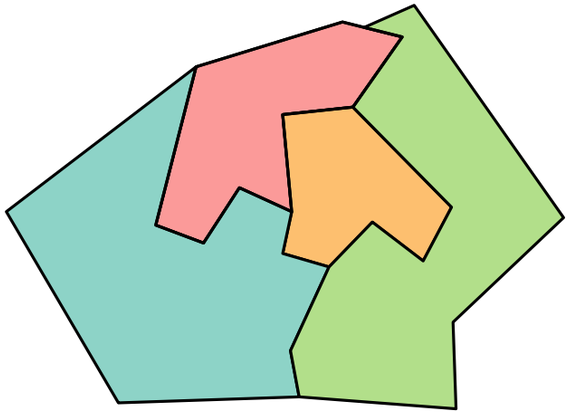
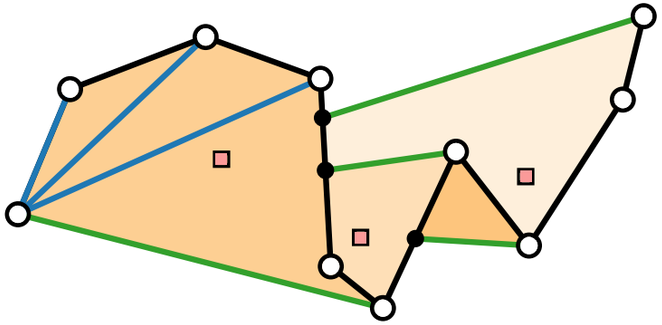
Universität Trier

Algorithmen für geographische Informationssysteme

10. Vorlesung
Flächenvereinfachung



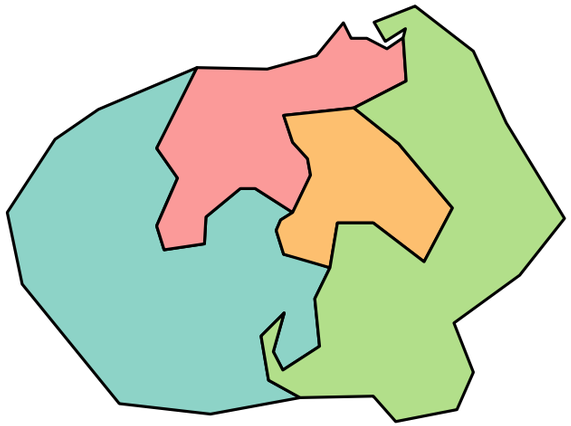
Teil I:
Problemstellung



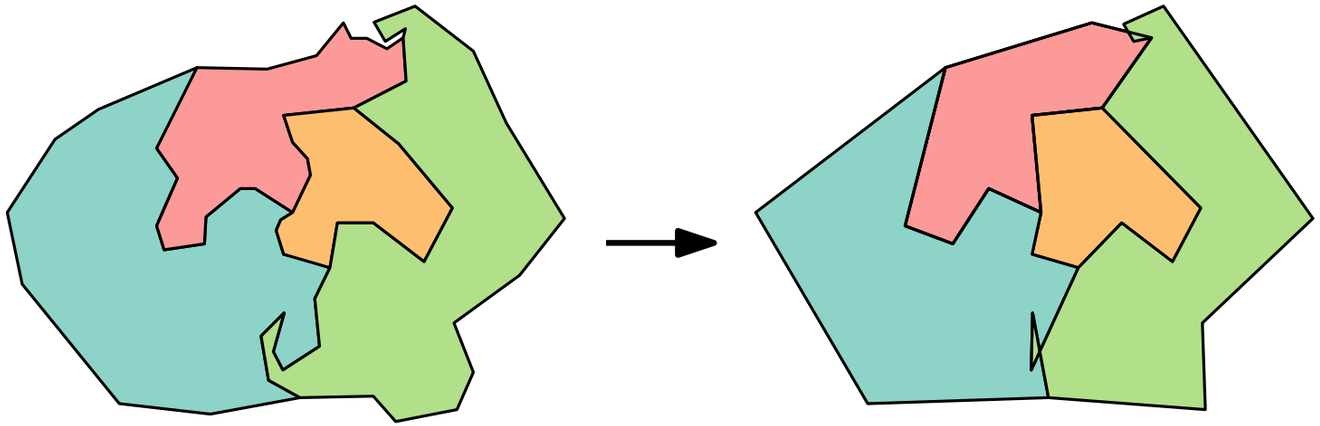
Philipp Kindermann

Wintersemester 2024/25

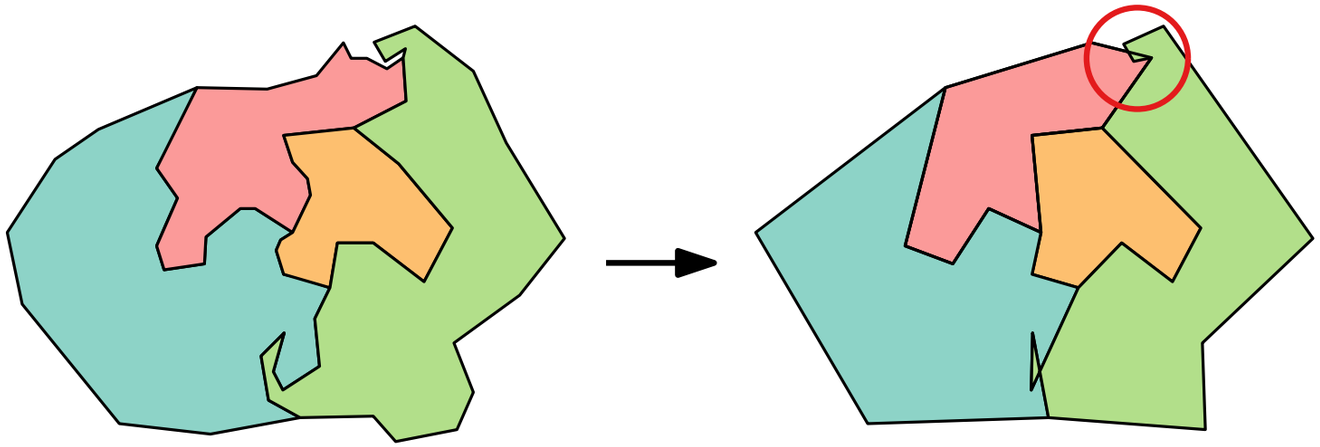
Pfadvereinfachung für Landkarten



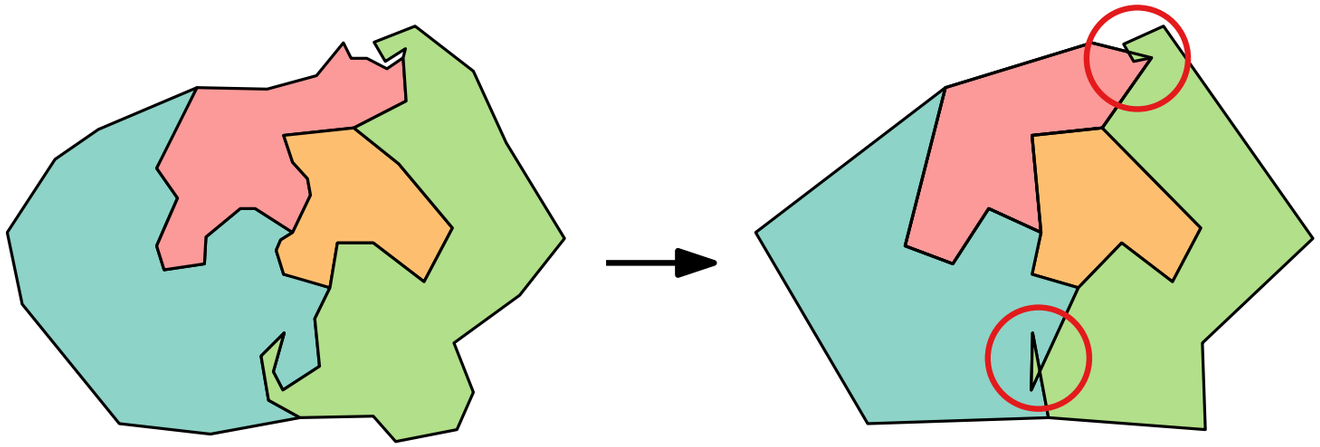
Pfadvereinfachung für Landkarten



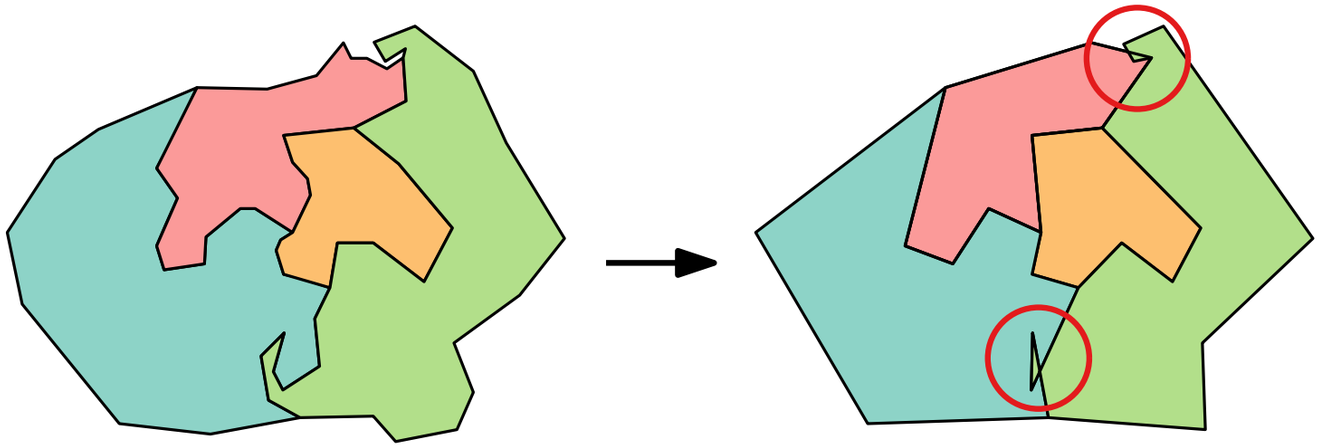
Pfadvereinfachung für Landkarten



Pfadvereinfachung für Landkarten

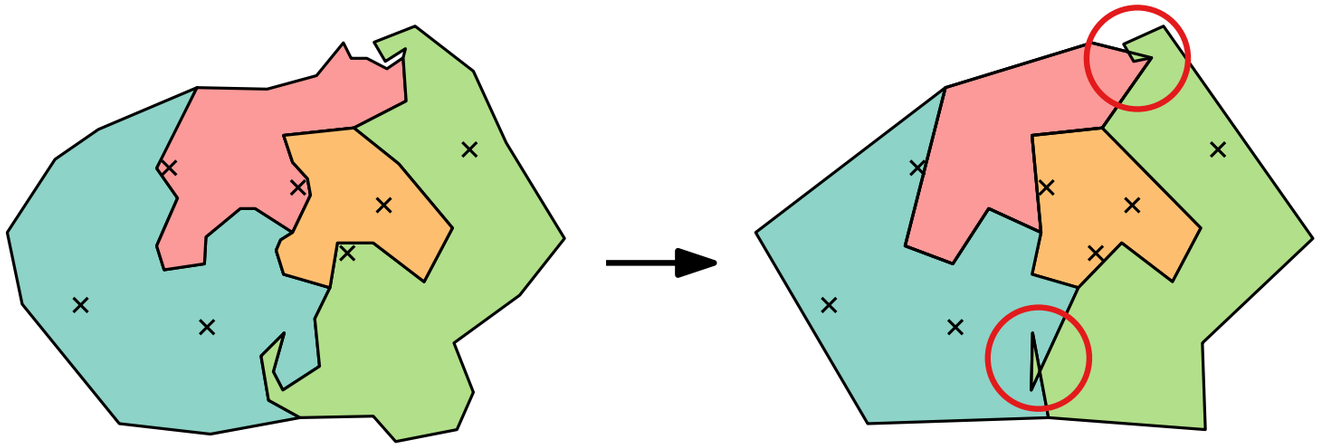


Pfadvereinfachung für Landkarten



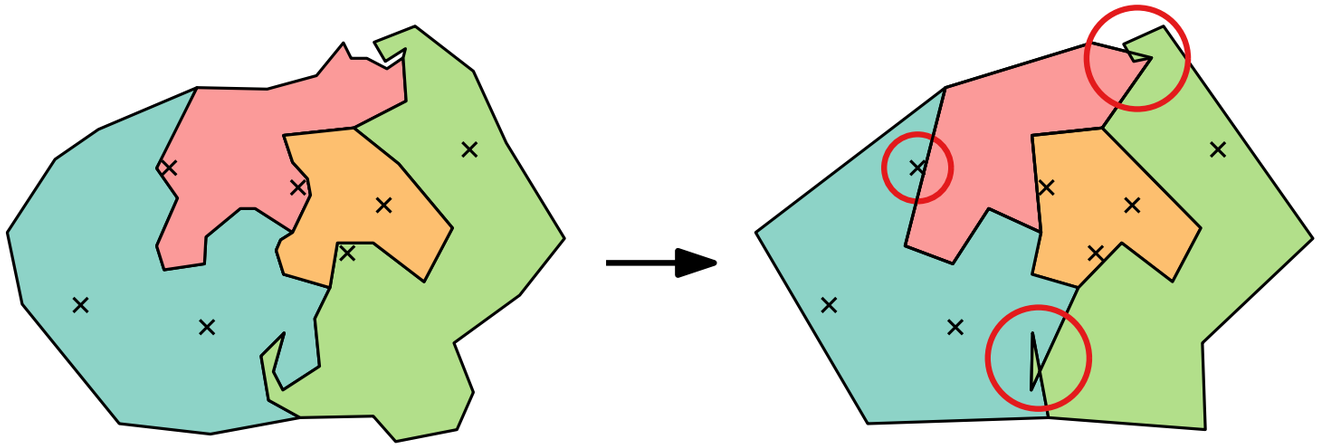
- Selbstschnitte und Schnitte mit anderen Pfaden möglich

Pfadvereinfachung für Landkarten



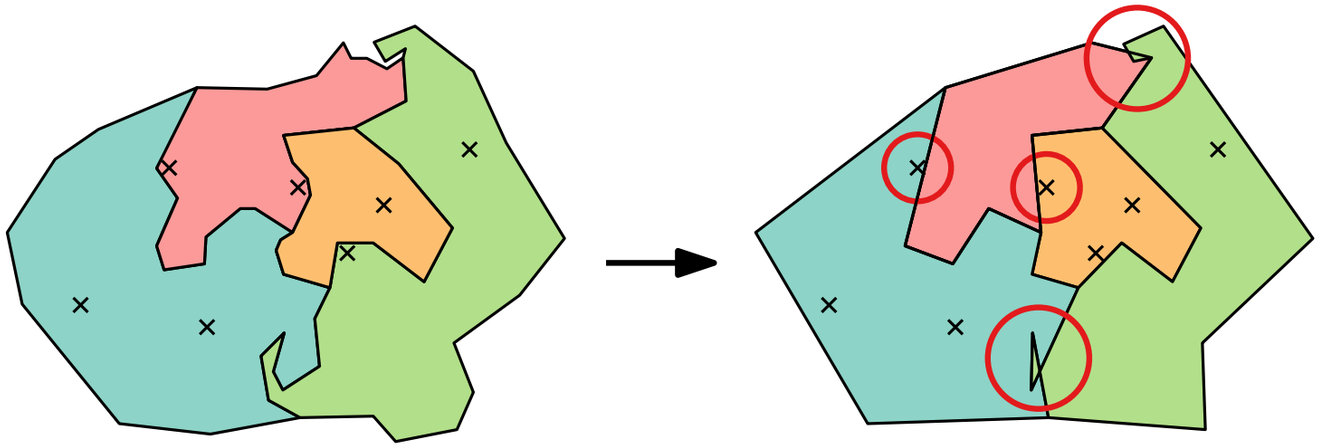
- Selbstschnitte und Schnitte mit anderen Pfaden möglich

Pfadvereinfachung für Landkarten



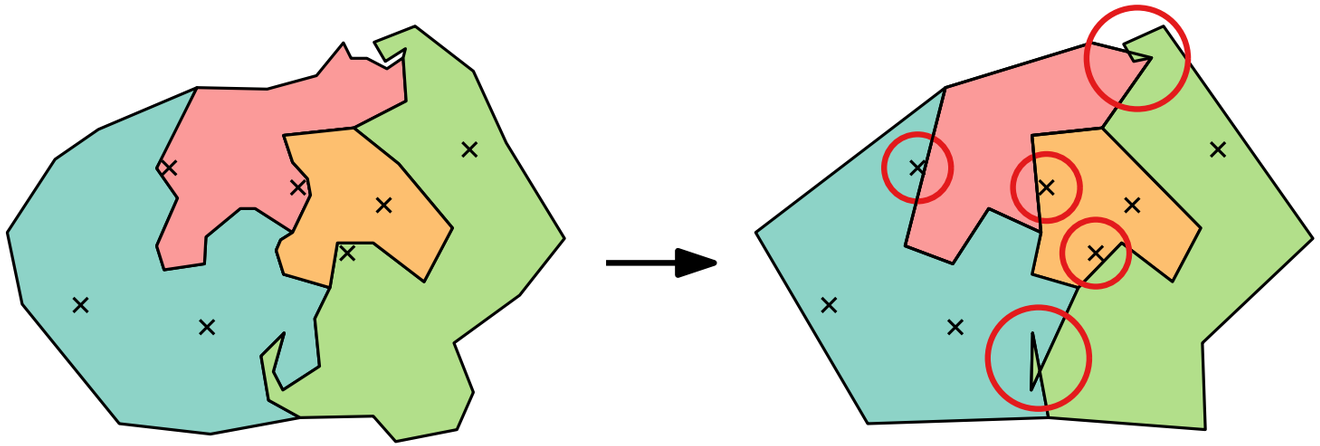
- Selbstschnitte und Schnitte mit anderen Pfaden möglich

Pfadvereinfachung für Landkarten



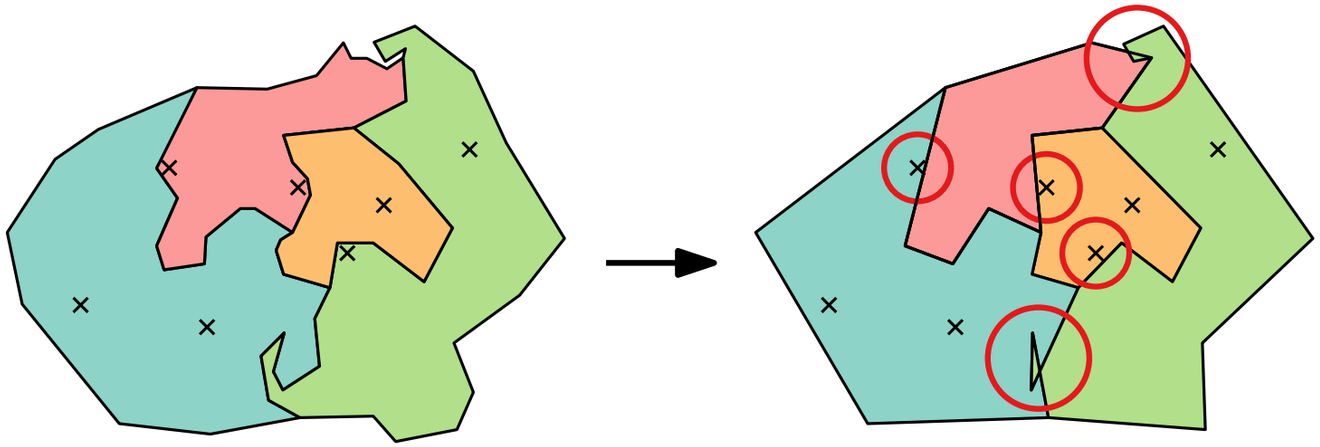
- Selbstschnitte und Schnitte mit anderen Pfaden möglich

Pfadvereinfachung für Landkarten



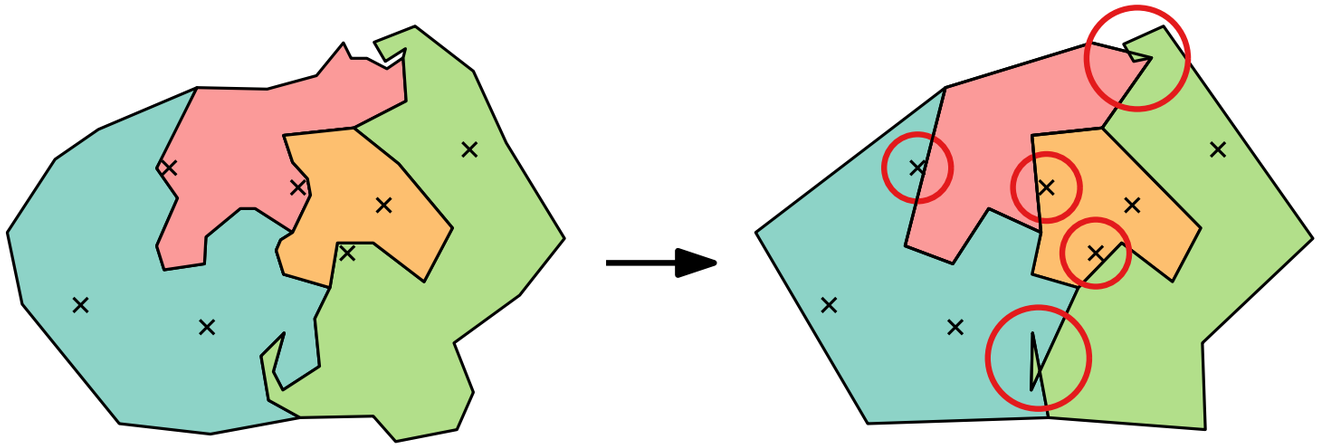
- Selbstschnitte und Schnitte mit anderen Pfaden möglich

Pfadvereinfachung für Landkarten



- Selbstschnitte und Schnitte mit anderen Pfaden möglich
- Punkte (z.B. Städte) können auf falsche Seite gelangen

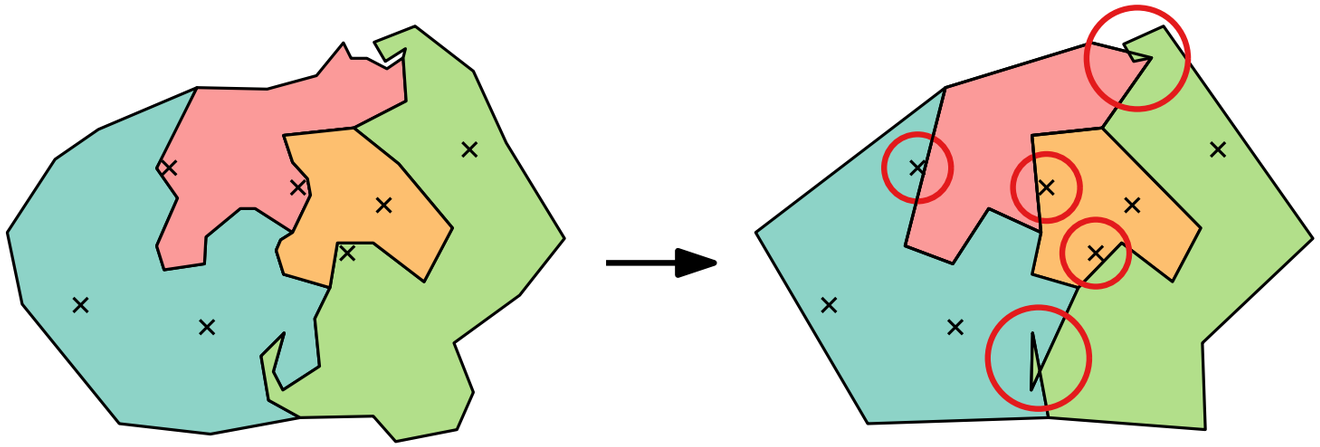
Pfadvereinfachung für Landkarten



- Selbstschnitte und Schnitte mit anderen Pfaden möglich
- Punkte (z.B. Städte) können auf falsche Seite gelangen

➡ benötigen topologisch korrekten Vereinfachungsalgorithmus

Pfadvereinfachung für Landkarten

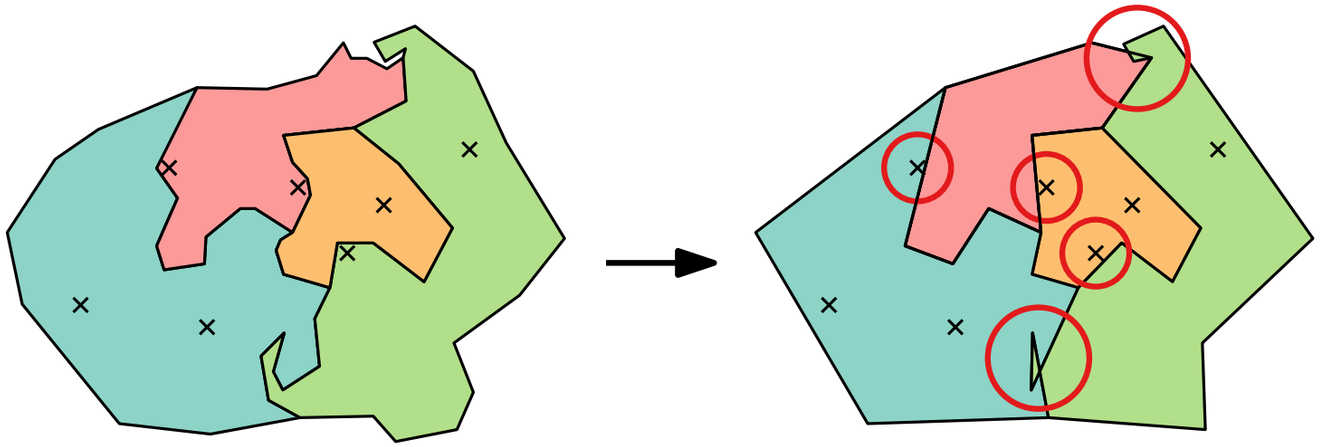


- Selbstschnitte und Schnitte mit anderen Pfaden möglich
- Punkte (z.B. Städte) können auf falsche Seite gelangen

➡ benötigen topologisch korrekten Vereinfachungsalgorithmus

Ansatz 1: nachträgliches Korrigieren

Pfadvereinfachung für Landkarten



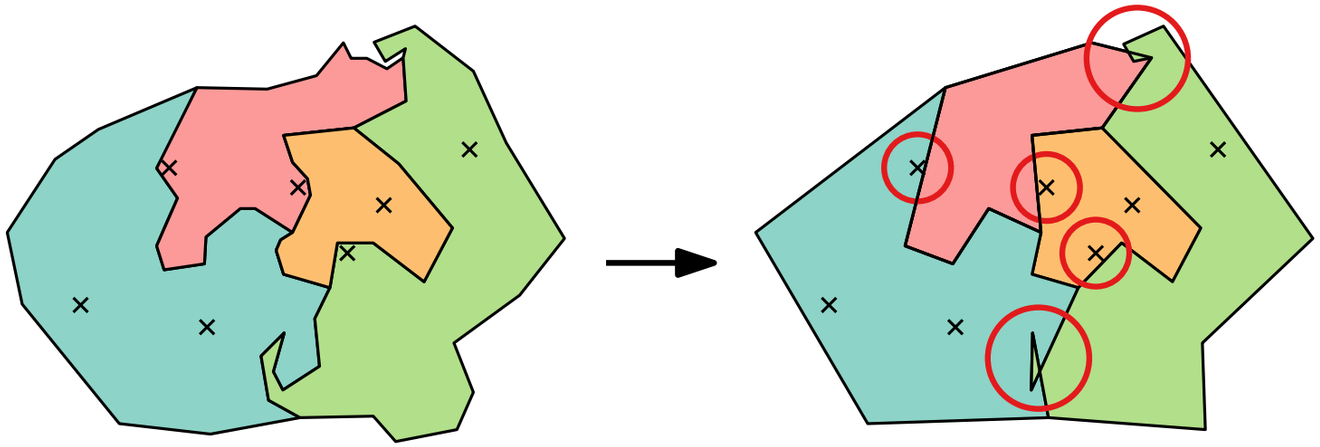
- Selbstschnitte und Schnitte mit anderen Pfaden möglich
- Punkte (z.B. Städte) können auf falsche Seite gelangen

