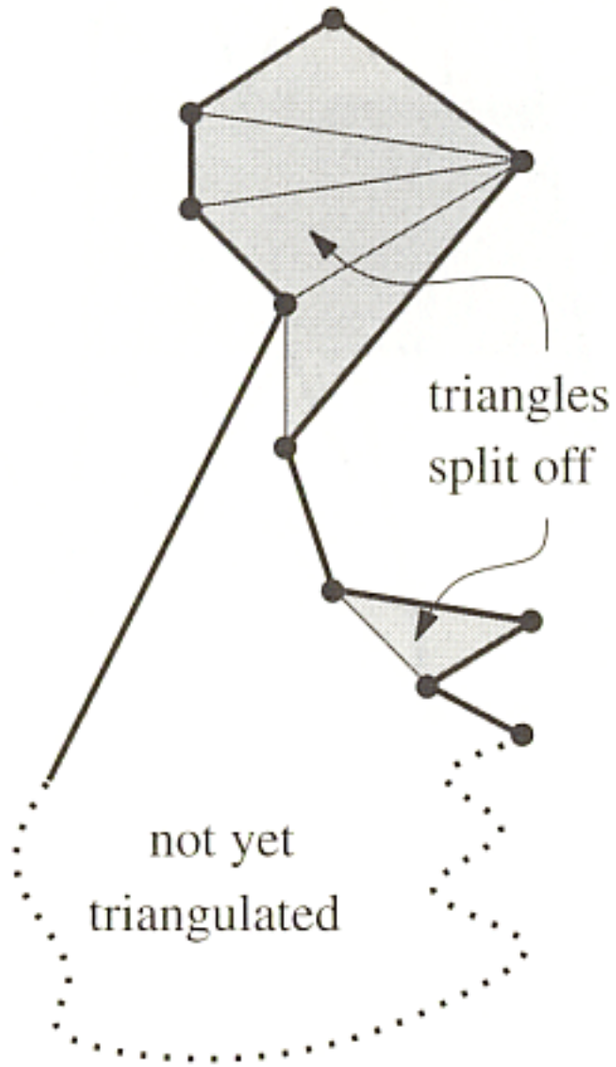




diagonal will be added when the sweep line reaches $v_m$

# $O(n \ \ell og \ n)$-Verfahren für Partition in monotone Polygone

**Theorem:**
Jedes einfache Polygon kann in $O(n \ \ell og \ n)$ Zeit in monotone Polygone partitioniert werden.

**Theorem:**
Jedes monotone Polygon kann in $O(n)$ Zeit trianguliert werden.




**Theorem:**
Jedes einfache Polygon kann in $O(n \ \ell og \ n)$ Zeit trianguliert werden.

**Theorem:**
Jedes Polygon mit Löchern kann in $O(n \ \ell og \ n)$ Zeit trianguliert werden.

# Wächterpositionen festlegen

Um die Positionen der Wächter/Kameras im Kunstgalerie-Problem (*art gallery problem*) festzulegen genügt $O(n \ \ell og \ n)$ Zeit:

1. Triangulation bestimmen: $O(n \ \ell og \ n)$

2. 3-Färbung der Ecken: $O(n)$

3. Rundum-Kameras an Ecken gleicher Farbe montieren

# Geht es noch besser?

Für die Triangulation von Polygonen mit Löchern(!) wurde eine unter Schranke von $\Omega(n\ \ell og\ \ell og\ n)$ bewiesen!
1988 haben Tarjan und Van Wyk eine gleichartige obere Schranke gezeigt, deren Beweis von Kirkpatrick et al. 1990 vereinfacht wurde:

**Theorem:**
Jedes Polygon kann in $O(n\ \ell og\ \ell og\ n)$ Zeit trianguliert werden!

Unter Verwendung von Randomisierten Verfahren, konnten Clarkson et a. 1989, Seidel 1991 und Devillers 1992 zeigen, dass man bei einfachen Polygonen mit noch  weniger Zeit auskommen kann:

**Theorem:**
Jedes Polygon kann in $O(n\ \ell og^*(n))$ Zeit trianguliert werden!

# Was ist das bisher Beste?

Dabei gibt $\ell og^*(n)$ an, wie oft man den Logarithmus auf $n$ iteriert anwenden muss, um das letzte Mal noch einen Wert größer 1 zu erhalten, d.h.:

$$\ell og(\ell og^*(n)) < 1 < \ell og^*(n)$$

Das ist schon sehr gut, jedoch hatte Chazelle 1990/91 in einem recht komplizierten Beweis einen deterministischen Algorithmus angegeben, der das Triangulationsproblem für einfache Polygone in linearer Zeit löst!

**Theorem:**
Jedes einfache Polygon kann in $O(n)$ Zeit trianguliert werden.

# History of Art-Gallery Algorithms

| Running Time | Designers | Year | Technique |
|---|---|---|---|
| $O(n^2)$ | *Lennes* | 1911 | Recursive diagonal insertion |
| $O(n^3)$ | *Meisters* | 1975 | Ear cutting |
| O(n log n) | *Garey, Johnson, Preparata & Tarjan* | 1978 | Decomposition into monotone pieces |
| O(n log n) | *Chazelle* | 1982 | Divide & Conquer |
| O( n + r log r) where r is the # of reflex vertices of the Polygon | *Hertel & Mehlhorn* | 1983 | - |
| O(n log s) where s is the sinuosity of P | *Chazelle* | 1983 | - |
| O(n log log n) | *Tarjan & Van Wyk* | 1987 | Using involved data structures |
| O(n (1 + to)) where to is the # of free triangles in the output triangulation. | *Toussaint* | 1988 | Sleeve-searching (Output sensitive) |
| $O(n^2)$ | *ElGindy, Everett & Toussaint* | 1990 | Finding an ear in linear time via prune & search |
| O(n) | *Chazelle* | 1990 | - |
| O(kn) | Kong, Everett & *Toussaint* | 1990 | Graham Scan |

# Welche weiteren 2-3 Themen sollen noch behandelt werden?

- Durchschnitte von Halbebenen und incrementelles und randomisiertes Lineares Programmieren.

- Binary space partitions und der Painter's Algorithm (*rendering, hidden surface removal, shading*) aber keine Programme sondern Grundlegendes.

- Mehr zu konvexen Hüllen (auch 3-D).

- *Mesh generation, Quad-trees* und VLSI-outline etc.

- kürzeste Wege für Roboter bei Hindernissen (*obstacles*).

- Roboterbewegungen, Bewegungsplanung...

- Mehr zu Algorithmischen in/zu Graphen: Flüsse in Netzwerken, Netzwerk-Algorithmen, etc.