➡ benötigen topologisch korrekten Vereinfachungsalgorithmus

Ansatz 1: nachträgliches Korrigieren

Ansatz 2: Vermeiden durch „schlauere“ Algorithmen

Pfadvereinfachung für Landkarten



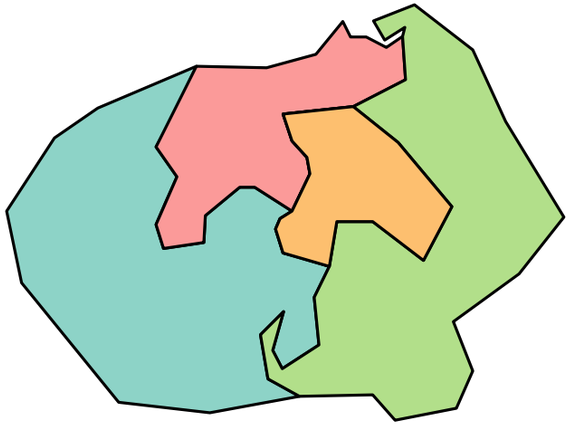
- Selbstschnitte und Schnitte mit anderen Pfaden möglich
- Punkte (z.B. Städte) können auf falsche Seite gelangen

➡ benötigen topologisch korrekten Vereinfachungsalgorithmus

Ansatz 1: nachträgliches Korrigieren

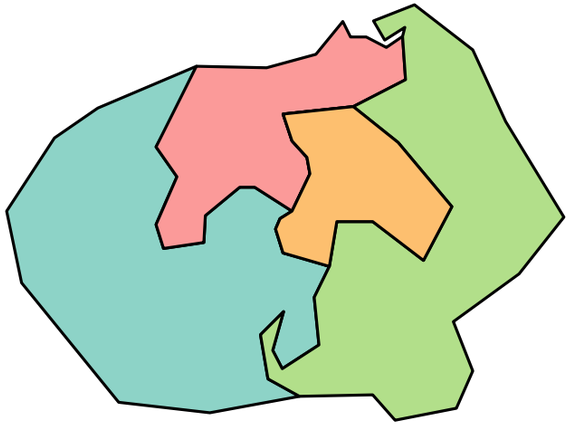
Ansatz 2: Vermeiden durch „schlauere“ Algorithmen

Notation



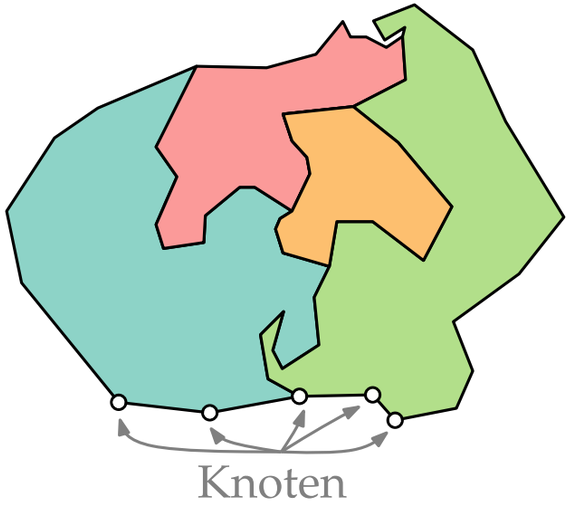
Notation

- Karte modelliert als Rechtecksunterteilung, d.h. als planarer Graph ➡ Knoten, Kanten, Facetten



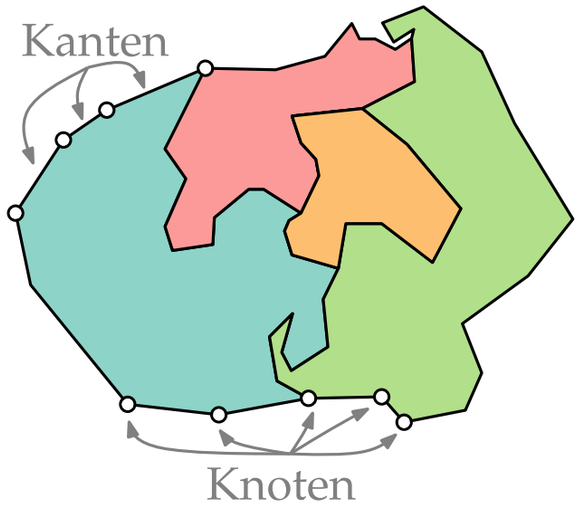
Notation

- Karte modelliert als Rechtecksunterteilung, d.h. als planarer Graph ➡ Knoten, Kanten, Facetten



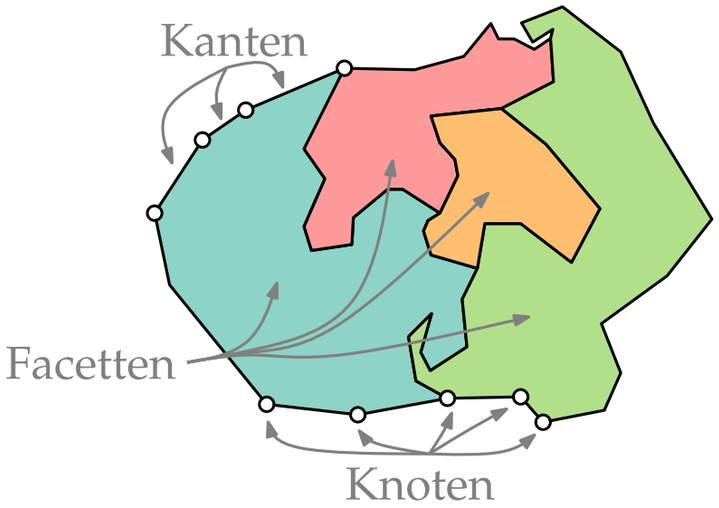
Notation

- Karte modelliert als Rechtecksunterteilung, d.h. als planarer Graph ➡ Knoten, Kanten, Facetten



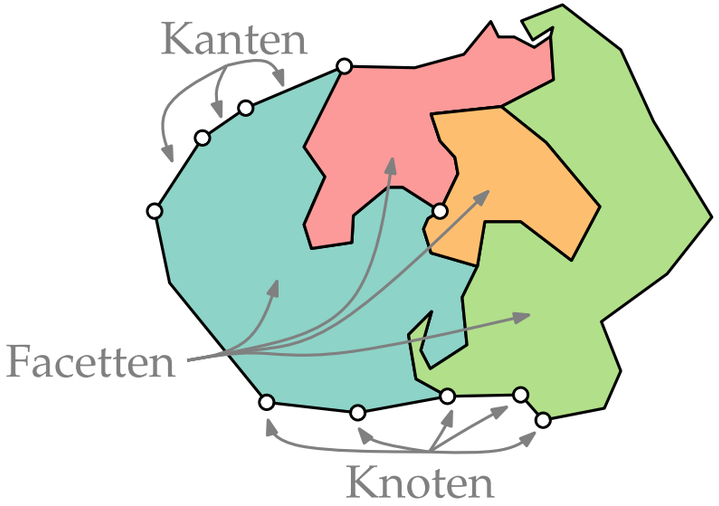
Notation

- Karte modelliert als Rechtecksunterteilung, d.h. als planarer Graph ➡ Knoten, Kanten, Facetten



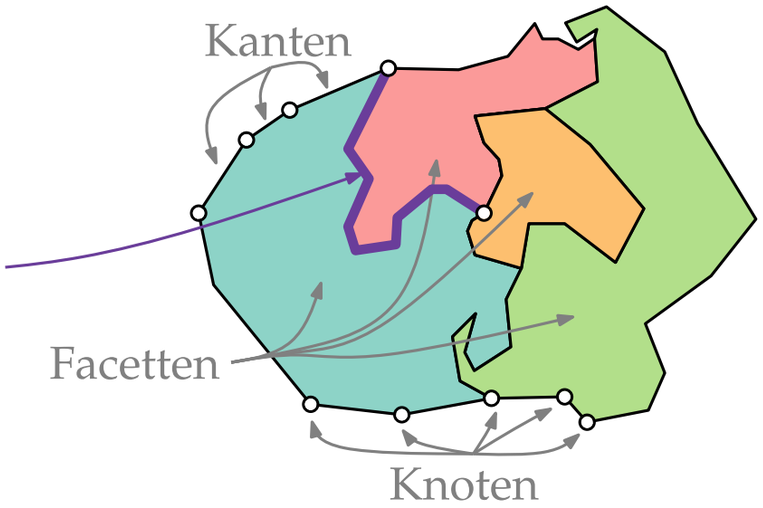
Notation

- Karte modelliert als Rechtecksunterteilung, d.h. als planarer Graph ➡ Knoten, Kanten, Facetten
- Knotengrade üblicherweise 2 oder 3



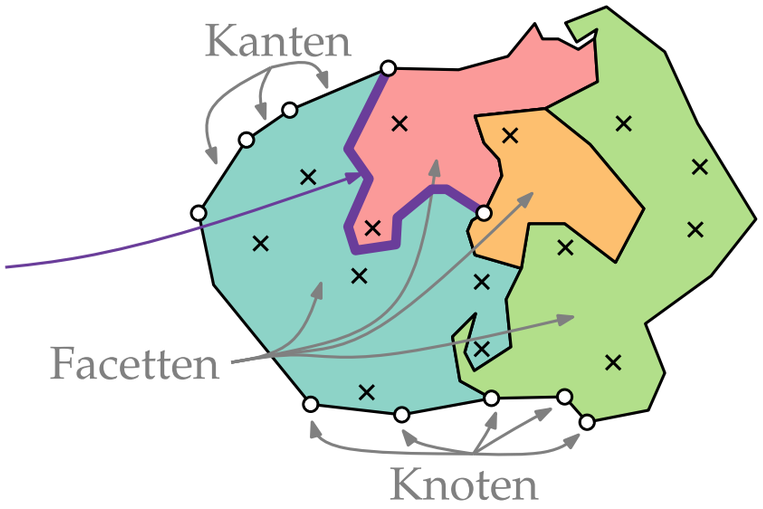
Notation

- Karte modelliert als Rechtecksunterteilung, d.h. als planarer Graph ➡ Knoten, Kanten, Facetten
- Knotengrade üblicherweise 2 oder 3
- maximale Pfade von Grad-2 Knoten bilden Kantenzüge



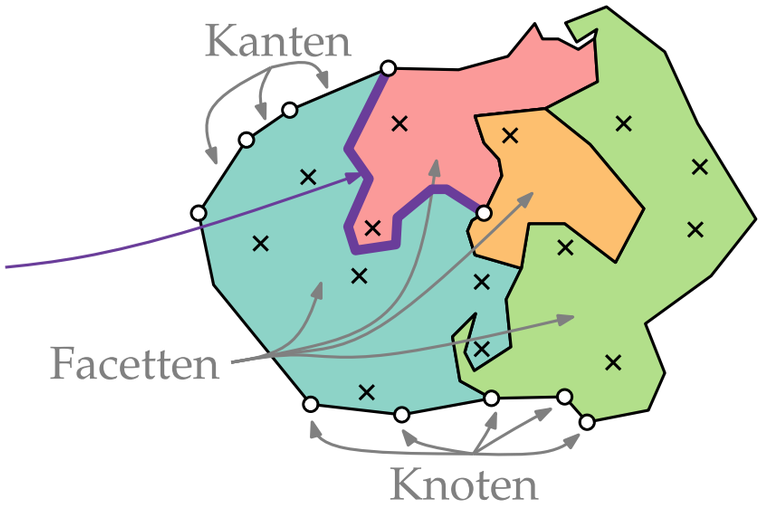
Notation

- Karte modelliert als Rechtecksunterteilung, d.h. als planarer Graph ➡ Knoten, Kanten, Facetten
- Knotengrade üblicherweise 2 oder 3
- maximale Pfade von Grad-2 Knoten bilden Kantenzüge
- zusätzlich Menge P von Punkten ($\times$)



Notation

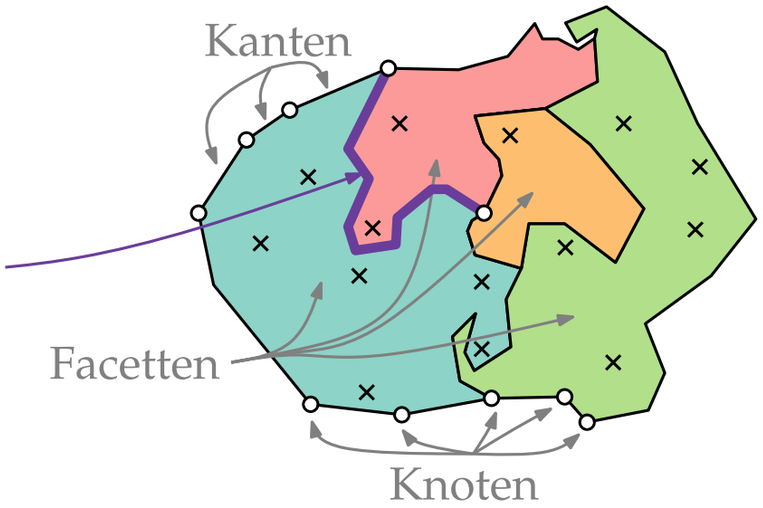
- Karte modelliert als Rechtecksunterteilung, d.h. als planarer Graph ➡ Knoten, Kanten, Facetten
- Knotengrade üblicherweise 2 oder 3
- maximale Pfade von Grad-2 Knoten bilden Kantenzüge
- zusätzlich Menge P von Punkten ($\times$)



Ziel:

Notation

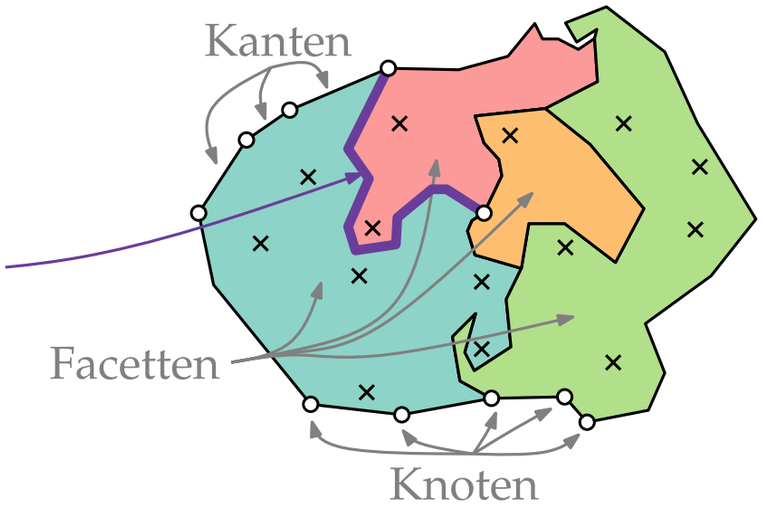
- Karte modelliert als Rechtecksunterteilung, d.h. als planarer Graph ➡ Knoten, Kanten, Facetten
- Knotengrade üblicherweise 2 oder 3
- maximale Pfade von Grad-2 Knoten bilden Kantenzüge
- zusätzlich Menge P von Punkten ($\times$)



Ziel: Ersetze alle Kantenzüge C durch vereinfachte Kantenzüge C' mit

Notation

- Karte modelliert als Rechtecksunterteilung, d.h. als planarer Graph ➡ Knoten, Kanten, Facetten
- Knotengrade üblicherweise 2 oder 3
- maximale Pfade von Grad-2 Knoten bilden Kantenzüge
- zusätzlich Menge P von Punkten ($\times$)

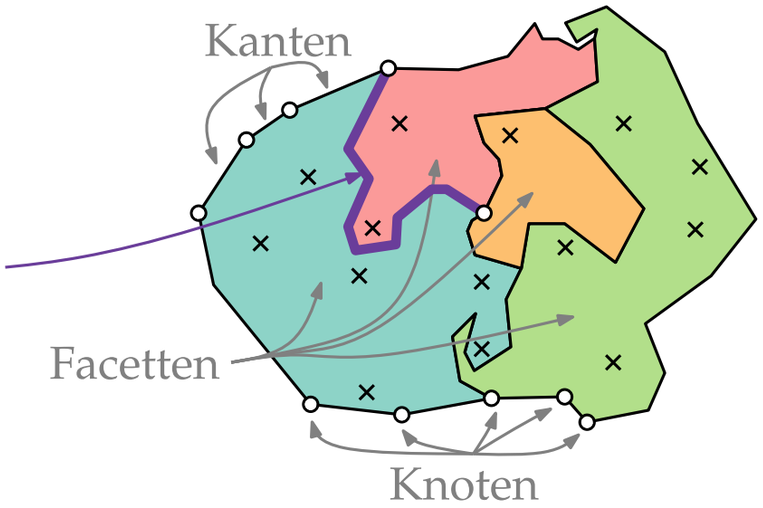


Ziel: Ersetze alle Kantenzüge C durch vereinfachte Kantenzüge C' mit

- kein Punkt von C ist weiter als ε von C' entfernt

Notation

- Karte modelliert als Rechtecksunterteilung, d.h. als planarer Graph ➡ Knoten, Kanten, Facetten
- Knotengrade üblicherweise 2 oder 3
- maximale Pfade von Grad-2 Knoten bilden Kantenzüge
- zusätzlich Menge P von Punkten ($\times$)

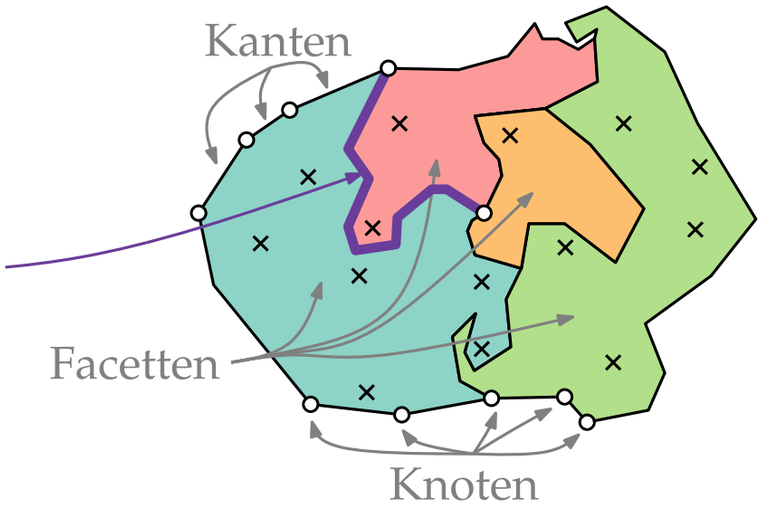


Ziel: Ersetze alle Kantenzüge C durch vereinfachte Kantenzüge C' mit

- kein Punkt von C ist weiter als ε von C' entfernt
- C' ist einfach (d.h. schnittfrei)

Notation

- Karte modelliert als Rechtecksunterteilung, d.h. als planarer Graph ➡ Knoten, Kanten, Facetten
- Knotengrade üblicherweise 2 oder 3
- maximale Pfade von Grad-2 Knoten bilden Kantenzüge
- zusätzlich Menge P von Punkten ($\times$)

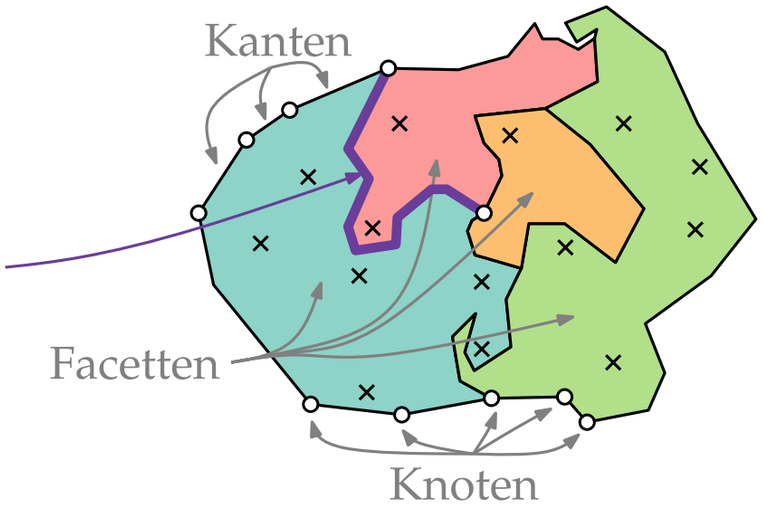


Ziel: Ersetze alle Kantenzüge C durch vereinfachte Kantenzüge C' mit

- kein Punkt von C ist weiter als ε von C' entfernt
- C' ist einfach (d.h. schnittfrei)
- C' schneidet keinen weiteren Kantenzug

Notation

- Karte modelliert als Rechtecksunterteilung, d.h. als planarer Graph ➡ Knoten, Kanten, Facetten
- Knotengrade üblicherweise 2 oder 3
- maximale Pfade von Grad-2 Knoten bilden Kantenzüge
- zusätzlich Menge P von Punkten ($\times$)

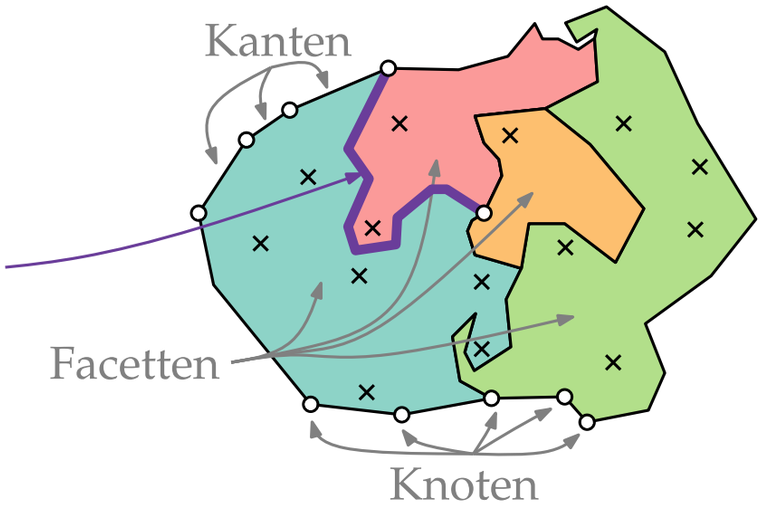


Ziel: Ersetze alle Kantenzüge C durch vereinfachte Kantenzüge C' mit

- kein Punkt von C ist weiter als ε von C' entfernt
- C' ist einfach (d.h. schnittfrei)
- C' schneidet keinen weiteren Kantenzug
- alle Punkte in P liegen auf den gleichen Seiten bzgl. C und C', d.h. Punkte bleiben in den richtigen Facetten

Notation

- Karte modelliert als Rechtecksunterteilung, d.h. als planarer Graph ➡ Knoten, Kanten, Facetten
- Knotengrade üblicherweise 2 oder 3
- maximale Pfade von Grad-2 Knoten bilden Kantenzüge
- zusätzlich Menge P von Punkten ($\times$)



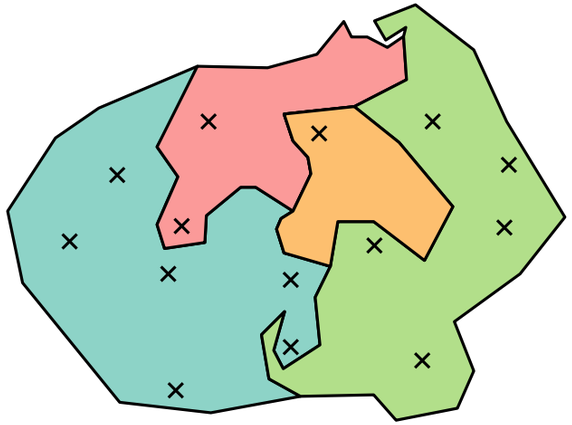
Ziel: Ersetze alle Kantenzüge C durch vereinfachte Kantenzüge C' mit

- kein Punkt von C ist weiter als ε von C' entfernt
- C' ist einfach (d.h. schnittfrei)
- C' schneidet keinen weiteren Kantenzug
- alle Punkte in P liegen auf den gleichen Seiten bzgl. C und C', d.h. Punkte bleiben in den richtigen Facetten

➡ solch ein Kantenzug C' heißt **konsistente** Vereinfachung von C

Algorithmus von de Berg et al. (1998)

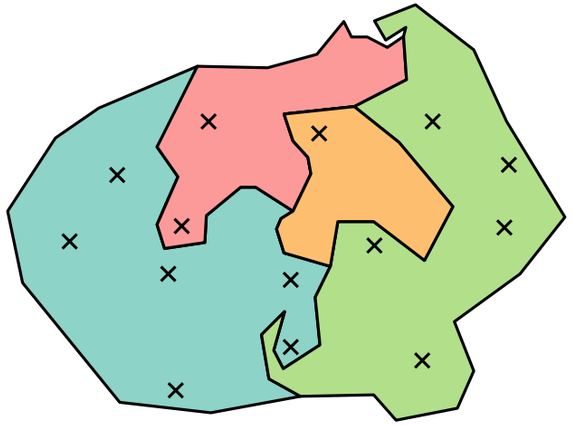
Idee:



Algorithmus von de Berg et al. (1998)

Idee:

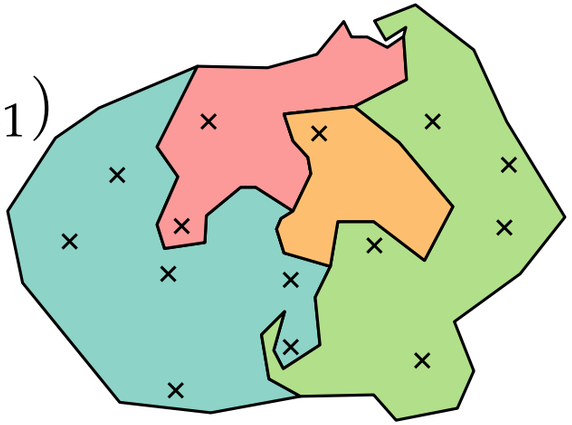
- graphbasierter Algorithmus (Imai & Iri) als Grundlage



Algorithmus von de Berg et al. (1998)

Idee:

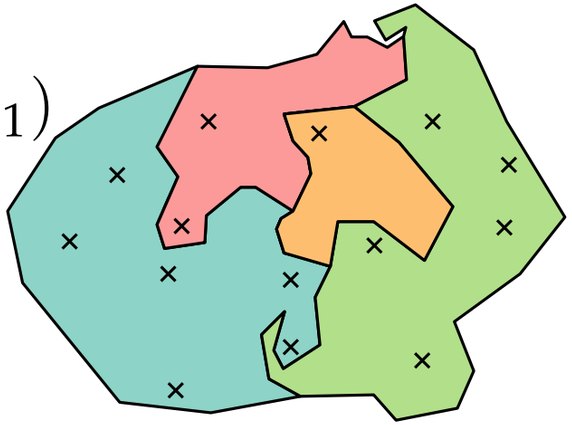
- graphbasierter Algorithmus (Imai & Iri) als Grundlage
- **zulässige** Abkürzungen bzgl. Fehler ε bilden $G_1 = (V, A_1)$



Algorithmus von de Berg et al. (1998)

Idee:

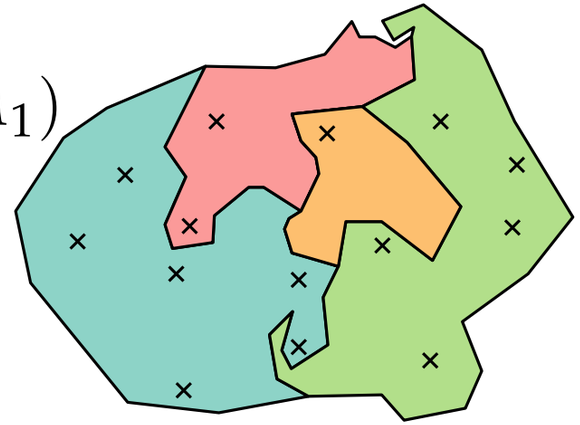
- graphbasierter Algorithmus (Imai & Iri) als Grundlage
- **zulässige** Abkürzungen bzgl. Fehler ε bilden $G_1 = (V, A_1)$
- **konsistente** Abkürzungen bzgl. P bilden $G_2 = (V, A_2)$



Algorithmus von de Berg et al. (1998)

Idee:

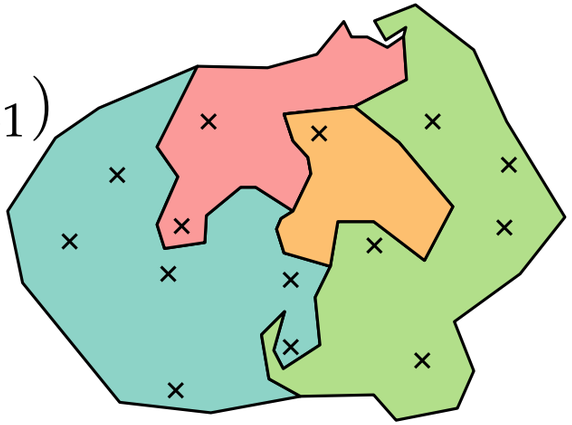
- graphbasierter Algorithmus (Imai & Iri) als Grundlage
- **zulässige** Abkürzungen bzgl. Fehler ε bilden $G_1 = (V, A_1)$
- **konsistente** Abkürzungen bzgl. P bilden $G_2 = (V, A_2)$
- kürzester Weg in Graph $G = (V, A_1 \cap A_2)$ liefert optimale zulässige und konsistente Vereinfachung



Algorithmus von de Berg et al. (1998)

Idee:

- graphbasierter Algorithmus (Imai & Iri) als Grundlage
- **zulässige** Abkürzungen bzgl. Fehler ε bilden $G_1 = (V, A_1)$
- **konsistente** Abkürzungen bzgl. P bilden $G_2 = (V, A_2)$
- kürzester Weg in Graph $G = (V, A_1 \cap A_2)$ liefert optimale zulässige und konsistente Vereinfachung

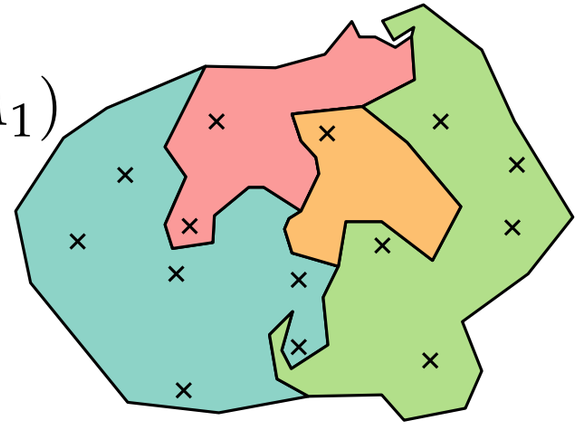


Kantenzug $C = (v_1, \ldots, v_n)$ und Vereinfachung C' heißen **konsistent bzgl.** P, wenn es Kantenzug B von v_n nach v_1 gibt, der C und C' zu einfachem Polygonen abschließt, das die gleiche Teilmenge von P einschließen.

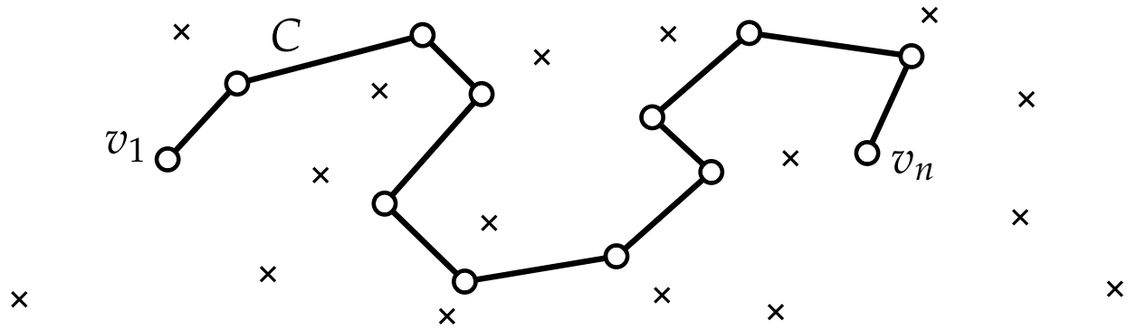
Algorithmus von de Berg et al. (1998)

Idee:

- graphbasierter Algorithmus (Imai & Iri) als Grundlage
- **zulässige** Abkürzungen bzgl. Fehler ε bilden $G_1 = (V, A_1)$
- **konsistente** Abkürzungen bzgl. P bilden $G_2 = (V, A_2)$
- kürzester Weg in Graph $G = (V, A_1 \cap A_2)$ liefert optimale zulässige und konsistente Vereinfachung



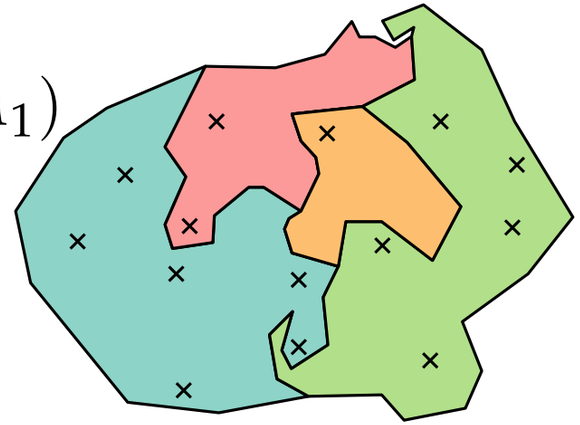
Kantenzug $C = (v_1, \ldots, v_n)$ und Vereinfachung C' heißen **konsistent bzgl.** P, wenn es Kantenzug B von v_n nach v_1 gibt, der C und C' zu einfachem Polygonen abschließt, das die gleiche Teilmenge von P einschließen.



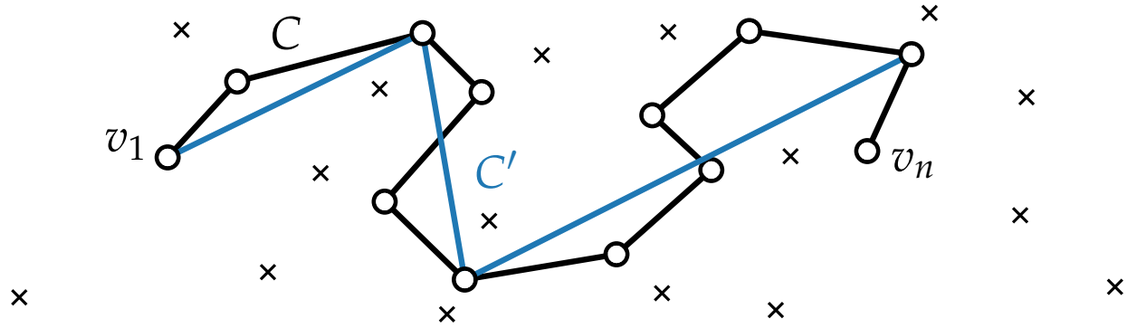
Algorithmus von de Berg et al. (1998)

Idee:

- graphbasierter Algorithmus (Imai & Iri) als Grundlage
- **zulässige** Abkürzungen bzgl. Fehler ε bilden $G_1 = (V, A_1)$
- **konsistente** Abkürzungen bzgl. P bilden $G_2 = (V, A_2)$
- kürzester Weg in Graph $G = (V, A_1 \cap A_2)$ liefert optimale zulässige und konsistente Vereinfachung



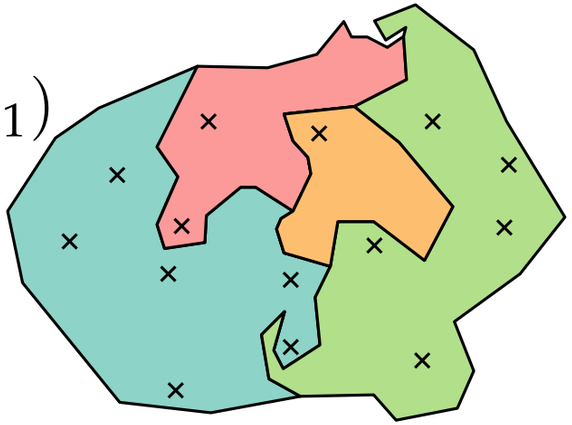
Kantenzug $C = (v_1, \ldots, v_n)$ und Vereinfachung C' heißen **konsistent bzgl.** P, wenn es Kantenzug B von v_n nach v_1 gibt, der C und C' zu einfachem Polygonen abschließt, das die gleiche Teilmenge von P einschließen.



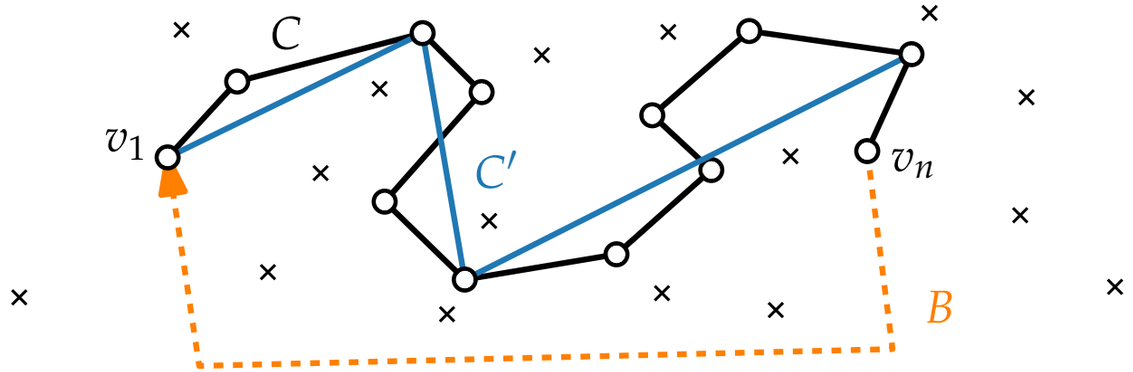
Algorithmus von de Berg et al. (1998)

Idee:

- graphbasierter Algorithmus (Imai & Iri) als Grundlage
- **zulässige** Abkürzungen bzgl. Fehler ε bilden $G_1 = (V, A_1)$
- **konsistente** Abkürzungen bzgl. P bilden $G_2 = (V, A_2)$
- kürzester Weg in Graph $G = (V, A_1 \cap A_2)$ liefert optimale zulässige und konsistente Vereinfachung



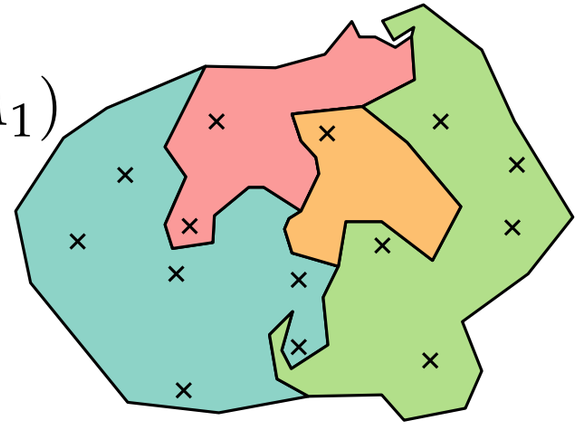
Kantenzug $C = (v_1, \ldots, v_n)$ und Vereinfachung C' heißen **konsistent bzgl.** P, wenn es Kantenzug B von v_n nach v_1 gibt, der C und C' zu einfachem Polygonen abschließt, das die gleiche Teilmenge von P einschließen.



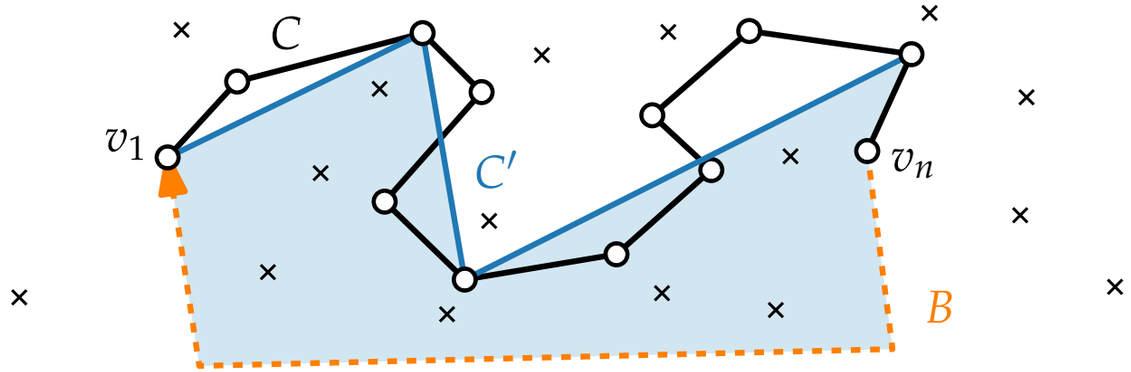
Algorithmus von de Berg et al. (1998)

Idee:

- graphbasierter Algorithmus (Imai & Iri) als Grundlage
- **zulässige** Abkürzungen bzgl. Fehler ε bilden $G_1 = (V, A_1)$
- **konsistente** Abkürzungen bzgl. P bilden $G_2 = (V, A_2)$
- kürzester Weg in Graph $G = (V, A_1 \cap A_2)$ liefert optimale zulässige und konsistente Vereinfachung



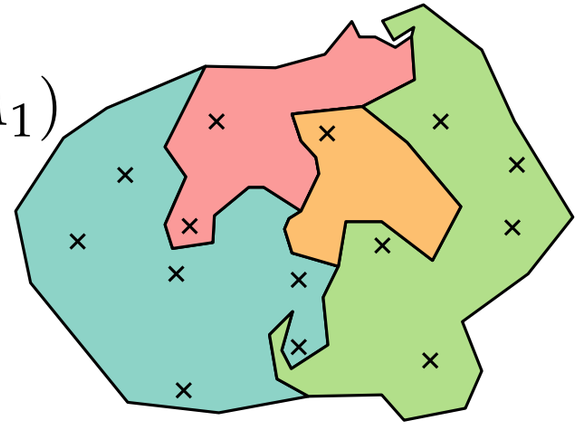
Kantenzug $C = (v_1, \ldots, v_n)$ und Vereinfachung C' heißen **konsistent bzgl.** P, wenn es Kantenzug B von v_n nach v_1 gibt, der C und C' zu einfachem Polygonen abschließt, das die gleiche Teilmenge von P einschließen.



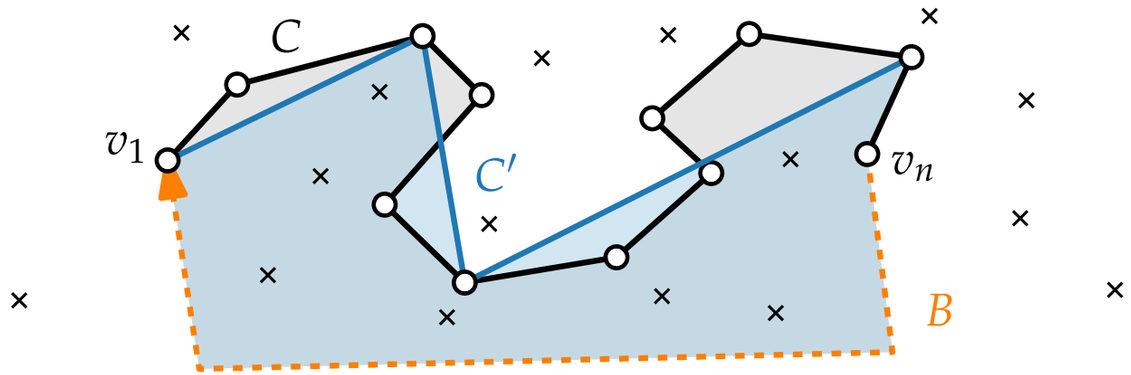
Algorithmus von de Berg et al. (1998)

Idee:

- graphbasierter Algorithmus (Imai & Iri) als Grundlage
- **zulässige** Abkürzungen bzgl. Fehler ε bilden $G_1 = (V, A_1)$
- **konsistente** Abkürzungen bzgl. P bilden $G_2 = (V, A_2)$
- kürzester Weg in Graph $G = (V, A_1 \cap A_2)$ liefert optimale zulässige und konsistente Vereinfachung



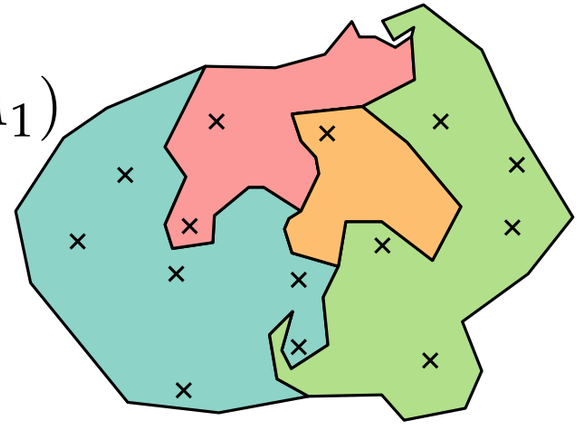
Kantenzug $C = (v_1, \ldots, v_n)$ und Vereinfachung C' heißen **konsistent bzgl.** P, wenn es Kantenzug B von v_n nach v_1 gibt, der C und C' zu einfachem Polygonen abschließt, das die gleiche Teilmenge von P einschließen.



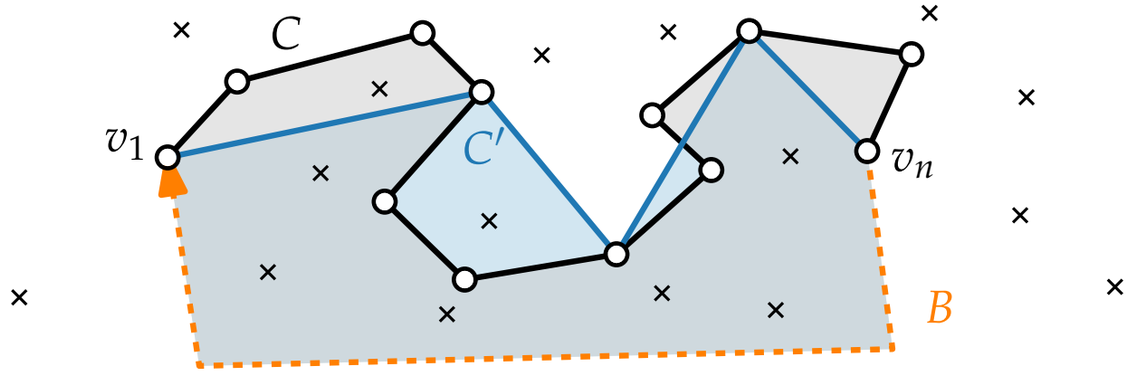
Algorithmus von de Berg et al. (1998)

Idee:

- graphbasierter Algorithmus (Imai & Iri) als Grundlage
- **zulässige** Abkürzungen bzgl. Fehler ε bilden $G_1 = (V, A_1)$
- **konsistente** Abkürzungen bzgl. P bilden $G_2 = (V, A_2)$
- kürzester Weg in Graph $G = (V, A_1 \cap A_2)$ liefert optimale zulässige und konsistente Vereinfachung



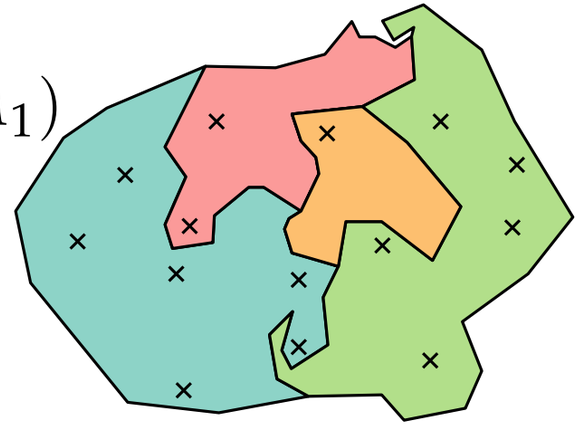
Kantenzug $C = (v_1, \ldots, v_n)$ und Vereinfachung C' heißen **konsistent bzgl.** P, wenn es Kantenzug B von v_n nach v_1 gibt, der C und C' zu einfachem Polygonen abschließt, das die gleiche Teilmenge von P einschließen.



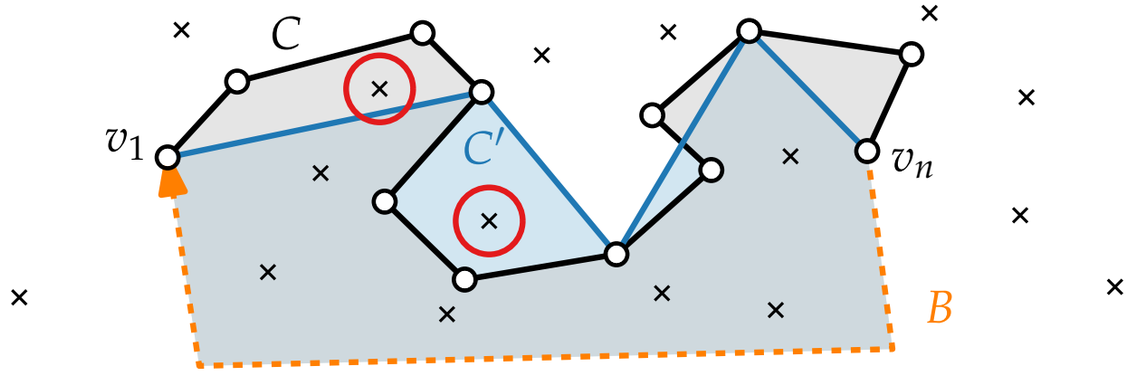
Algorithmus von de Berg et al. (1998)

Idee:

- graphbasierter Algorithmus (Imai & Iri) als Grundlage
- **zulässige** Abkürzungen bzgl. Fehler ε bilden $G_1 = (V, A_1)$
- **konsistente** Abkürzungen bzgl. P bilden $G_2 = (V, A_2)$
- kürzester Weg in Graph $G = (V, A_1 \cap A_2)$ liefert optimale zulässige und konsistente Vereinfachung



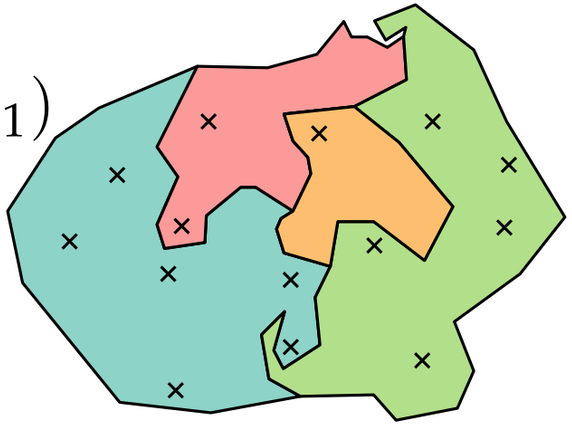
Kantenzug $C = (v_1, \ldots, v_n)$ und Vereinfachung C' heißen **konsistent bzgl.** P, wenn es Kantenzug B von v_n nach v_1 gibt, der C und C' zu einfachem Polygonen abschließt, das die gleiche Teilmenge von P einschließen.



Algorithmus von de Berg et al. (1998)

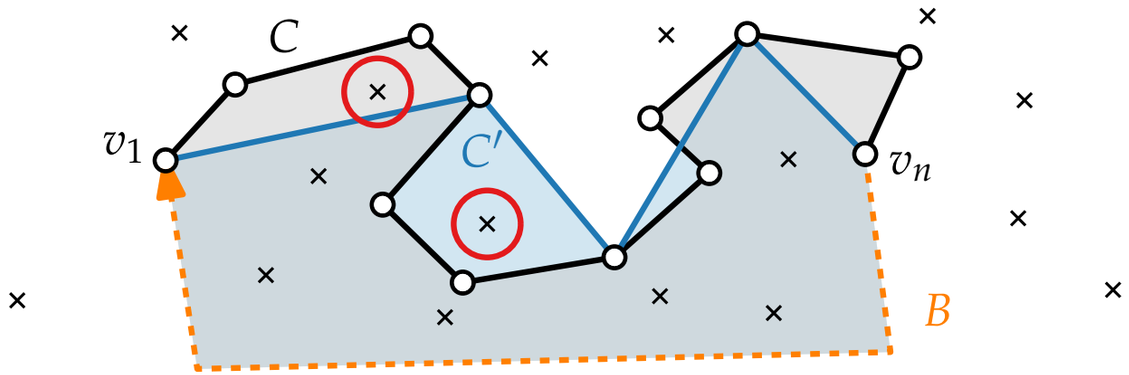
Idee:

- graphbasierter Algorithmus (Imai & Iri) als Grundlage
- **zulässige** Abkürzungen bzgl. Fehler ε bilden $G_1 = (V, A_1)$
- **konsistente** Abkürzungen bzgl. P bilden $G_2 = (V, A_2)$
- kürzester Weg in Graph $G = (V, A_1 \cap A_2)$ liefert optimale zulässige und konsistente Vereinfachung



Kantenzug $C = (v_1, \ldots, v_n)$ und Vereinfachung C' heißen **konsistent bzgl.** P, wenn es Kantenzug B von v_n nach v_1 gibt, der C und C' zu einfachem Polygonen abschließt, das die gleiche Teilmenge von P einschließen.

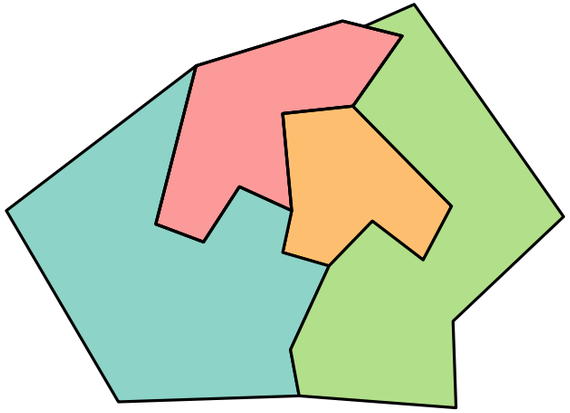
Erstmal nur mit Einschränkung:
x-monotone Kantenzüge



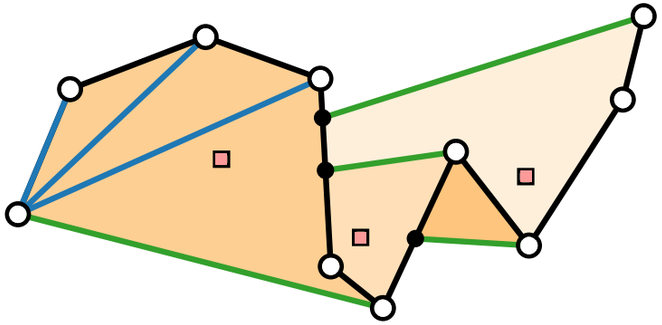
Universität Trier

Algorithmen für geographische Informationssysteme

10. Vorlesung
Flächenvereinfachung



Teil II:
x-monotone Liniensegmente

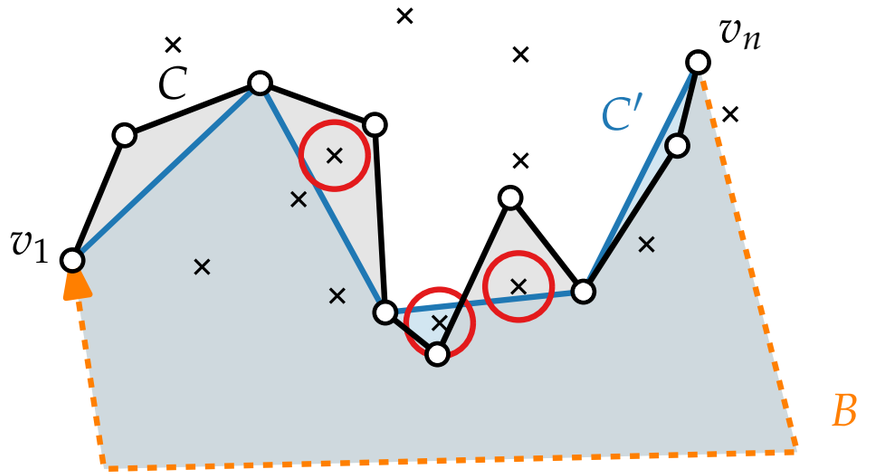


Philipp Kindermann

Wintersemester 2024/25

Konsistente Vereinfachung

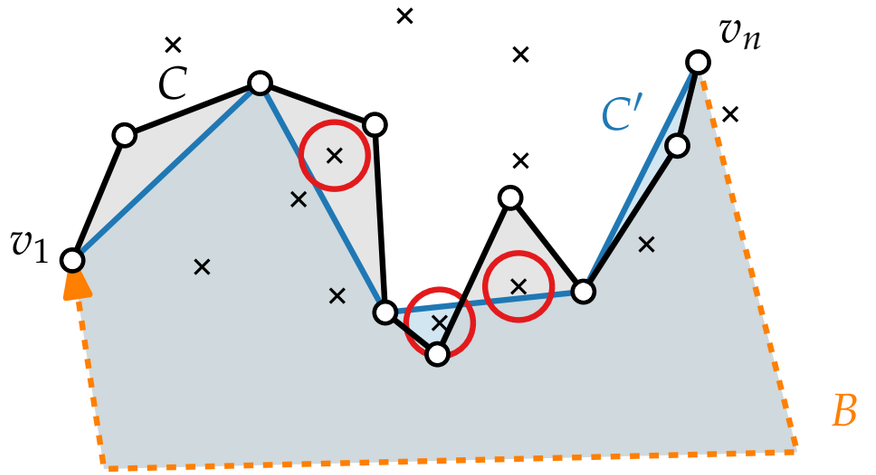
Lemma 1: C' konsistente Vereinfachung von C bzgl. P
$\Leftrightarrow$ kein Punkt von P liegt in Facette von CC'



Konsistente Vereinfachung

Lemma 1: C' konsistente Vereinfachung von C bzgl. P
$\Leftrightarrow$ kein Punkt von P liegt in Facette von CC'

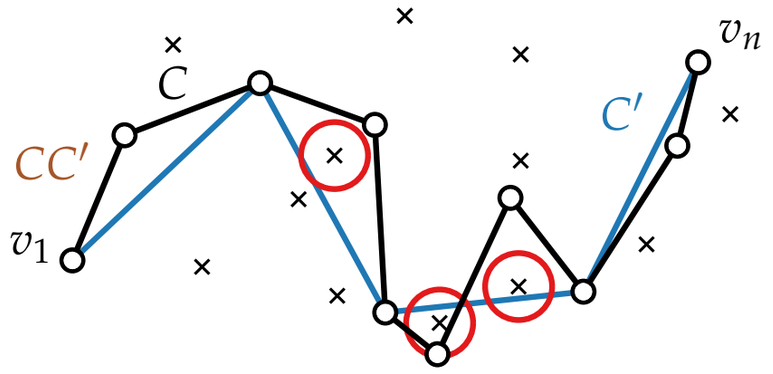
Fläche zwischen C und C'



Konsistente Vereinfachung

Lemma 1: C' konsistente Vereinfachung von C bzgl. P
$\Leftrightarrow$ kein Punkt von P liegt in Facette von CC'

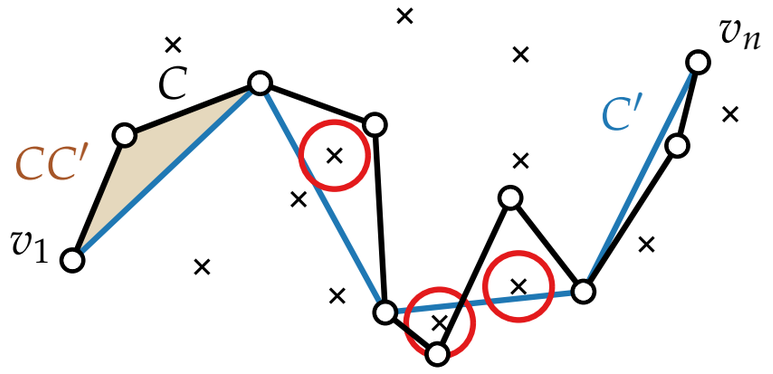
Fläche zwischen C und C'



Konsistente Vereinfachung

Lemma 1: C' konsistente Vereinfachung von C bzgl. P
$\Leftrightarrow$ kein Punkt von P liegt in Facette von CC'

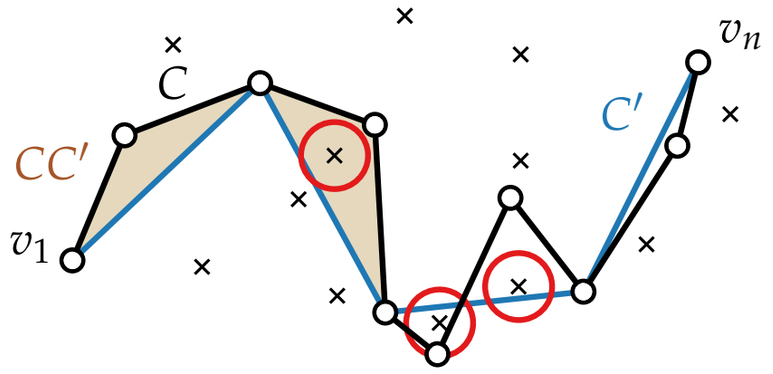
Fläche zwischen C und C'



Konsistente Vereinfachung

Lemma 1: C' konsistente Vereinfachung von C bzgl. P
$\Leftrightarrow$ kein Punkt von P liegt in Facette von CC'

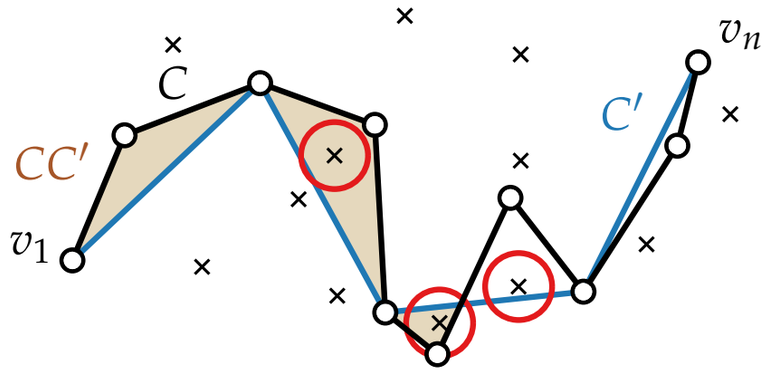
Fläche zwischen C und C'



Konsistente Vereinfachung

Lemma 1: C' konsistente Vereinfachung von C bzgl. P
$\Leftrightarrow$ kein Punkt von P liegt in Facette von CC'

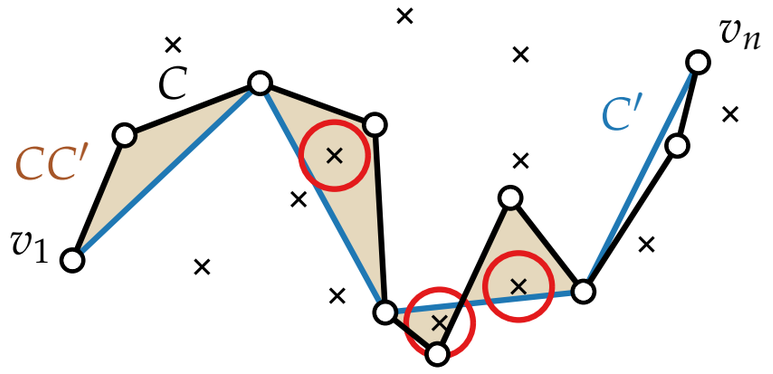
Fläche zwischen C und C'



Konsistente Vereinfachung

Lemma 1: C' konsistente Vereinfachung von C bzgl. P
$\Leftrightarrow$ kein Punkt von P liegt in Facette von CC'

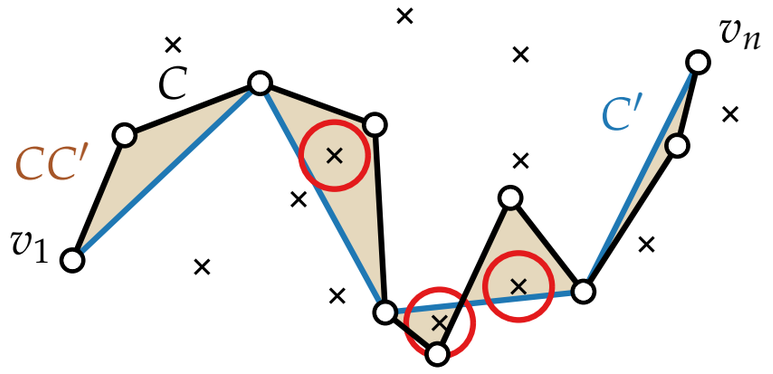
Fläche zwischen C und C'



Konsistente Vereinfachung

Lemma 1: C' konsistente Vereinfachung von C bzgl. P
$\Leftrightarrow$ kein Punkt von P liegt in Facette von CC'

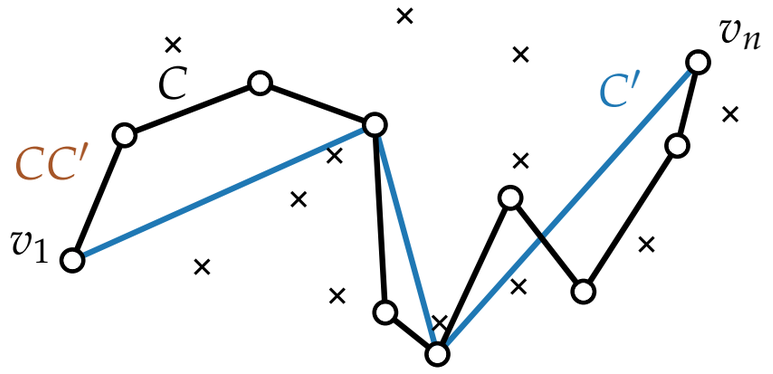
Fläche zwischen C und C'



Konsistente Vereinfachung

Lemma 1: C' konsistente Vereinfachung von C bzgl. P $\Leftrightarrow$ kein Punkt von P liegt in Facette von CC'

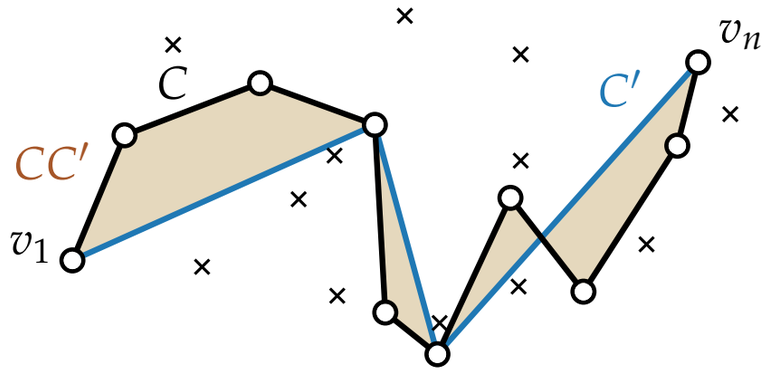
Fläche zwischen C und C'



Konsistente Vereinfachung

Lemma 1: C' konsistente Vereinfachung von C bzgl. P
$\Leftrightarrow$ kein Punkt von P liegt in Facette von CC'

Fläche zwischen C und C'



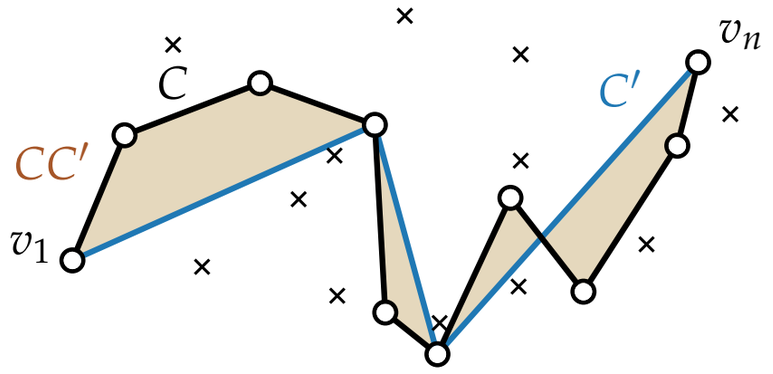
Konsistente Vereinfachung

Lemma 1: C' konsistente Vereinfachung von C bzgl. P $\Leftrightarrow$ kein Punkt von P liegt in Facette von CC'

Fläche zwischen C und C'

Punkt-in-Polygon-Kriterium (PIP)

Punkt $p \in$ Polygon $Q \Leftrightarrow$ (negativer vertikaler) Strahl schneidet Q ungerade oft



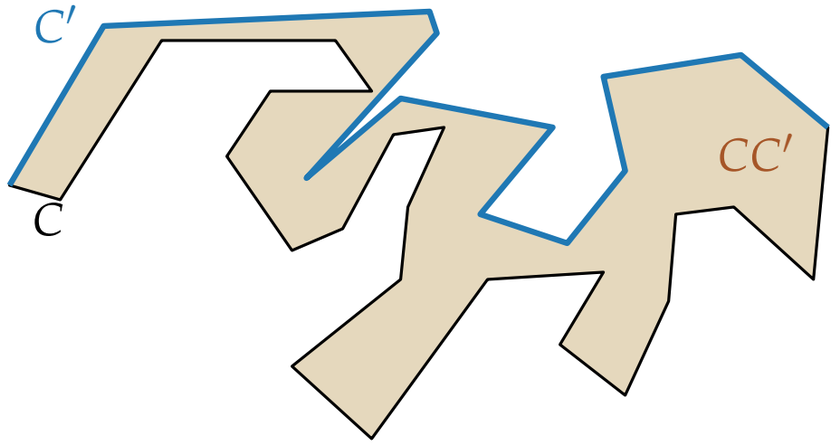
Konsistente Vereinfachung

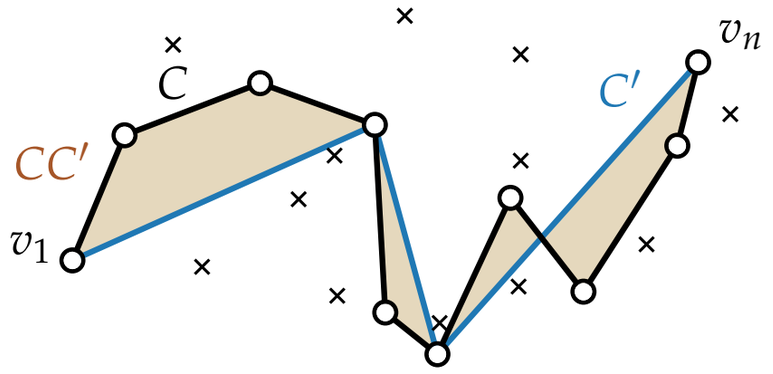
Lemma 1: C' konsistente Vereinfachung von C bzgl. P $\Leftrightarrow$ kein Punkt von P liegt in Facette von CC'

Fläche zwischen C und C'

Punkt-in-Polygon-Kriterium (PIP)

Punkt $p \in$ Polygon $Q \Leftrightarrow$ (negativer vertikaler) Strahl schneidet Q ungerade oft





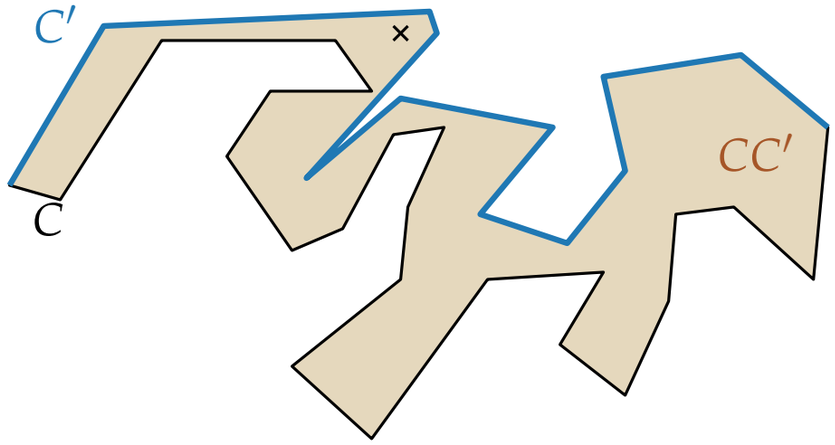
Konsistente Vereinfachung

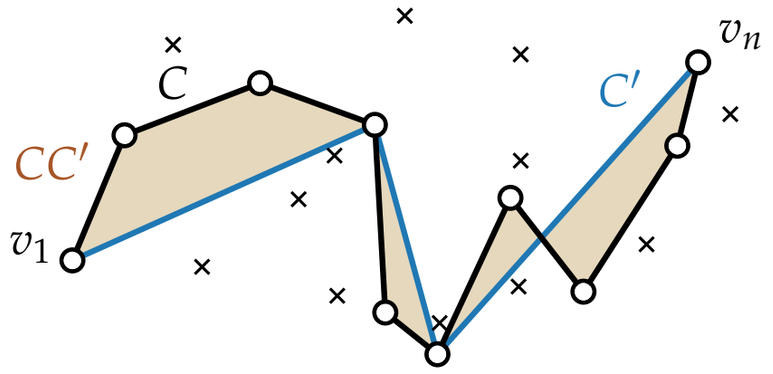
Lemma 1: C' konsistente Vereinfachung von C bzgl. P $\Leftrightarrow$ kein Punkt von P liegt in Facette von CC'

Fläche zwischen C und C'

Punkt-in-Polygon-Kriterium (PIP)

Punkt $p \in$ Polygon $Q \Leftrightarrow$ (negativer vertikaler) Strahl schneidet Q ungerade oft





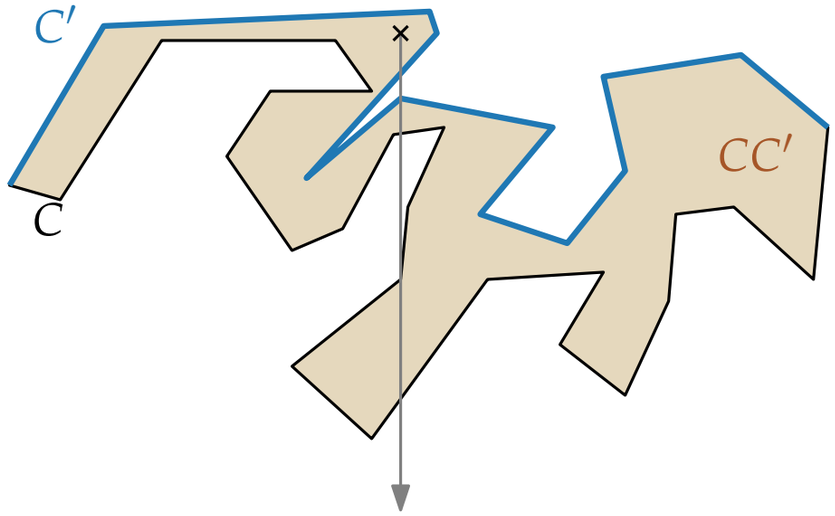
Konsistente Vereinfachung

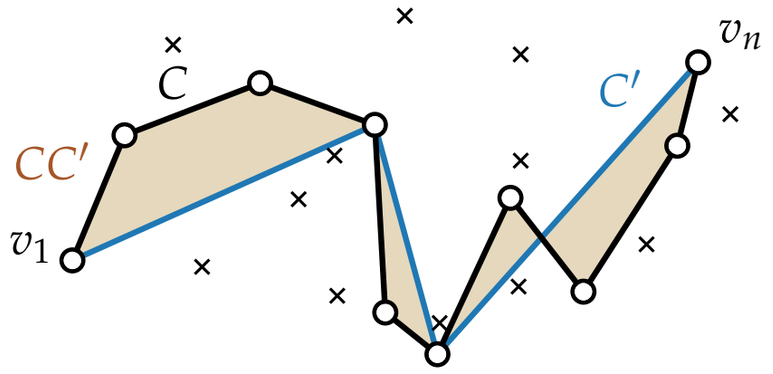
Lemma 1: C' konsistente Vereinfachung von C bzgl. P $\Leftrightarrow$ kein Punkt von P liegt in Facette von CC'

Fläche zwischen C und C'

Punkt-in-Polygon-Kriterium (PIP)

Punkt $p \in$ Polygon $Q \Leftrightarrow$ (negativer vertikaler) Strahl schneidet Q ungerade oft





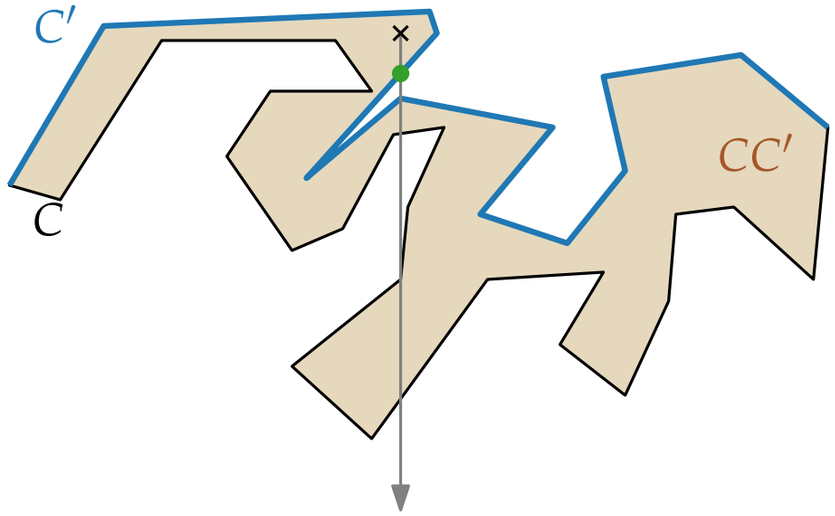
Konsistente Vereinfachung

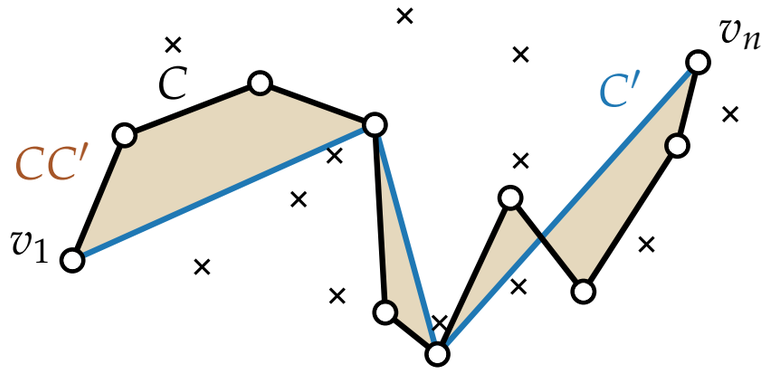
Lemma 1: C' konsistente Vereinfachung von C bzgl. P $\Leftrightarrow$ kein Punkt von P liegt in Facette von CC'

Fläche zwischen C und C'

Punkt-in-Polygon-Kriterium (PIP)

Punkt $p \in$ Polygon $Q \Leftrightarrow$ (negativer vertikaler) Strahl schneidet Q ungerade oft





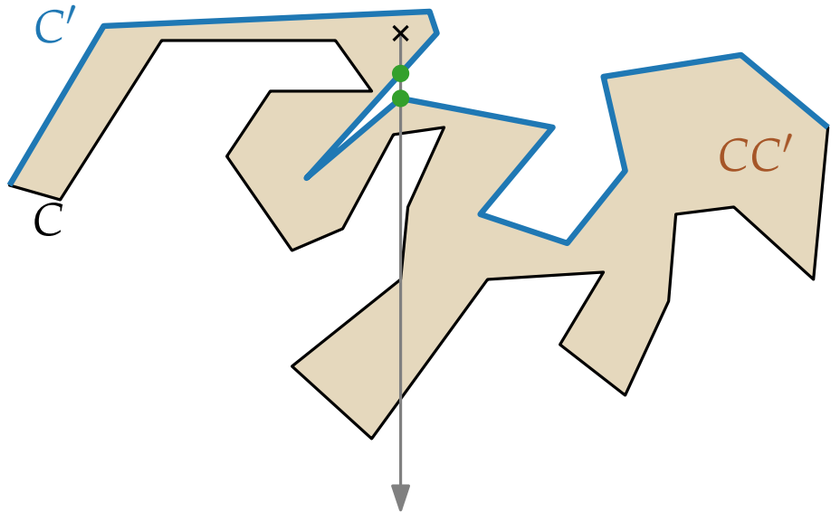
Konsistente Vereinfachung

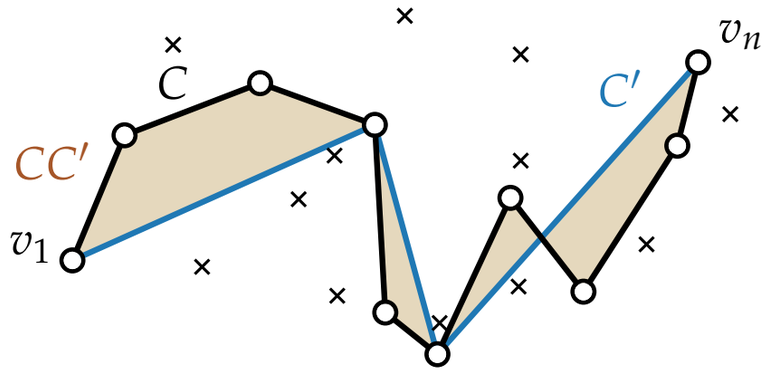
Lemma 1: C' konsistente Vereinfachung von C bzgl. P $\Leftrightarrow$ kein Punkt von P liegt in Facette von CC'

Fläche zwischen C und C'

Punkt-in-Polygon-Kriterium (PIP)

Punkt $p \in$ Polygon $Q \Leftrightarrow$ (negativer vertikaler) Strahl schneidet Q ungerade oft





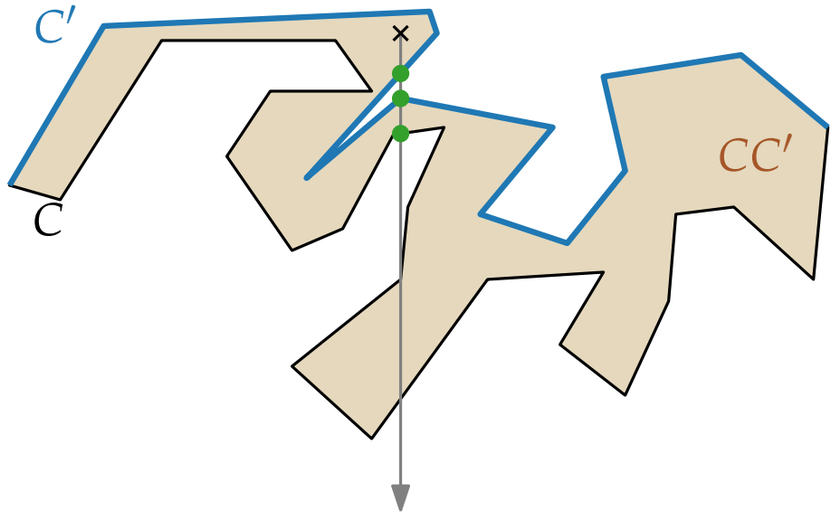
Konsistente Vereinfachung

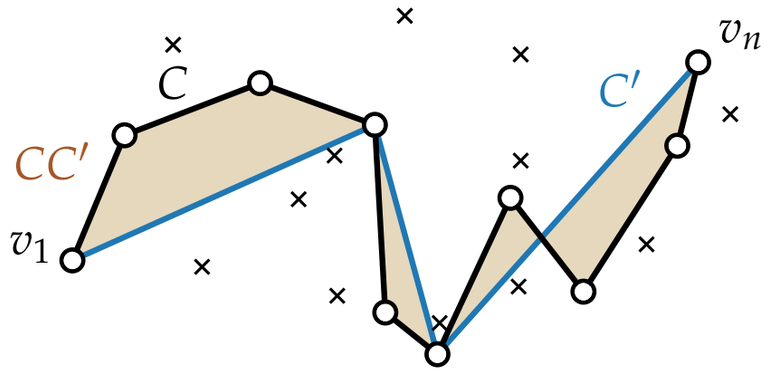
Lemma 1: C' konsistente Vereinfachung von C bzgl. P $\Leftrightarrow$ kein Punkt von P liegt in Facette von CC'

Fläche zwischen C und C'

Punkt-in-Polygon-Kriterium (PIP)

Punkt $p \in$ Polygon $Q \Leftrightarrow$ (negativer vertikaler) Strahl schneidet Q ungerade oft





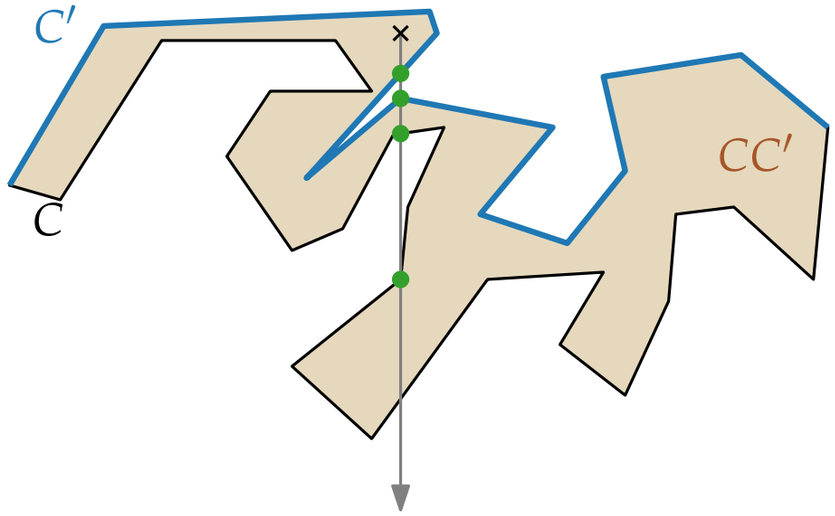
Konsistente Vereinfachung

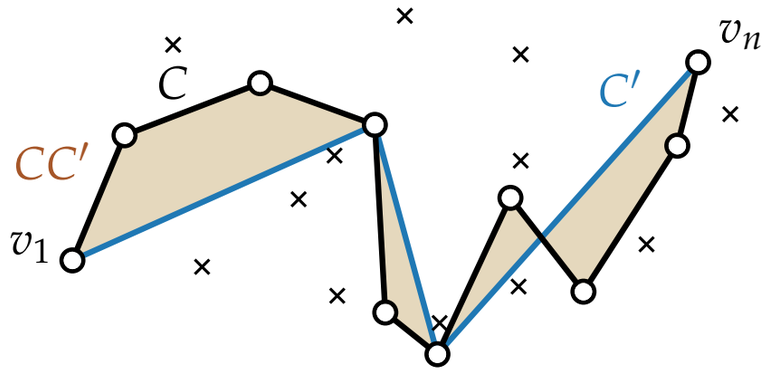
Lemma 1: C' konsistente Vereinfachung von C bzgl. P $\Leftrightarrow$ kein Punkt von P liegt in Facette von CC'

Fläche zwischen C und C'

Punkt-in-Polygon-Kriterium (PIP)

Punkt $p \in$ Polygon $Q \Leftrightarrow$ (negativer vertikaler) Strahl schneidet Q ungerade oft





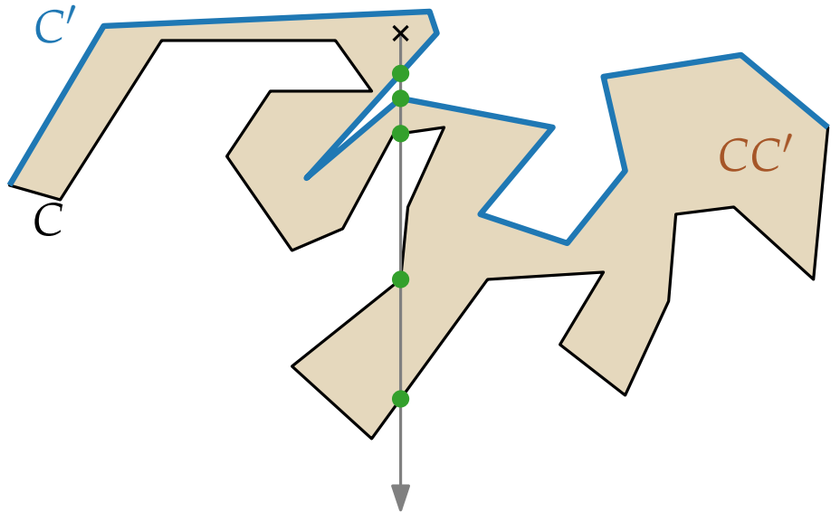
Konsistente Vereinfachung

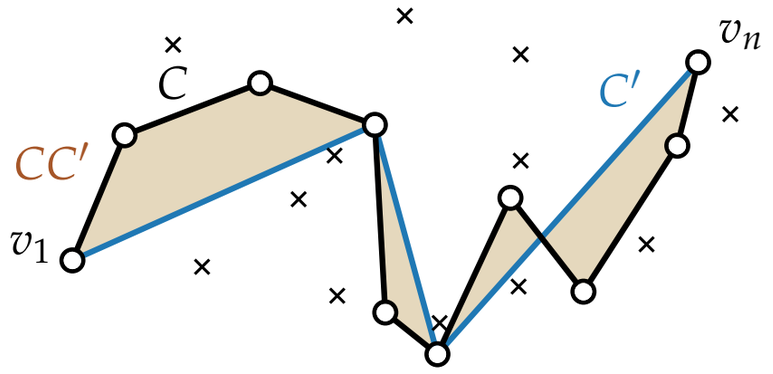
Lemma 1: C' konsistente Vereinfachung von C bzgl. P $\Leftrightarrow$ kein Punkt von P liegt in Facette von CC'

Fläche zwischen C und C'

Punkt-in-Polygon-Kriterium (PIP)

Punkt $p \in$ Polygon $Q \Leftrightarrow$ (negativer vertikaler) Strahl schneidet Q ungerade oft





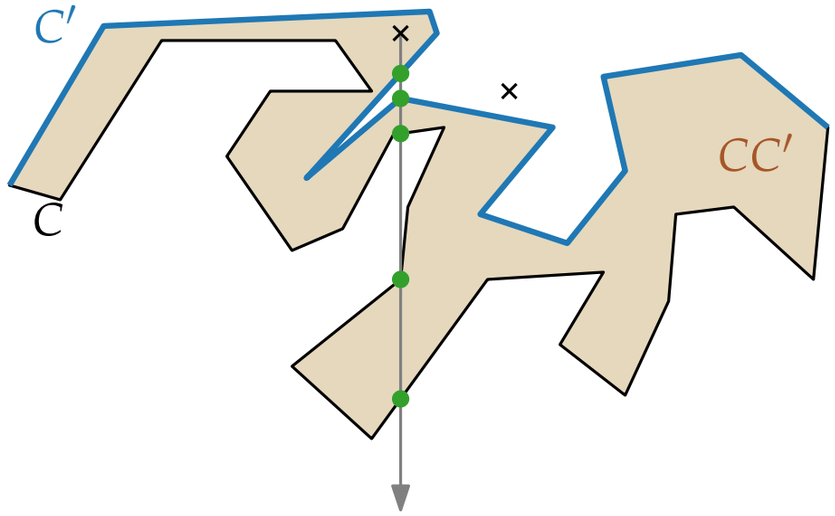
Konsistente Vereinfachung

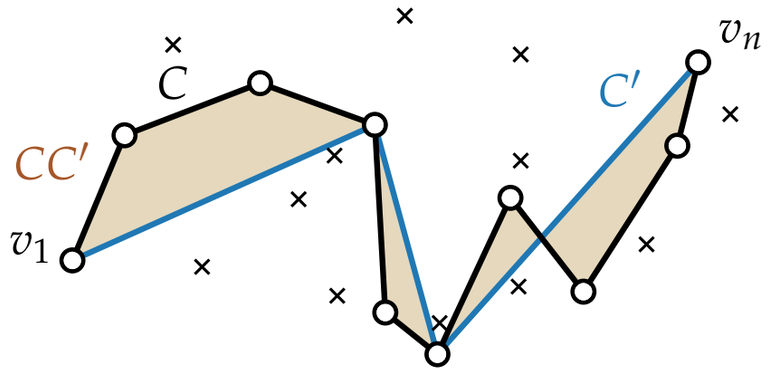
Lemma 1: C' konsistente Vereinfachung von C bzgl. P $\Leftrightarrow$ kein Punkt von P liegt in Facette von CC'

Fläche zwischen C und C'

Punkt-in-Polygon-Kriterium (PIP)

Punkt $p \in$ Polygon $Q \Leftrightarrow$ (negativer vertikaler) Strahl schneidet Q ungerade oft





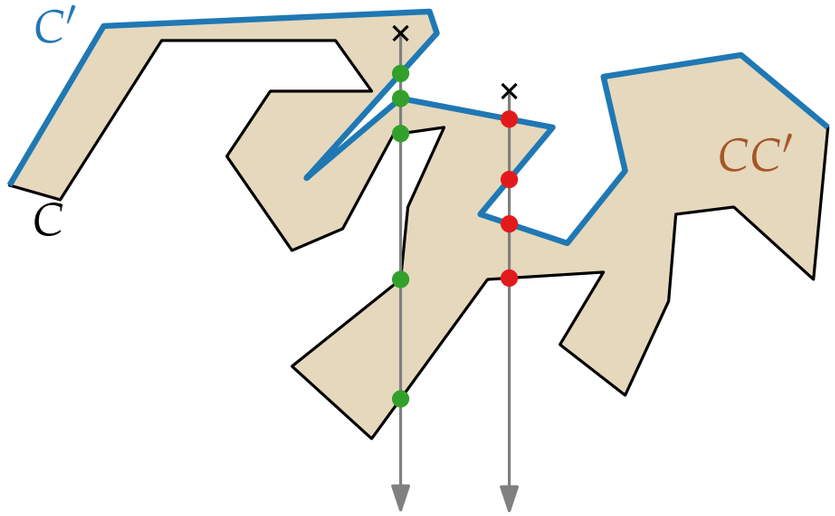
Konsistente Vereinfachung

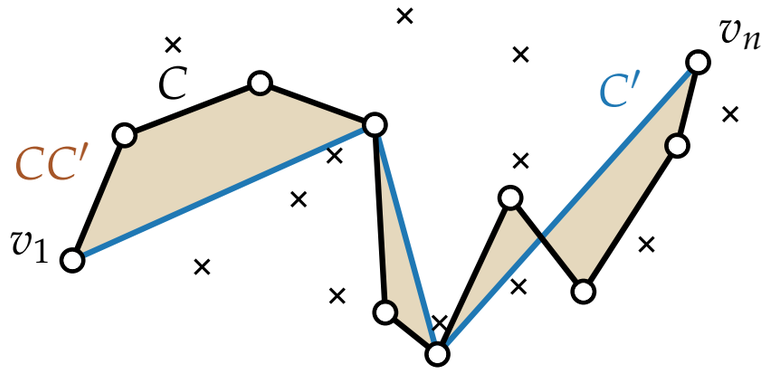
Lemma 1: C' konsistente Vereinfachung von C bzgl. P $\Leftrightarrow$ kein Punkt von P liegt in Facette von CC'

Fläche zwischen C und C'

Punkt-in-Polygon-Kriterium (PIP)

Punkt $p \in$ Polygon $Q \Leftrightarrow$ (negativer vertikaler) Strahl schneidet Q ungerade oft





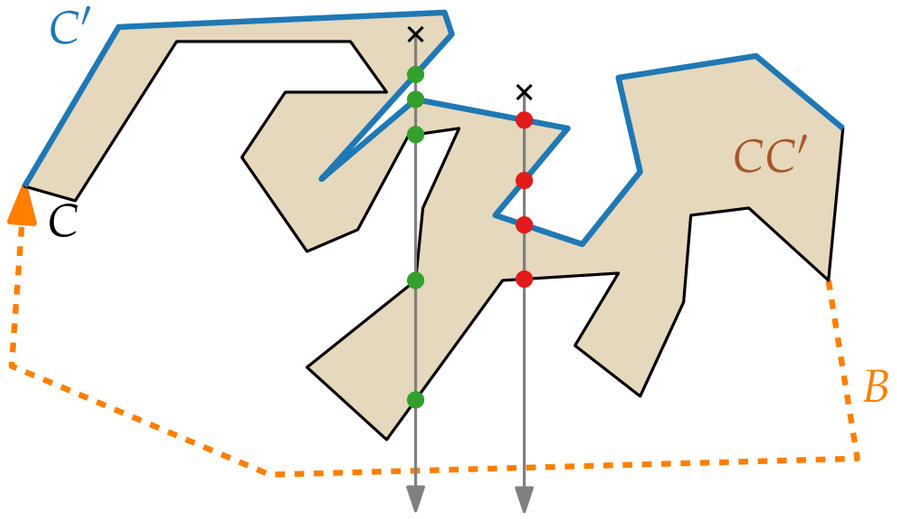
Konsistente Vereinfachung

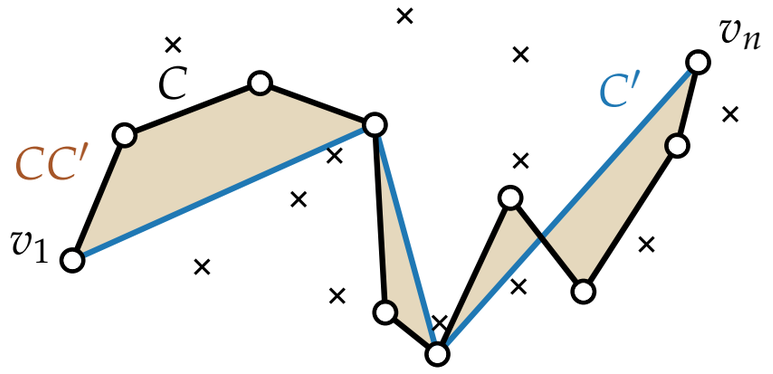
Lemma 1: C' konsistente Vereinfachung von C bzgl. P $\Leftrightarrow$ kein Punkt von P liegt in Facette von CC'

Fläche zwischen C und C'

Punkt-in-Polygon-Kriterium (PIP)

Punkt $p \in$ Polygon $Q \Leftrightarrow$ (negativer vertikaler) Strahl schneidet Q ungerade oft





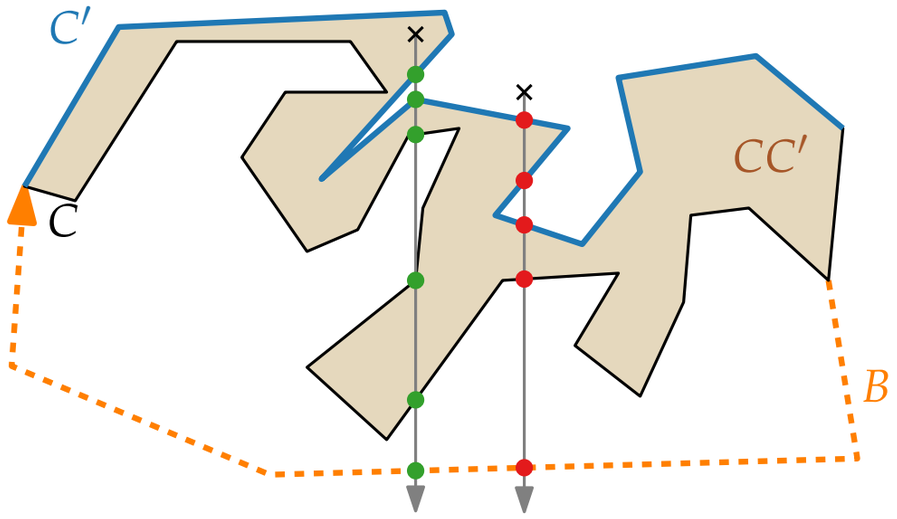
Konsistente Vereinfachung

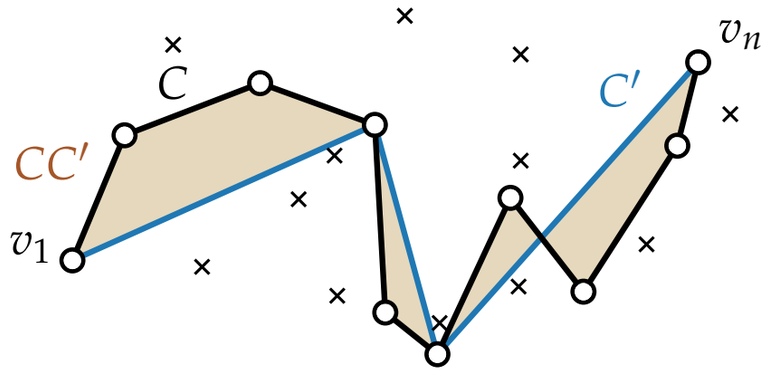
Lemma 1: C' konsistente Vereinfachung von C bzgl. P $\Leftrightarrow$ kein Punkt von P liegt in Facette von CC'

Fläche zwischen C und C'

Punkt-in-Polygon-Kriterium (PIP)

Punkt $p \in$ Polygon $Q \Leftrightarrow$ (negativer vertikaler) Strahl schneidet Q ungerade oft





Konsistente Vereinfachung

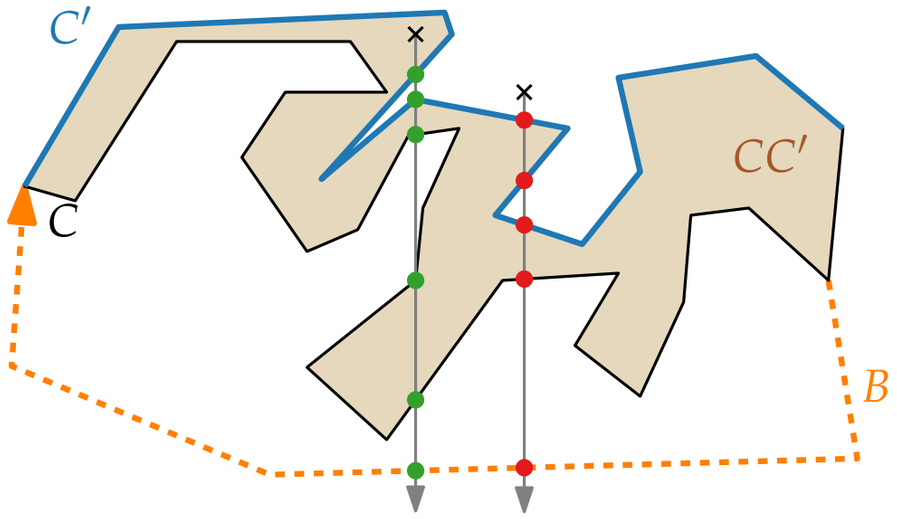
Lemma 1: C' konsistente Vereinfachung von C bzgl. P $\Leftrightarrow$ kein Punkt von P liegt in Facette von CC'

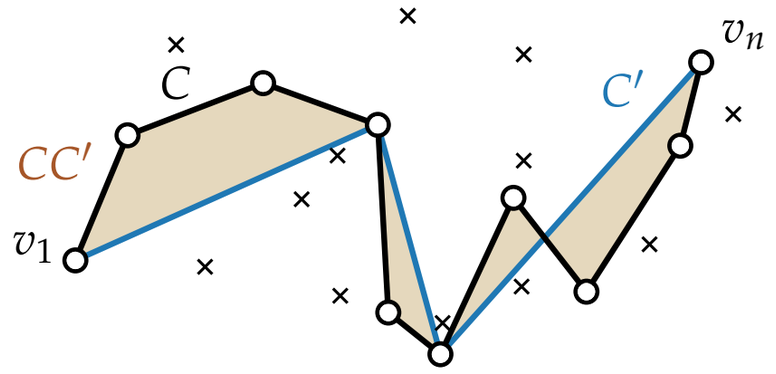
Fläche zwischen C und C'

Punkt-in-Polygon-Kriterium (PIP)

Punkt $p \in$ Polygon $Q \Leftrightarrow$ (negativer vertikaler) Strahl schneidet Q ungerade oft

$p \in CC'$ ➡ Strahl schneidet CC' ungerade oft





Konsistente Vereinfachung

Lemma 1: C' konsistente Vereinfachung von C bzgl. P $\Leftrightarrow$ kein Punkt von P liegt in Facette von CC'

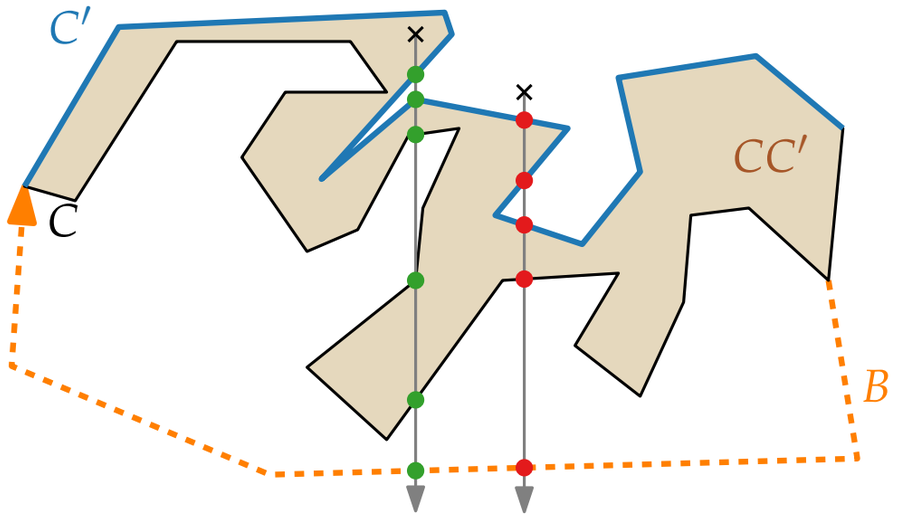
Fläche zwischen C und C'

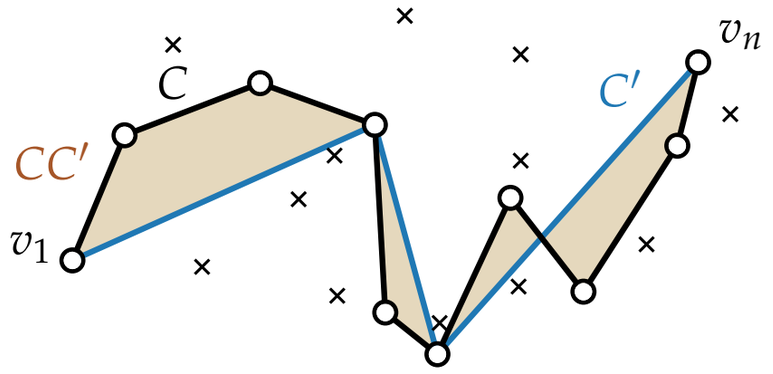
Punkt-in-Polygon-Kriterium (PIP)

Punkt $p \in$ Polygon $Q \Leftrightarrow$ (negativer vertikaler) Strahl schneidet Q ungerade oft

$p \in CC'$ ➡ Strahl schneidet CC' ungerade oft

➡ Strahl schneidet C ungerade und C' gerade oft (oder andersrum)





Konsistente Vereinfachung

Lemma 1: C' konsistente Vereinfachung von C bzgl. P $\Leftrightarrow$ kein Punkt von P liegt in Facette von CC'

Fläche zwischen C und C'

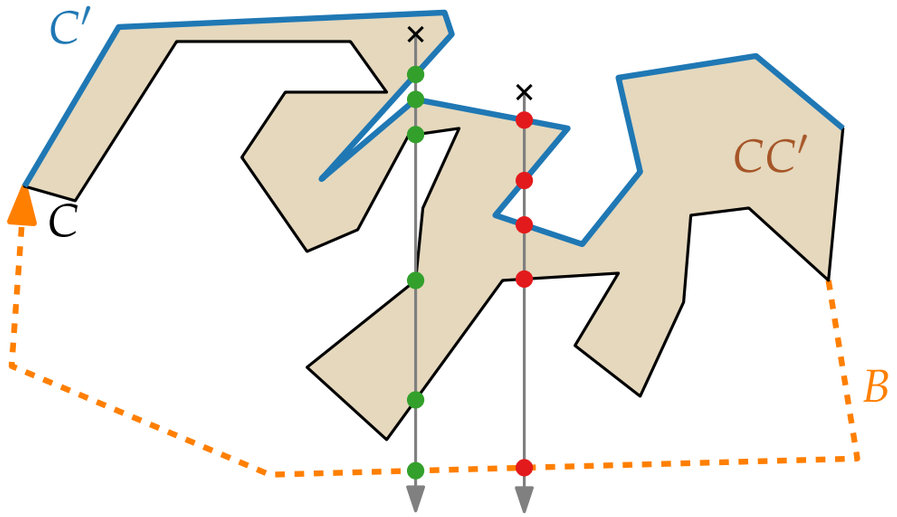
Punkt-in-Polygon-Kriterium (PIP)

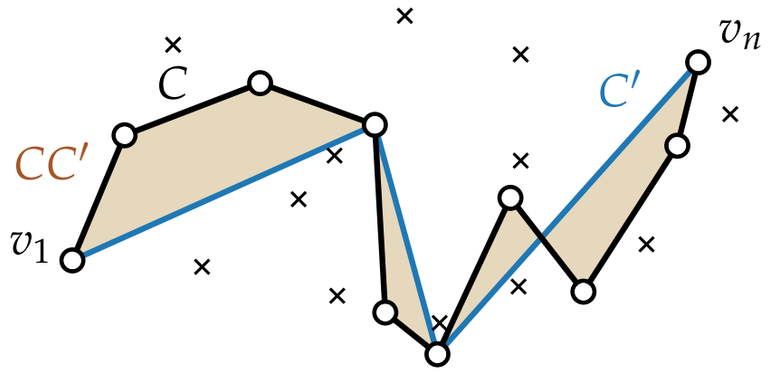
Punkt $p \in$ Polygon $Q \Leftrightarrow$ (negativer vertikaler) Strahl schneidet Q ungerade oft

$p \in CC'$ ➡ Strahl schneidet CC' ungerade oft

➡ Strahl schneidet C ungerade und C' gerade oft (oder andersrum)

➡ $p \in C$ und $p \notin C'$ (oder $p \notin C$ und $p \in C'$)





Konsistente Vereinfachung

Lemma 1: C' konsistente Vereinfachung von C bzgl. P $\Leftrightarrow$ kein Punkt von P liegt in Facette von CC'

Fläche zwischen C und C'

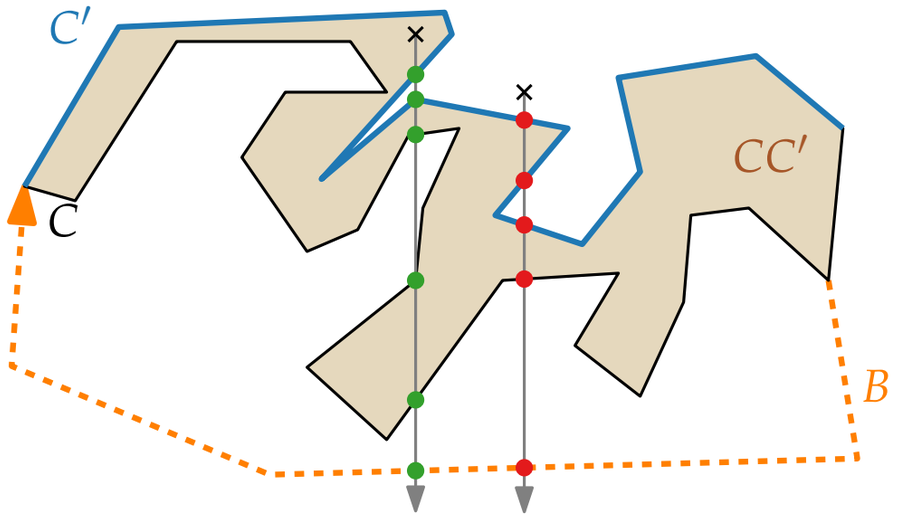
Punkt-in-Polygon-Kriterium (PIP)

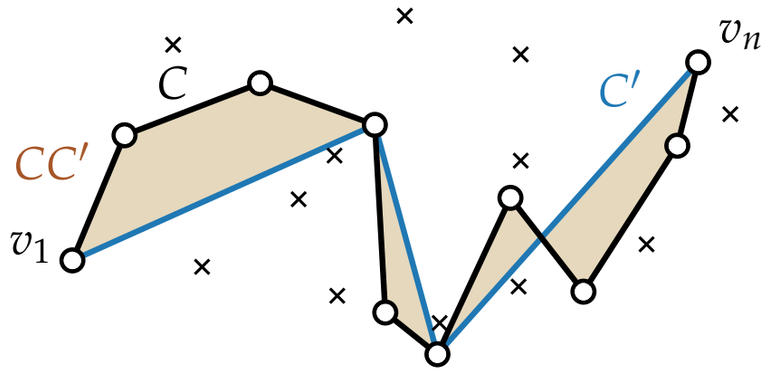
Punkt $p \in$ Polygon $Q \Leftrightarrow$ (negativer vertikaler) Strahl schneidet Q ungerade oft

$p \in CC'$ ➡ Strahl schneidet CC' ungerade oft

➡ Strahl schneidet C ungerade und C' gerade oft (oder andersrum)

➡ $p \in C$ und $p \notin C'$ (oder $p \notin C$ und $p \in C'$) □



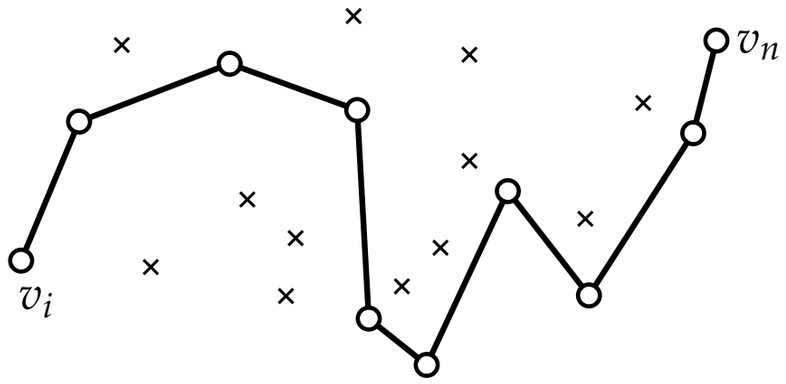


Vorgehen

Lemma 1: C' konsistente Vereinfachung von C bzgl. P $\Leftrightarrow$ kein Punkt von P liegt in Facette von CC'

Fläche zwischen C und C'

Bestimmung aller konsistenten Abkürzungen von v_i



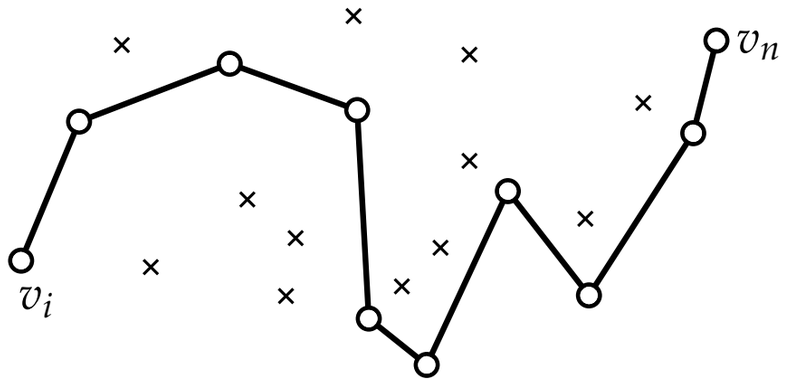
Vorgehen

Lemma 1: C' konsistente Vereinfachung von C bzgl. P $\Leftrightarrow$ kein Punkt von P liegt in Facette von CC'

Fläche zwischen C und C'

Bestimmung aller konsistenten Abkürzungen von v_i

- C_{ij}: Kantenzug von v_i bis v_j



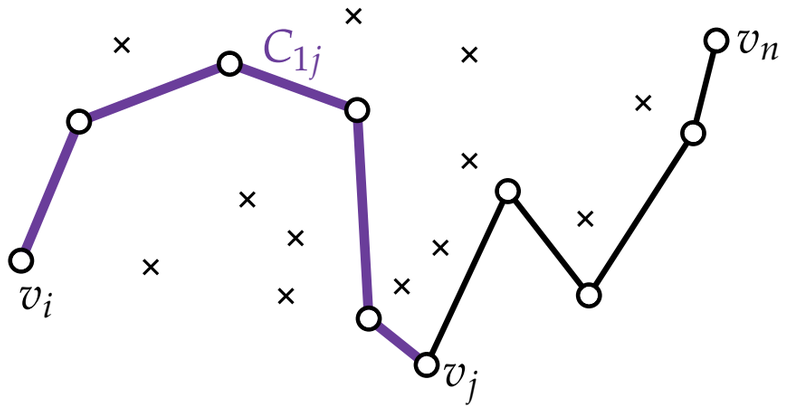
Vorgehen

Lemma 1: C' konsistente Vereinfachung von C bzgl. P
$\Leftrightarrow$ kein Punkt von P liegt in Facette von CC'

Fläche zwischen C und C'

Bestimmung aller konsistenten Abkürzungen von v_i

- C_{ij}: Kantenzug von v_i bis v_j



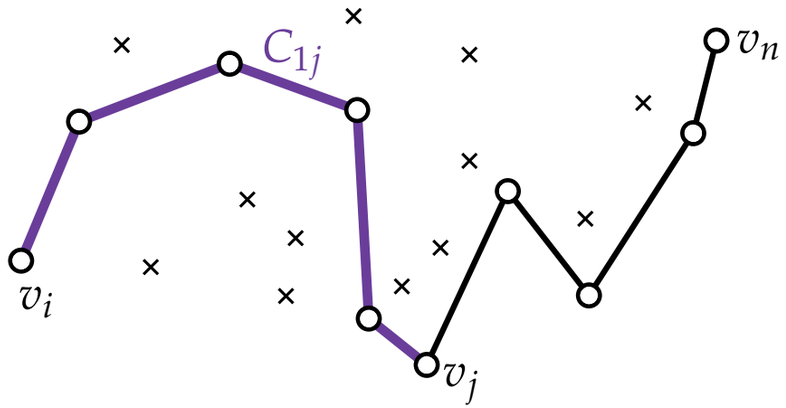
Vorgehen

Lemma 1: C' konsistente Vereinfachung von C bzgl. P $\Leftrightarrow$ kein Punkt von P liegt in Facette von CC'

Fläche zwischen C und C'

Bestimmung aller konsistenten Abkürzungen von v_i

- C_{ij}: Kantenzug von v_i bis v_j
- Q_{ij}: Polygon $C_{ij} \cup \overline{v_i v_j}$



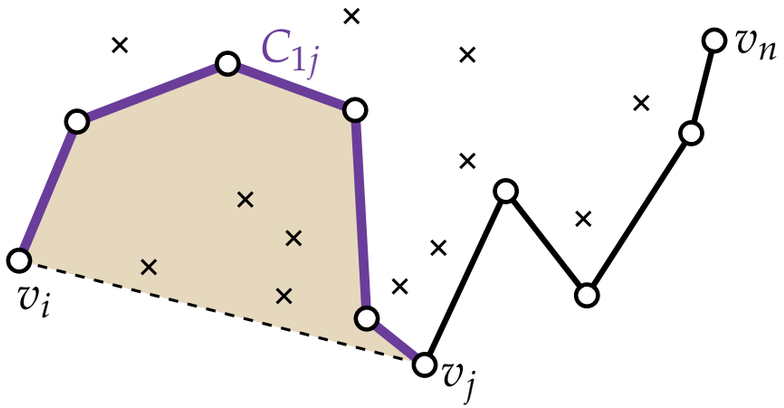
Vorgehen

Lemma 1: C' konsistente Vereinfachung von C bzgl. P $\Leftrightarrow$ kein Punkt von P liegt in Facette von CC'

Fläche zwischen C und C'

Bestimmung aller konsistenten Abkürzungen von v_i

- C_{ij}: Kantenzug von v_i bis v_j
- Q_{ij}: Polygon $C_{ij} \cup \overline{v_i v_j}$



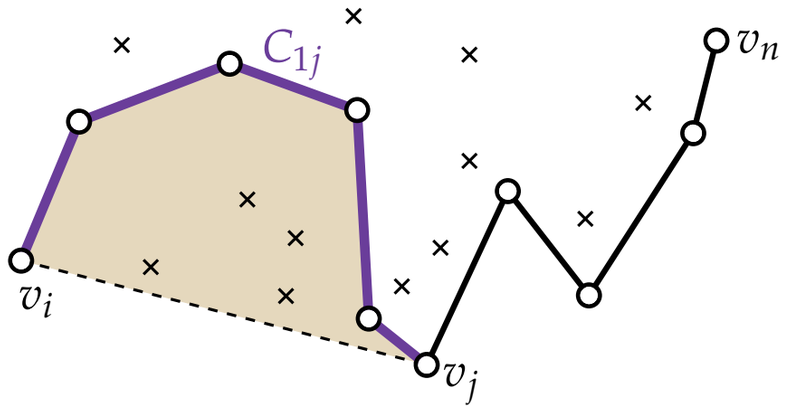
Vorgehen

Lemma 1: C' konsistente Vereinfachung von C bzgl. P $\Leftrightarrow$ kein Punkt von P liegt in Facette von CC'

Fläche zwischen C und C'

Bestimmung aller konsistenten Abkürzungen von v_i

- C_{ij}: Kantenzug von v_i bis v_j
- Q_{ij}: Polygon $C_{ij} \cup \overline{v_i v_j}$
- $(v_i, v_j) \in A \Leftrightarrow Q_{ij}$ enthält keine Punkte aus P



Vorgehen

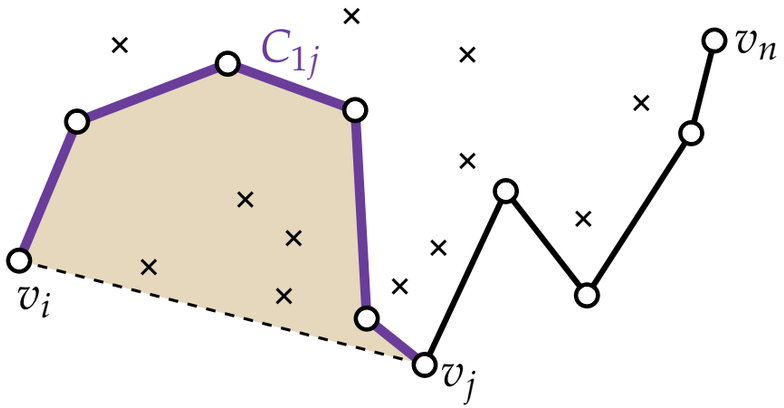
Lemma 1: C' konsistente Vereinfachung von C bzgl. P $\Leftrightarrow$ kein Punkt von P liegt in Facette von CC'

Fläche zwischen C und C'

Bestimmung aller konsistenten Abkürzungen von v_i

- C_{ij}: Kantenzug von v_i bis v_j
- Q_{ij}: Polygon $C_{ij} \cup \overline{v_i v_j}$
- $(v_i, v_j) \in A \Leftrightarrow Q_{ij}$ enthält keine Punkte aus P

(v_i, v_j) ist eine **Tangente**, wenn v_{j-1} und v_{j+1} auf gleicher Seite von $v_i v_j$ liegen.



Vorgehen

Lemma 1: C' konsistente Vereinfachung von C bzgl. P $\Leftrightarrow$ kein Punkt von P liegt in Facette von CC'

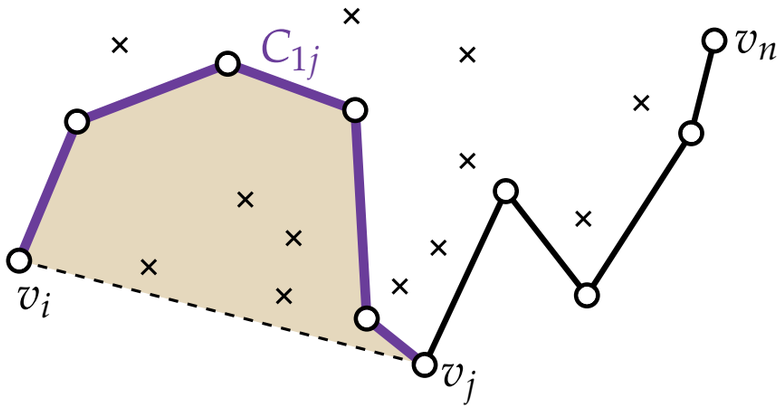
Fläche zwischen C und C'

Bestimmung aller konsistenten Abkürzungen von v_i

- C_{ij}: Kantenzug von v_i bis v_j
- Q_{ij}: Polygon $C_{ij} \cup \overline{v_i v_j}$
- $(v_i, v_j) \in A \Leftrightarrow Q_{ij}$ enthält keine Punkte aus P

(v_i, v_j) ist eine **Tangente**, wenn v_{j-1} und v_{j+1} auf gleicher Seite von $v_i v_j$ liegen.

Vorgehen für festes v_i in 3 Schritten



Vorgehen

Lemma 1: C' konsistente Vereinfachung von C bzgl. P $\Leftrightarrow$ kein Punkt von P liegt in Facette von CC'

Fläche zwischen C und C'

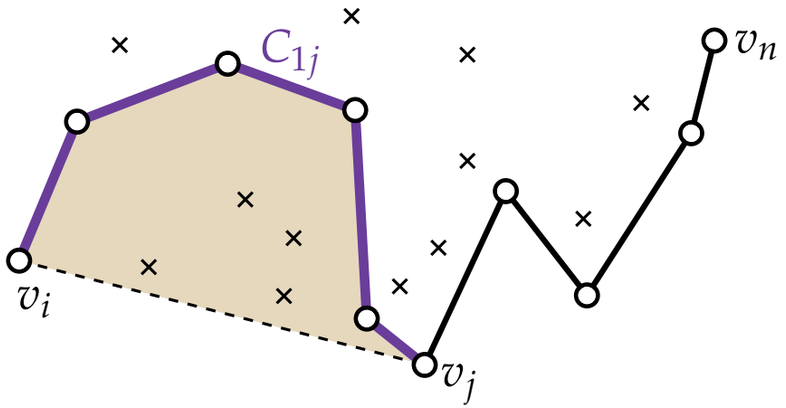
Bestimmung aller konsistenten Abkürzungen von v_i

- C_{ij}: Kantenzug von v_i bis v_j
- Q_{ij}: Polygon $C_{ij} \cup \overline{v_i v_j}$
- $(v_i, v_j) \in A \Leftrightarrow Q_{ij}$ enthält keine Punkte aus P

(v_i, v_j) ist eine **Tangente**, wenn v_{j-1} und v_{j+1} auf gleicher Seite von $v_i v_j$ liegen.

Vorgehen für festes v_i in 3 Schritten

1. berechne Tangentensegmente und induzierte Unterteilung S_i



Vorgehen

Lemma 1: C' konsistente Vereinfachung von C bzgl. P $\Leftrightarrow$ kein Punkt von P liegt in Facette von CC'

Fläche zwischen C und C'

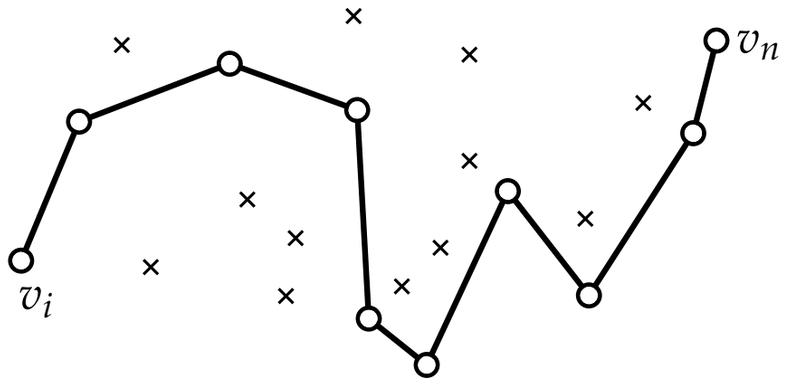
Bestimmung aller konsistenten Abkürzungen von v_i

- C_{ij}: Kantenzug von v_i bis v_j
- Q_{ij}: Polygon $C_{ij} \cup \overline{v_i v_j}$
- $(v_i, v_j) \in A \Leftrightarrow Q_{ij}$ enthält keine Punkte aus P

(v_i, v_j) ist eine **Tangente**, wenn v_{j-1} und v_{j+1} auf gleicher Seite von $v_i v_j$ liegen.

Vorgehen für festes v_i in 3 Schritten

1. berechne Tangentensegmente und induzierte Unterteilung S_i



Vorgehen

Lemma 1: C' konsistente Vereinfachung von C bzgl. P $\Leftrightarrow$ kein Punkt von P liegt in Facette von CC'

Fläche zwischen C und C'

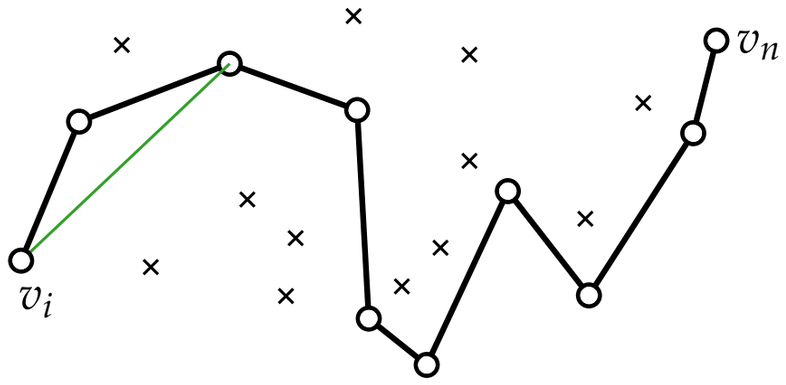
Bestimmung aller konsistenten Abkürzungen von v_i

- C_{ij}: Kantenzug von v_i bis v_j
- Q_{ij}: Polygon $C_{ij} \cup \overline{v_i v_j}$
- $(v_i, v_j) \in A \Leftrightarrow Q_{ij}$ enthält keine Punkte aus P

(v_i, v_j) ist eine **Tangente**, wenn v_{j-1} und v_{j+1} auf gleicher Seite von $v_i v_j$ liegen.

Vorgehen für festes v_i in 3 Schritten

1. berechne Tangentensegmente und induzierte Unterteilung S_i



Vorgehen

Lemma 1: C' konsistente Vereinfachung von C bzgl. P $\Leftrightarrow$ kein Punkt von P liegt in Facette von CC'

Fläche zwischen C und C'

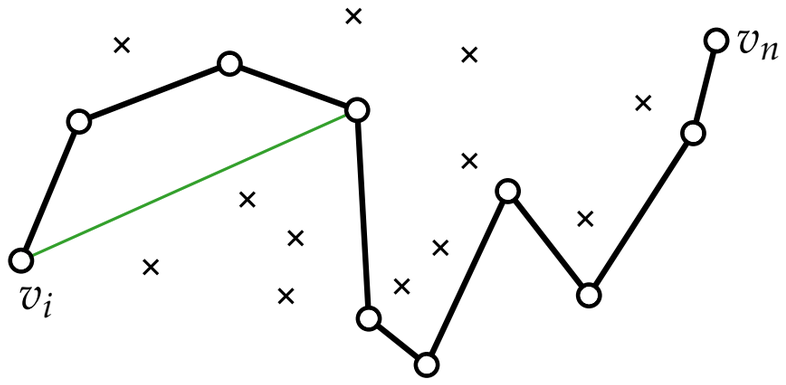
Bestimmung aller konsistenten Abkürzungen von v_i

- C_{ij}: Kantenzug von v_i bis v_j
- Q_{ij}: Polygon $C_{ij} \cup \overline{v_i v_j}$
- $(v_i, v_j) \in A \Leftrightarrow Q_{ij}$ enthält keine Punkte aus P

(v_i, v_j) ist eine **Tangente**, wenn v_{j-1} und v_{j+1} auf gleicher Seite von $v_i v_j$ liegen.

Vorgehen für festes v_i in 3 Schritten

1. berechne Tangentensegmente und induzierte Unterteilung S_i



Vorgehen

Lemma 1: C' konsistente Vereinfachung von C bzgl. P $\Leftrightarrow$ kein Punkt von P liegt in Facette von CC'

Fläche zwischen C und C'

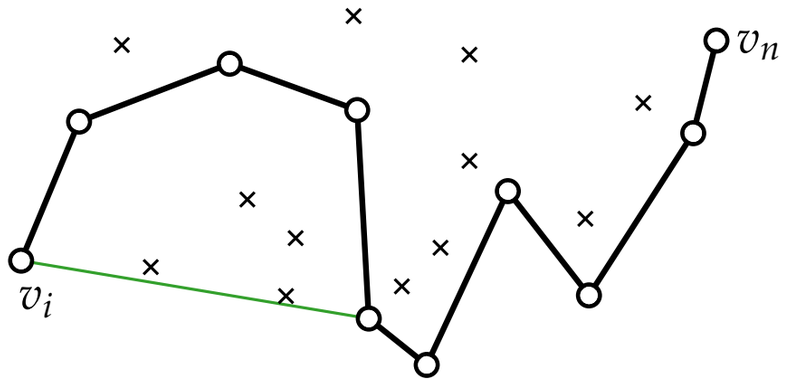
Bestimmung aller konsistenten Abkürzungen von v_i

- C_{ij}: Kantenzug von v_i bis v_j
- Q_{ij}: Polygon $C_{ij} \cup \overline{v_i v_j}$
- $(v_i, v_j) \in A \Leftrightarrow Q_{ij}$ enthält keine Punkte aus P

(v_i, v_j) ist eine **Tangente**, wenn v_{j-1} und v_{j+1} auf gleicher Seite von $v_i v_j$ liegen.

Vorgehen für festes v_i in 3 Schritten

1. berechne Tangentensegmente und induzierte Unterteilung S_i



Vorgehen

Lemma 1: C' konsistente Vereinfachung von C bzgl. P $\Leftrightarrow$ kein Punkt von P liegt in Facette von CC'

Fläche zwischen C und C'

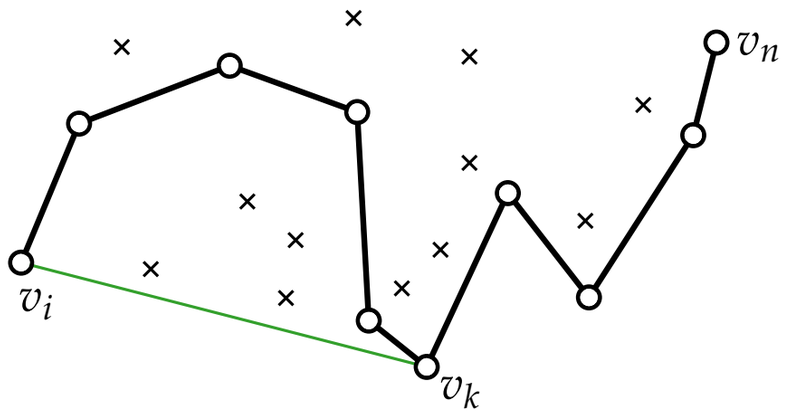
Bestimmung aller konsistenten Abkürzungen von v_i

- C_{ij}: Kantenzug von v_i bis v_j
- Q_{ij}: Polygon $C_{ij} \cup \overline{v_i v_j}$
- $(v_i, v_j) \in A \Leftrightarrow Q_{ij}$ enthält keine Punkte aus P

(v_i, v_j) ist eine **Tangente**, wenn v_{j-1} und v_{j+1} auf gleicher Seite von $v_i v_j$ liegen.

Vorgehen für festes v_i in 3 Schritten

1. berechne Tangentensegmente und induzierte Unterteilung S_i



Vorgehen

Lemma 1: C' konsistente Vereinfachung von C bzgl. P $\Leftrightarrow$ kein Punkt von P liegt in Facette von CC'

Fläche zwischen C und C'

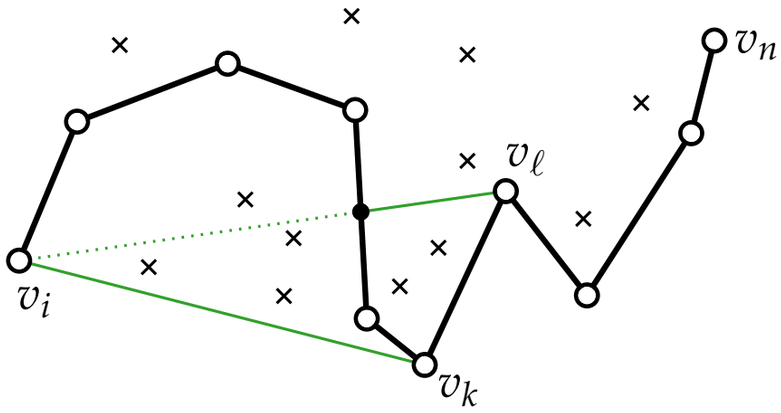
Bestimmung aller konsistenten Abkürzungen von v_i

- C_{ij}: Kantenzug von v_i bis v_j
- Q_{ij}: Polygon $C_{ij} \cup \overline{v_i v_j}$
- $(v_i, v_j) \in A \Leftrightarrow Q_{ij}$ enthält keine Punkte aus P

(v_i, v_j) ist eine **Tangente**, wenn v_{j-1} und v_{j+1} auf gleicher Seite von $v_i v_j$ liegen.

Vorgehen für festes v_i in 3 Schritten

1. berechne Tangentensegmente und induzierte Unterteilung S_i



Vorgehen

Lemma 1: C' konsistente Vereinfachung von C bzgl. P $\Leftrightarrow$ kein Punkt von P liegt in Facette von CC'

Fläche zwischen C und C'

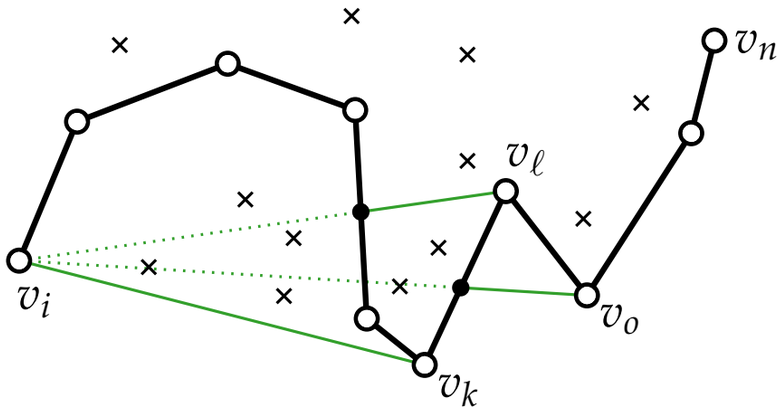
Bestimmung aller konsistenten Abkürzungen von v_i

- C_{ij}: Kantenzug von v_i bis v_j
- Q_{ij}: Polygon $C_{ij} \cup \overline{v_i v_j}$
- $(v_i, v_j) \in A \Leftrightarrow Q_{ij}$ enthält keine Punkte aus P

(v_i, v_j) ist eine **Tangente**, wenn v_{j-1} und v_{j+1} auf gleicher Seite von $v_i v_j$ liegen.

Vorgehen für festes v_i in 3 Schritten

1. berechne Tangentensegmente und induzierte Unterteilung S_i



Vorgehen

Lemma 1: C' konsistente Vereinfachung von C bzgl. P $\Leftrightarrow$ kein Punkt von P liegt in Facette von CC'

Fläche zwischen C und C'

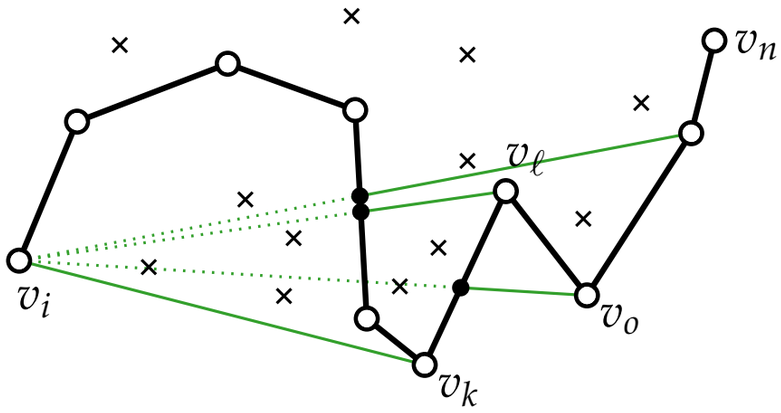
Bestimmung aller konsistenten Abkürzungen von v_i

- C_{ij}: Kantenzug von v_i bis v_j
- Q_{ij}: Polygon $C_{ij} \cup \overline{v_i v_j}$
- $(v_i, v_j) \in A \Leftrightarrow Q_{ij}$ enthält keine Punkte aus P

(v_i, v_j) ist eine **Tangente**, wenn v_{j-1} und v_{j+1} auf gleicher Seite von $v_i v_j$ liegen.

Vorgehen für festes v_i in 3 Schritten

1. berechne Tangentensegmente und induzierte Unterteilung S_i



Vorgehen

Lemma 1: C' konsistente Vereinfachung von C bzgl. P $\Leftrightarrow$ kein Punkt von P liegt in Facette von CC'

Fläche zwischen C und C'

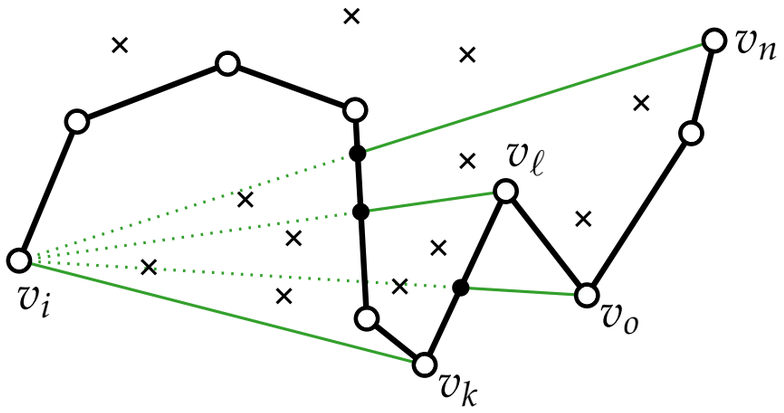
Bestimmung aller konsistenten Abkürzungen von v_i

- C_{ij}: Kantenzug von v_i bis v_j
- Q_{ij}: Polygon $C_{ij} \cup \overline{v_i v_j}$
- $(v_i, v_j) \in A \Leftrightarrow Q_{ij}$ enthält keine Punkte aus P

(v_i, v_j) ist eine **Tangente**, wenn v_{j-1} und v_{j+1} auf gleicher Seite von $v_i v_j$ liegen.

Vorgehen für festes v_i in 3 Schritten

1. berechne Tangentensegmente und induzierte Unterteilung S_i



Vorgehen

Lemma 1: C' konsistente Vereinfachung von C bzgl. P $\Leftrightarrow$ kein Punkt von P liegt in Facette von CC'

Fläche zwischen C und C'

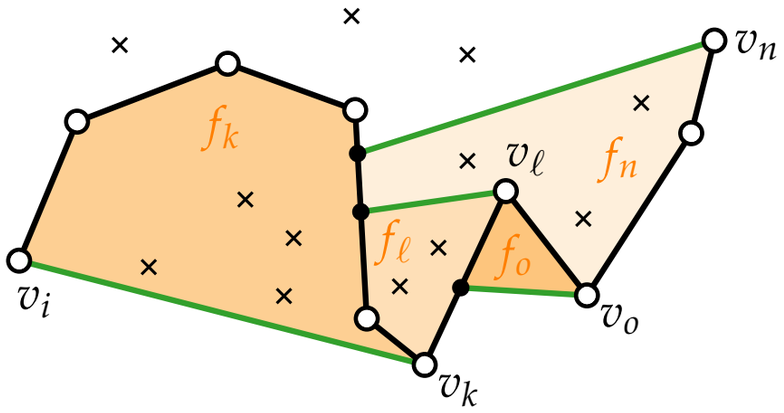
Bestimmung aller konsistenten Abkürzungen von v_i

- C_{ij}: Kantenzug von v_i bis v_j
- Q_{ij}: Polygon $C_{ij} \cup \overline{v_i v_j}$
- $(v_i, v_j) \in A \Leftrightarrow Q_{ij}$ enthält keine Punkte aus P

(v_i, v_j) ist eine **Tangente**, wenn v_{j-1} und v_{j+1} auf gleicher Seite von $v_i v_j$ liegen.

Vorgehen für festes v_i in 3 Schritten

1. berechne Tangentensegmente und induzierte Unterteilung S_i



Vorgehen

Lemma 1: C' konsistente Vereinfachung von C bzgl. P $\Leftrightarrow$ kein Punkt von P liegt in Facette von CC'

Fläche zwischen C und C'

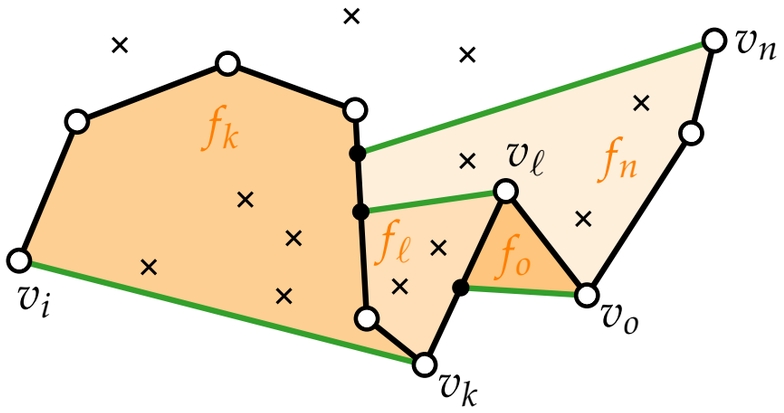
Bestimmung aller konsistenten Abkürzungen von v_i

- C_{ij}: Kantenzug von v_i bis v_j
- Q_{ij}: Polygon $C_{ij} \cup \overline{v_i v_j}$
- $(v_i, v_j) \in A \Leftrightarrow Q_{ij}$ enthält keine Punkte aus P

(v_i, v_j) ist eine **Tangente**, wenn v_{j-1} und v_{j+1} auf gleicher Seite von $v_i v_j$ liegen.

Vorgehen für festes v_i in 3 Schritten

1. berechne Tangentensegmente und induzierte Unterteilung S_i
2. verteile P auf Regionen von S_i



Vorgehen

Lemma 1: C' konsistente Vereinfachung von C bzgl. P $\Leftrightarrow$ kein Punkt von P liegt in Facette von CC'

Fläche zwischen C und C'

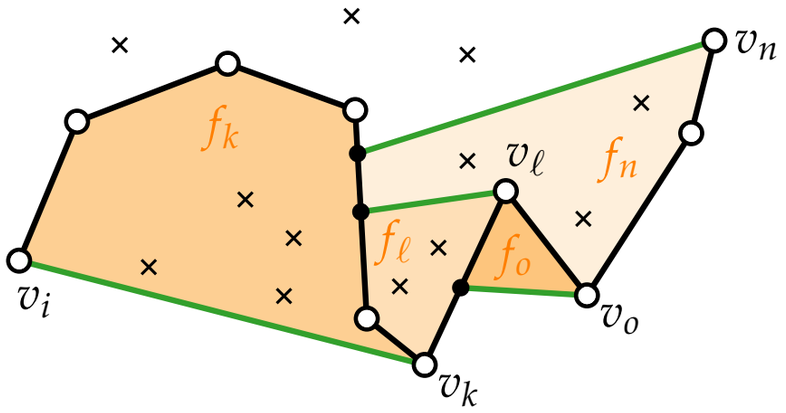
Bestimmung aller konsistenten Abkürzungen von v_i

- C_{ij}: Kantenzug von v_i bis v_j
- Q_{ij}: Polygon $C_{ij} \cup \overline{v_i v_j}$
- $(v_i, v_j) \in A \Leftrightarrow Q_{ij}$ enthält keine Punkte aus P

(v_i, v_j) ist eine **Tangente**, wenn v_{j-1} und v_{j+1} auf gleicher Seite von $v_i v_j$ liegen.

Vorgehen für festes v_i in 3 Schritten

1. berechne Tangentensegmente und induzierte Unterteilung S_i
2. verteile P auf Regionen von S_i
3. berechne konsistente Abkürzungen



Vorgehen

Lemma 1: C' konsistente Vereinfachung von C bzgl. P $\Leftrightarrow$ kein Punkt von P liegt in Facette von CC'

Fläche zwischen C und C'

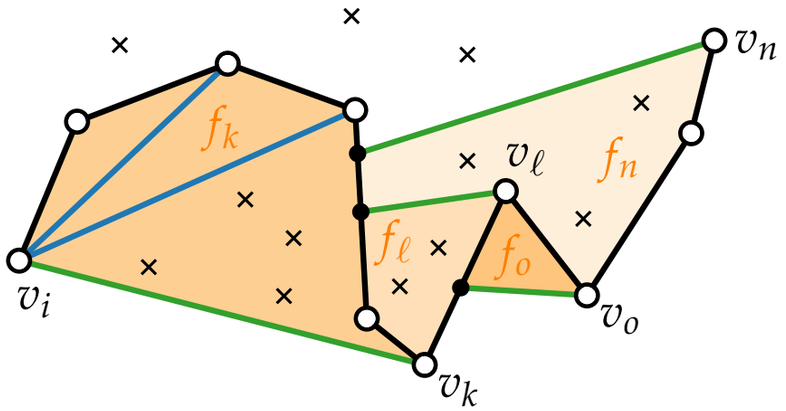
Bestimmung aller konsistenten Abkürzungen von v_i

- C_{ij}: Kantenzug von v_i bis v_j
- Q_{ij}: Polygon $C_{ij} \cup \overline{v_i v_j}$
- $(v_i, v_j) \in A \Leftrightarrow Q_{ij}$ enthält keine Punkte aus P

(v_i, v_j) ist eine **Tangente**, wenn v_{j-1} und v_{j+1} auf gleicher Seite von $v_i v_j$ liegen.

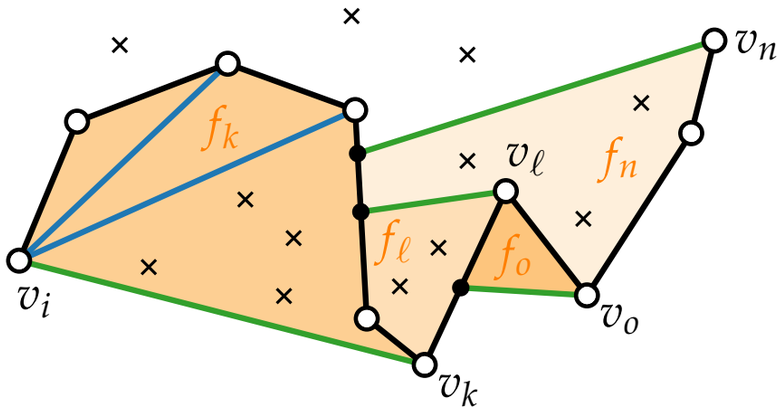
Vorgehen für festes v_i in 3 Schritten

1. berechne Tangentensegmente und induzierte Unterteilung S_i
2. verteile P auf Regionen von S_i
3. berechne konsistente Abkürzungen



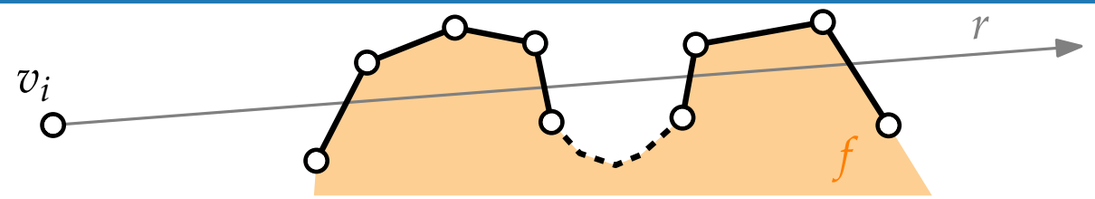
1) Tangentensegmente

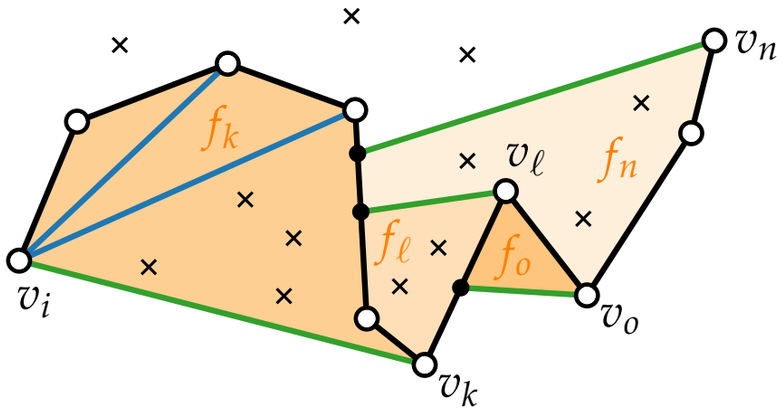
Lemma 2. Jede begrenzte Facette f von S_i ist **θ-monoton** bzgl. v_i, d.h. $|r \cap f| \in \{0, 1\}$ für jeden Strahl r von v_i



1) Tangentensegmente

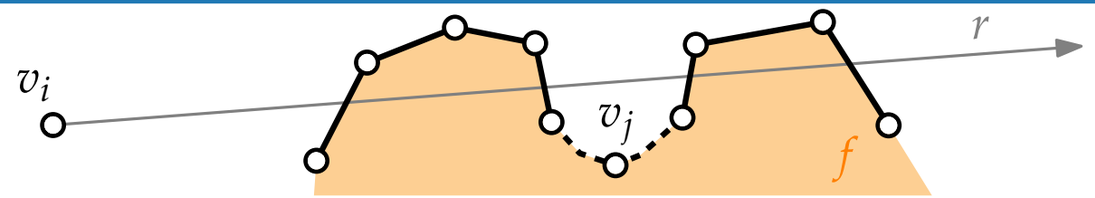
Lemma 2. Jede begrenzte Facette f von S_i ist **θ-monoton** bzgl. v_i, d.h. $|r \cap f| \in \{0,1\}$ für jeden Strahl r von v_i

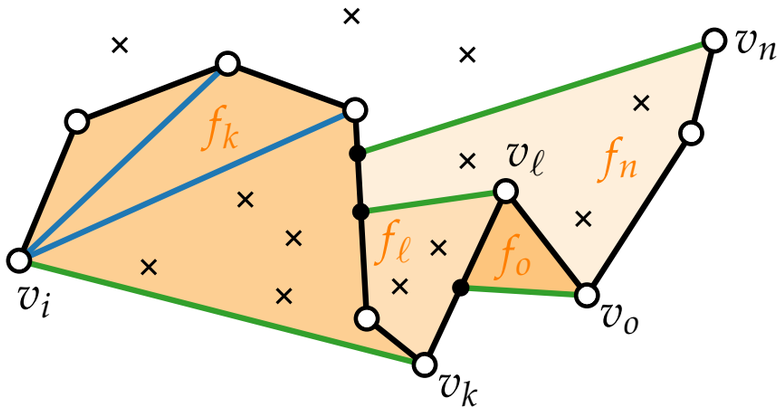




1) Tangentensegmente

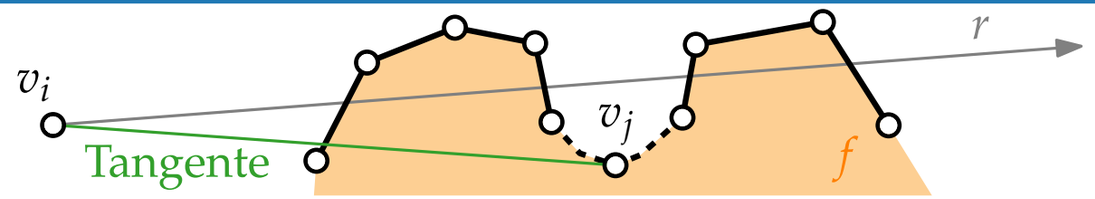
Lemma 2. Jede begrenzte Facette f von S_i ist **θ-monoton** bzgl. v_i, d.h. $|r \cap f| \in \{0,1\}$ für jeden Strahl r von v_i

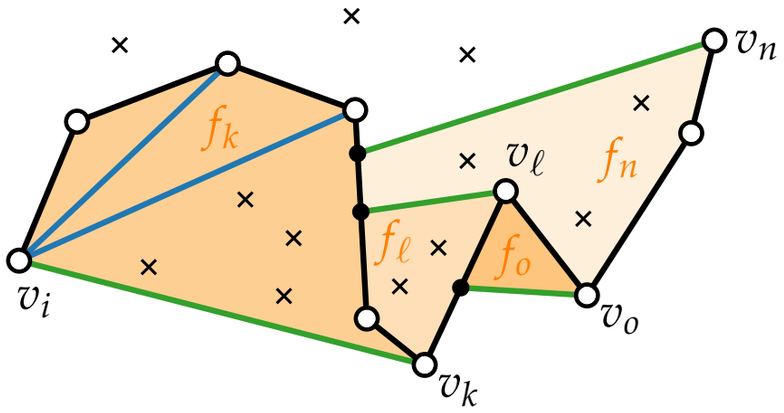




1) Tangentensegmente

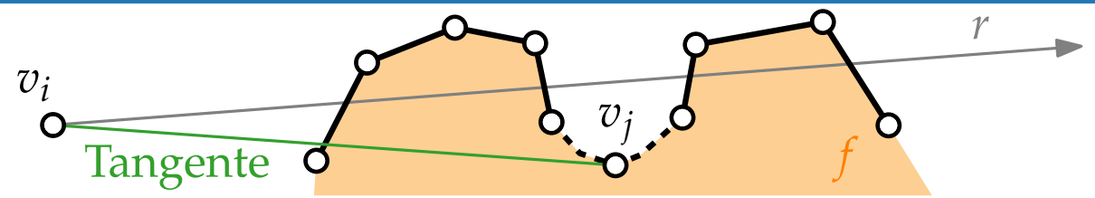
Lemma 2. Jede begrenzte Facette f von S_i ist **θ-monoton** bzgl. v_i, d.h. $|r \cap f| \in \{0,1\}$ für jeden Strahl r von v_i





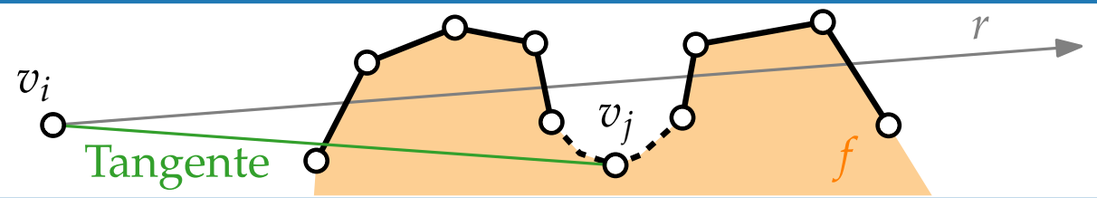
1) Tangentensegmente

Lemma 2. Jede begrenzte Facette f von S_i ist **θ-monoton** bzgl. v_i, d.h. $|r \cap f| \in \{0,1\}$ für jeden Strahl r von v_i

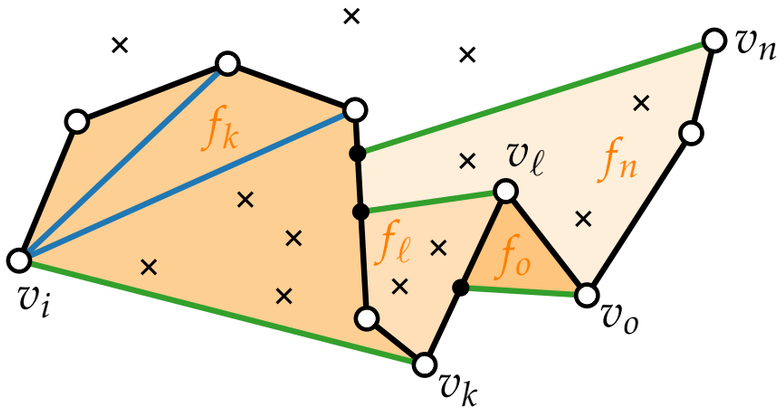


1) Tangentensegmente

Lemma 2. Jede begrenzte Facette f von S_i ist **θ-monoton** bzgl. v_i, d.h. $|r \cap f| \in \{0,1\}$ für jeden Strahl r von v_i

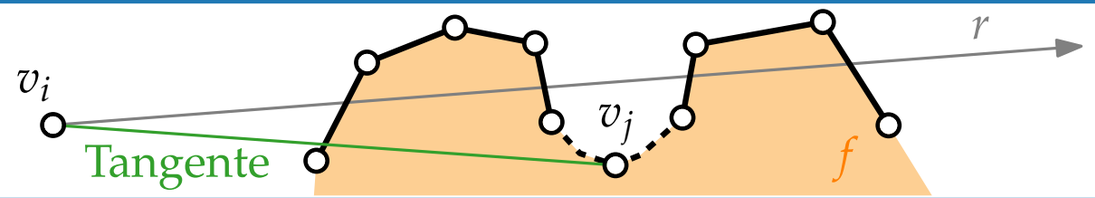


Lemma 3. Jede begrenzte Facette f von S_i besitzt genau einen zshg. Kantenzug, über den Strahlen r von v_i Facette f verlassen

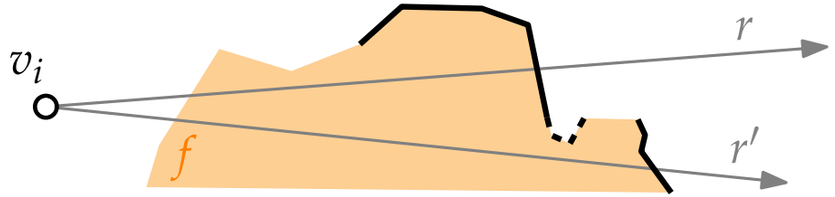


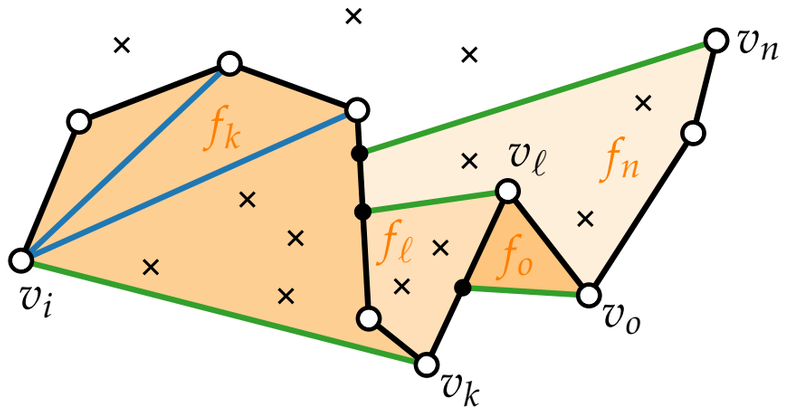
1) Tangentensegmente

Lemma 2. Jede begrenzte Facette f von S_i ist **θ-monoton** bzgl. v_i, d.h. $|r \cap f| \in \{0,1\}$ für jeden Strahl r von v_i



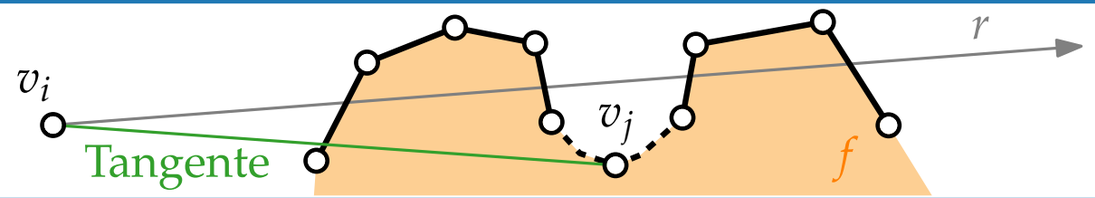
Lemma 3. Jede begrenzte Facette f von S_i besitzt genau einen zshg. Kantenzug, über den Strahlen r von v_i Facette f verlassen



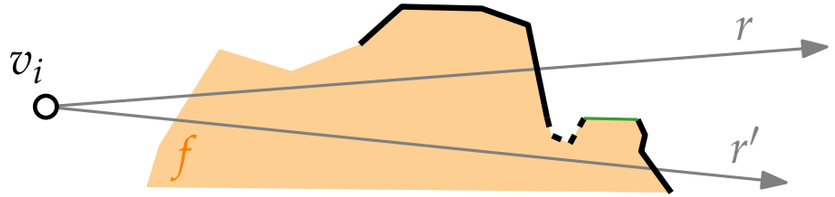


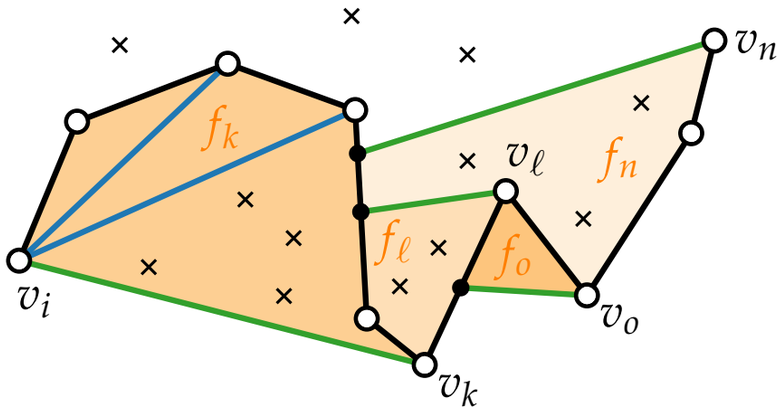
1) Tangentensegmente

Lemma 2. Jede begrenzte Facette f von S_i ist **θ-monoton** bzgl. v_i, d.h. $|r \cap f| \in \{0, 1\}$ für jeden Strahl r von v_i



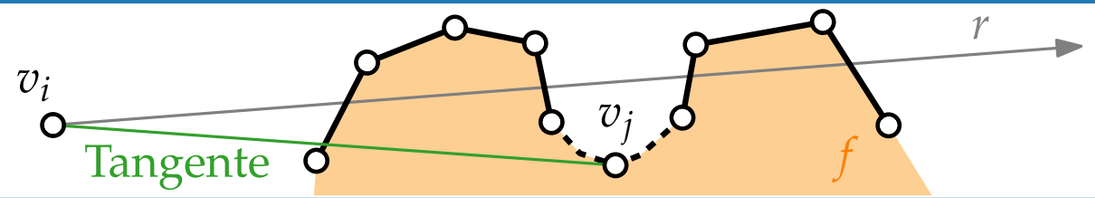
Lemma 3. Jede begrenzte Facette f von S_i besitzt genau einen zshg. Kantenzug, über den Strahlen r von v_i Facette f verlassen



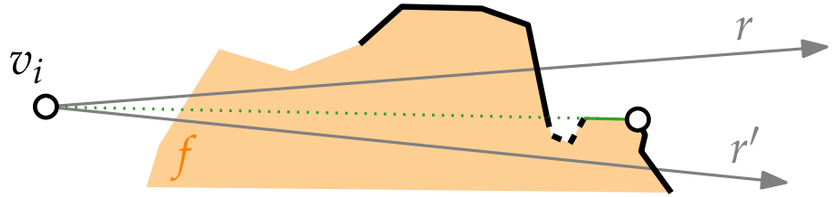


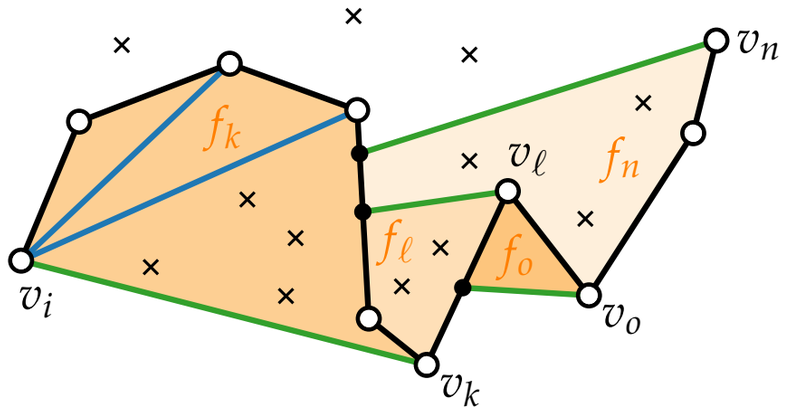
1) Tangentensegmente

Lemma 2. Jede begrenzte Facette f von S_i ist **θ-monoton** bzgl. v_i, d.h. $|r \cap f| \in \{0, 1\}$ für jeden Strahl r von v_i



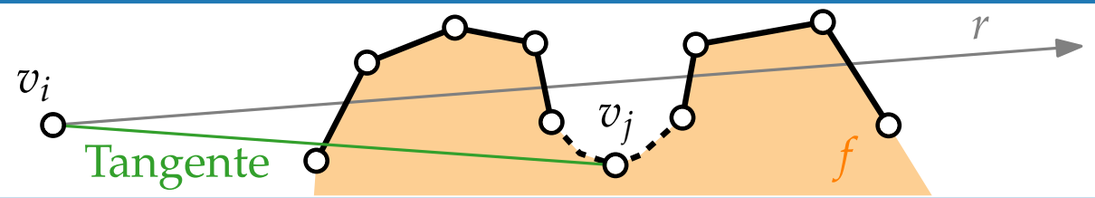
Lemma 3. Jede begrenzte Facette f von S_i besitzt genau einen zshg. Kantenzug, über den Strahlen r von v_i Facette f verlassen



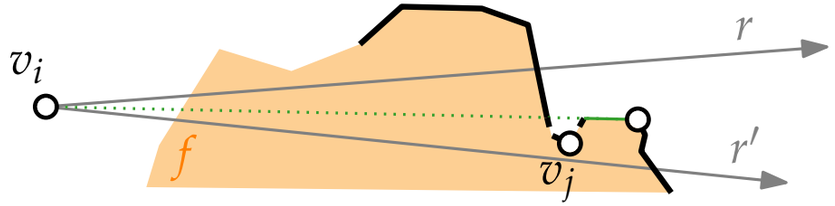


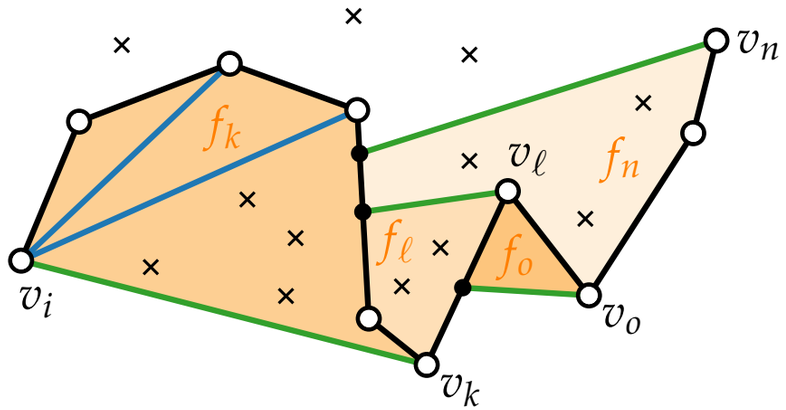
1) Tangentensegmente

Lemma 2. Jede begrenzte Facette f von S_i ist **θ-monoton** bzgl. v_i, d.h. $|r \cap f| \in \{0,1\}$ für jeden Strahl r von v_i



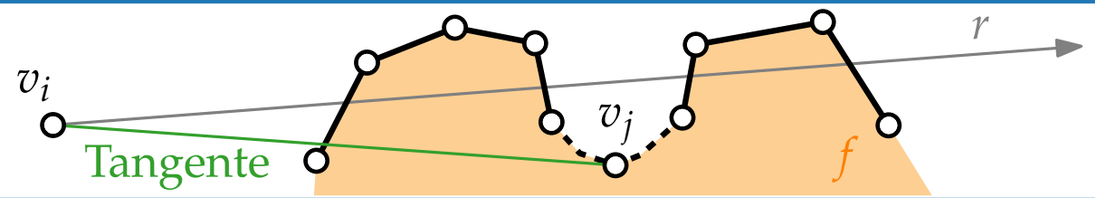
Lemma 3. Jede begrenzte Facette f von S_i besitzt genau einen zshg. Kantenzug, über den Strahlen r von v_i Facette f verlassen



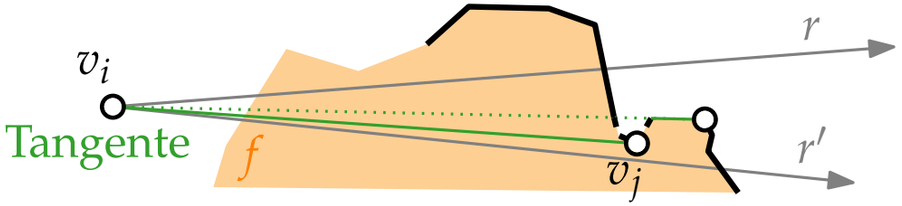


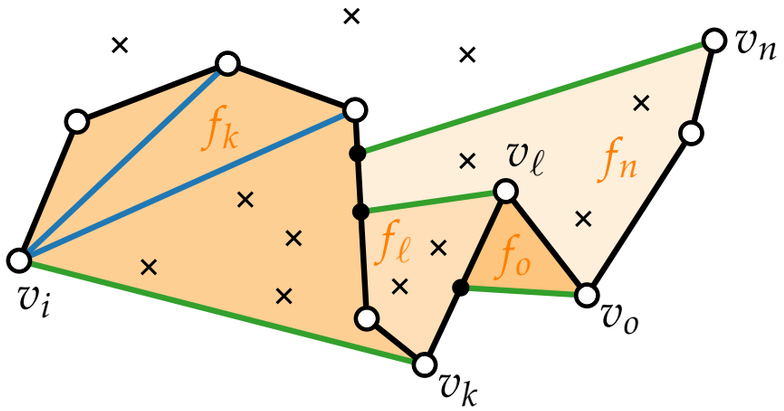
1) Tangentensegmente

Lemma 2. Jede begrenzte Facette f von S_i ist **θ-monoton** bzgl. v_i, d.h. $|r \cap f| \in \{0,1\}$ für jeden Strahl r von v_i



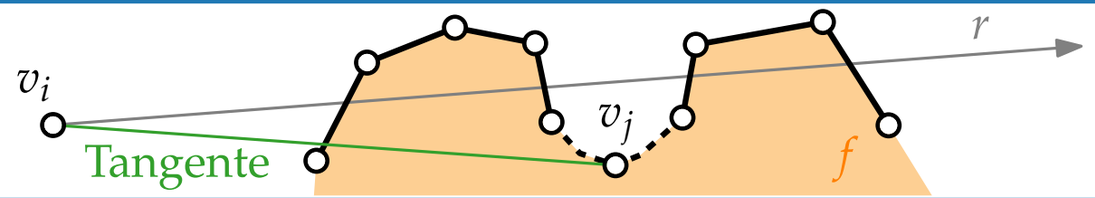
Lemma 3. Jede begrenzte Facette f von S_i besitzt genau einen zshg. Kantenzug, über den Strahlen r von v_i Facette f verlassen



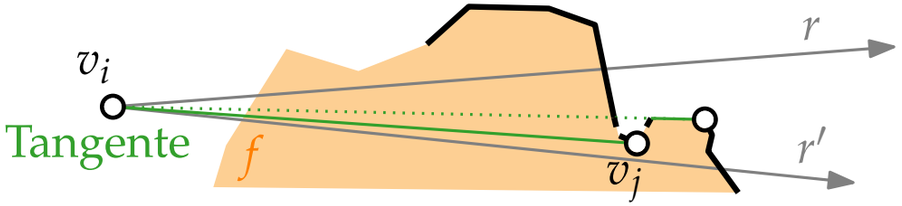


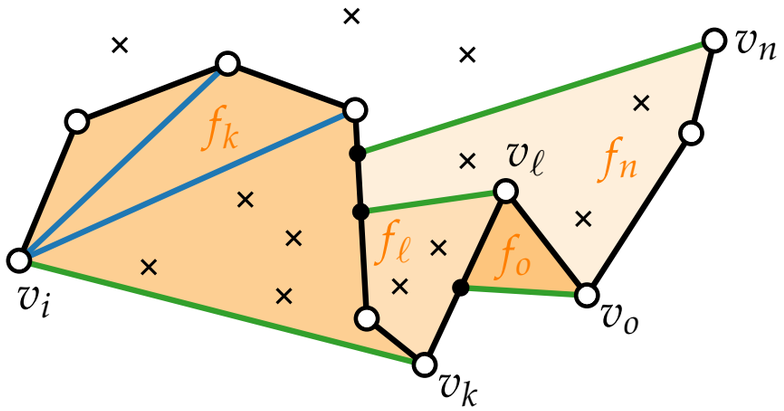
1) Tangentensegmente

Lemma 2. Jede begrenzte Facette f von S_i ist **θ-monoton** bzgl. v_i, d.h. $|r \cap f| \in \{0, 1\}$ für jeden Strahl r von v_i



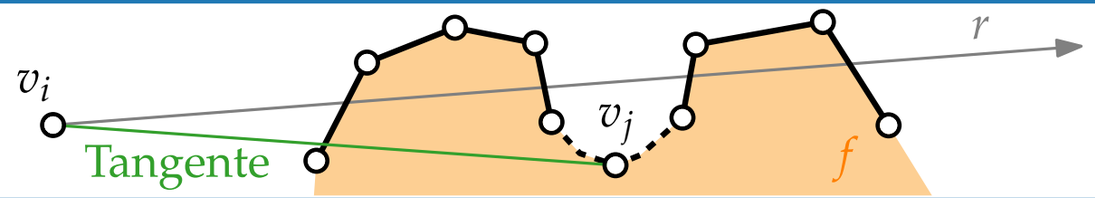
Lemma 3. Jede begrenzte Facette f von S_i besitzt genau einen zshg. Kantenzug, über den Strahlen r von v_i Facette f verlassen



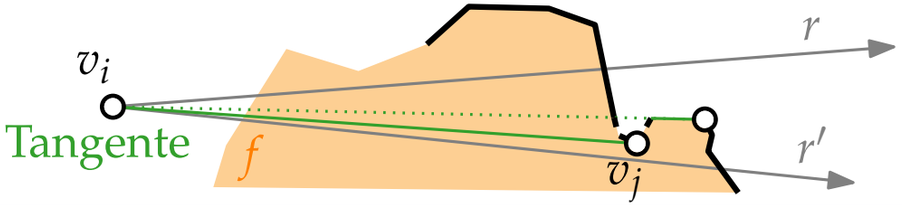


1) Tangentensegmente

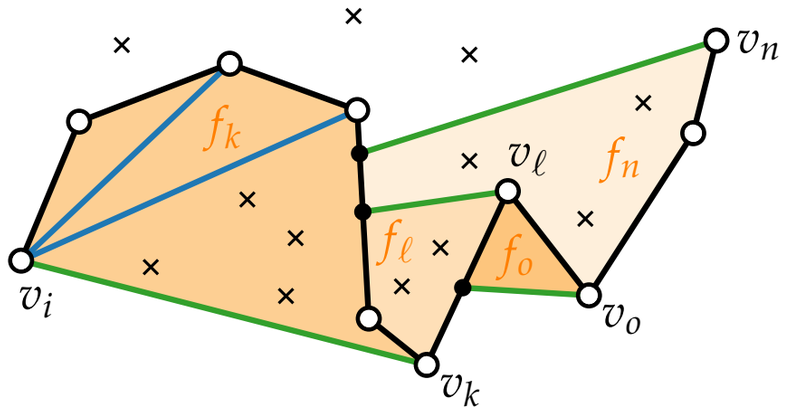
Lemma 2. Jede begrenzte Facette f von S_i ist **θ-monoton** bzgl. v_i, d.h. $|r \cap f| \in \{0, 1\}$ für jeden Strahl r von v_i



Lemma 3. Jede begrenzte Facette f von S_i besitzt genau einen zshg. Kantenzug, über den Strahlen r von v_i Facette f verlassen

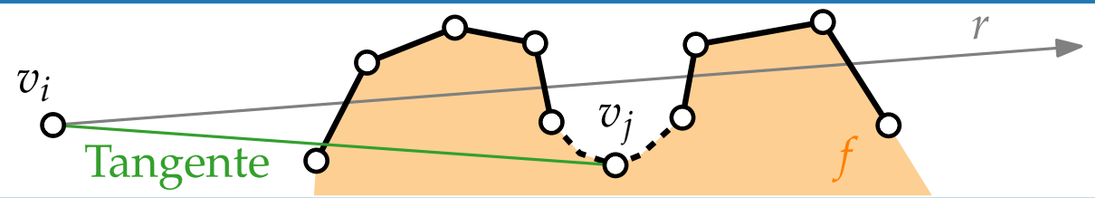


Lemma 4. Jeder Strahl von v_i schneidet die Facetten von S_i in aufsteigender Reihenfolge

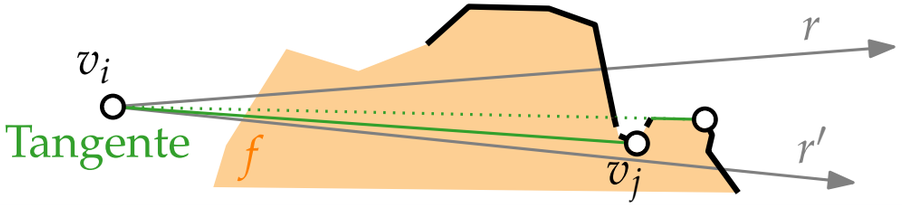


1) Tangentensegmente

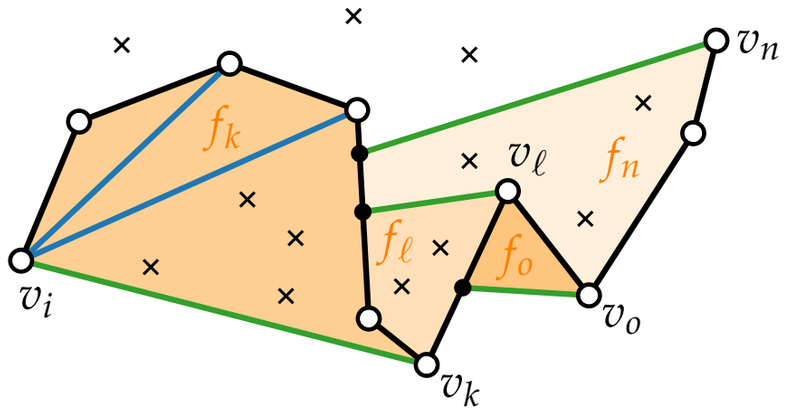
Lemma 2. Jede begrenzte Facette f von S_i ist **θ-monoton** bzgl. v_i, d.h. $|r \cap f| \in \{0, 1\}$ für jeden Strahl r von v_i



Lemma 3. Jede begrenzte Facette f von S_i besitzt genau einen zshg. Kantenzug, über den Strahlen r von v_i Facette f verlassen

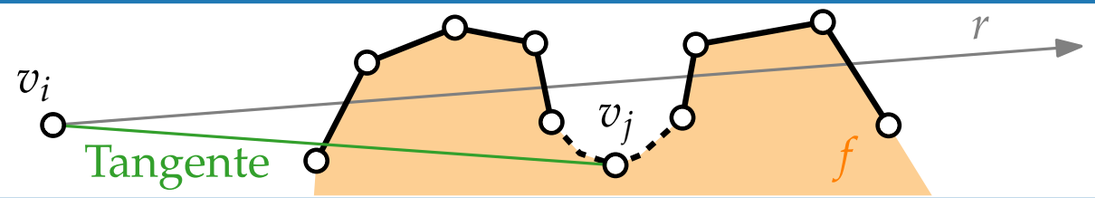


Lemma 4. Jeder Strahl von v_i schneidet die Facetten von S_i in aufsteigender Reihenfolge

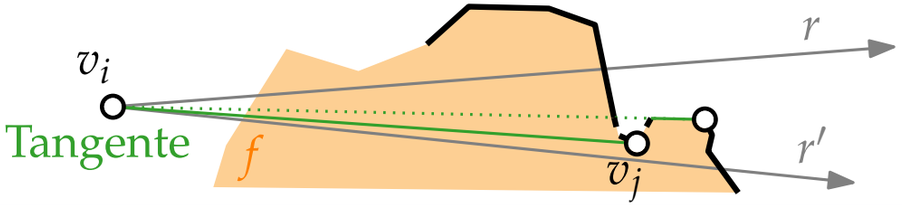


1) Tangentensegmente

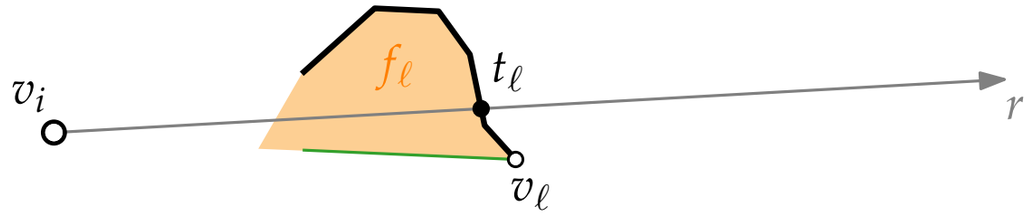
Lemma 2. Jede begrenzte Facette f von S_i ist **θ-monoton** bzgl. v_i, d.h. $|r \cap f| \in \{0, 1\}$ für jeden Strahl r von v_i

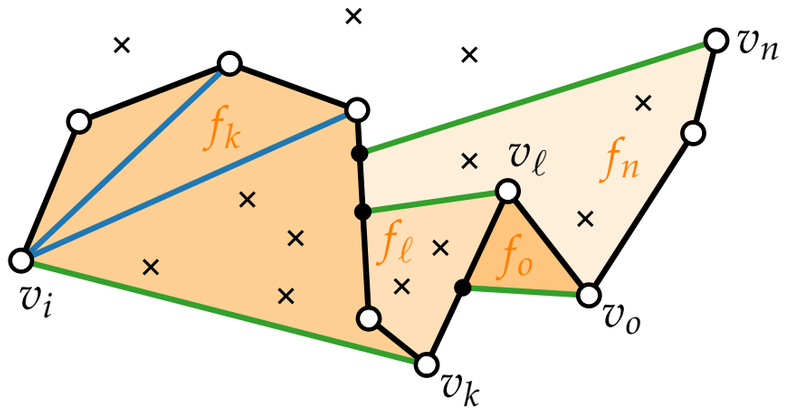


Lemma 3. Jede begrenzte Facette f von S_i besitzt genau einen zshg. Kantenzug, über den Strahlen r von v_i Facette f verlassen



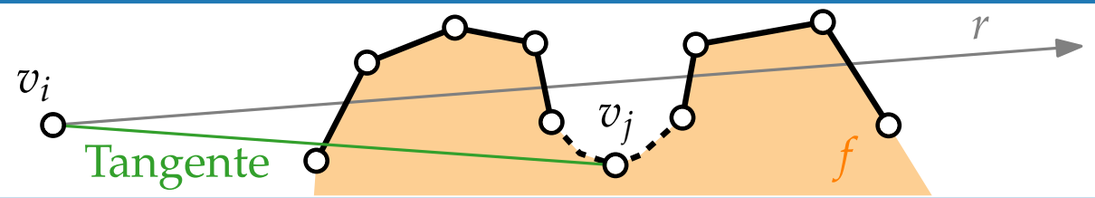
Lemma 4. Jeder Strahl von v_i schneidet die Facetten von S_i in aufsteigender Reihenfolge



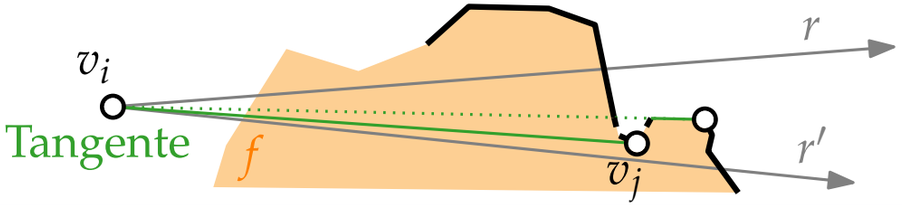


1) Tangentensegmente

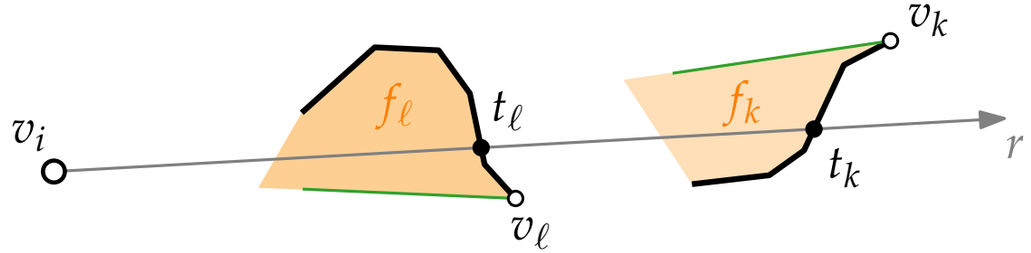
Lemma 2. Jede begrenzte Facette f von S_i ist **θ-monoton** bzgl. v_i, d.h. $|r \cap f| \in \{0, 1\}$ für jeden Strahl r von v_i

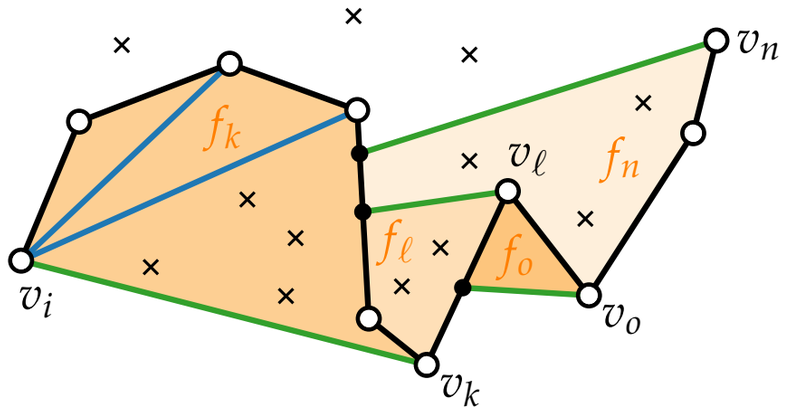


Lemma 3. Jede begrenzte Facette f von S_i besitzt genau einen zshg. Kantenzug, über den Strahlen r von v_i Facette f verlassen



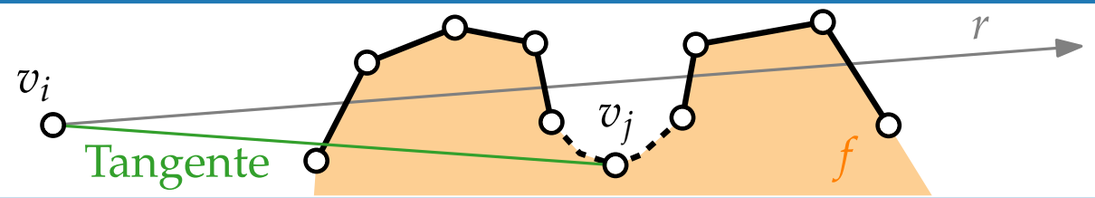
Lemma 4. Jeder Strahl von v_i schneidet die Facetten von S_i in aufsteigender Reihenfolge



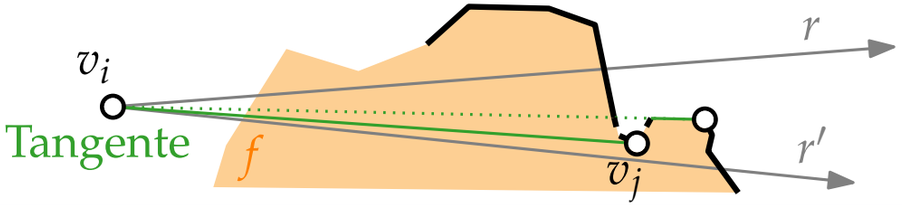


1) Tangentensegmente

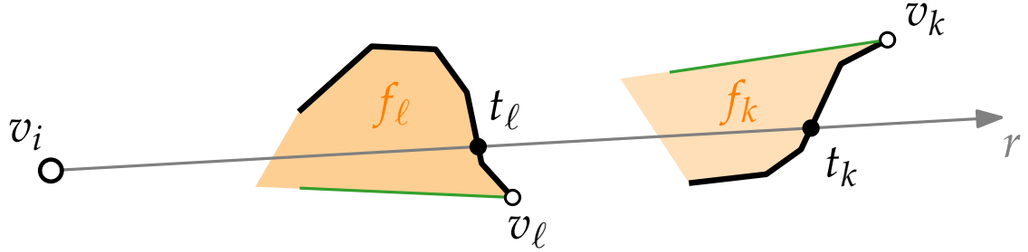
Lemma 2. Jede begrenzte Facette f von S_i ist **θ-monoton** bzgl. v_i, d.h. $|r \cap f| \in \{0, 1\}$ für jeden Strahl r von v_i



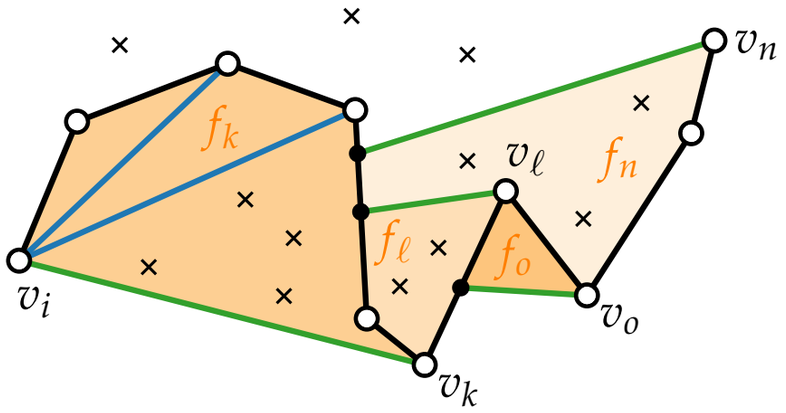
Lemma 3. Jede begrenzte Facette f von S_i besitzt genau einen zshg. Kantenzug, über den Strahlen r von v_i Facette f verlassen



Lemma 4. Jeder Strahl von v_i schneidet die Facetten von S_i in aufsteigender Reihenfolge

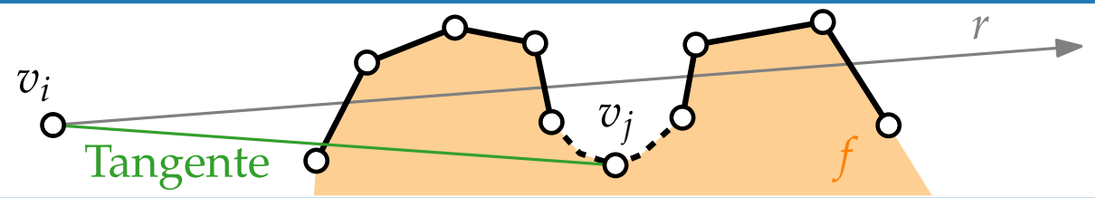


$k < \ell$

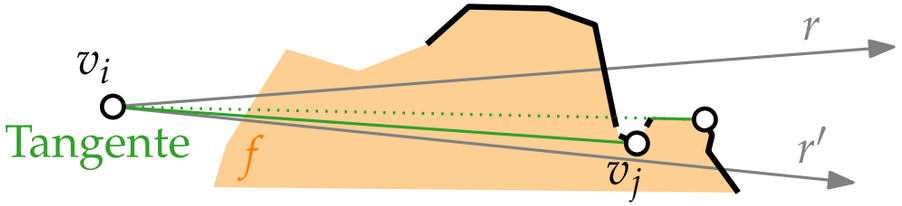


1) Tangentensegmente

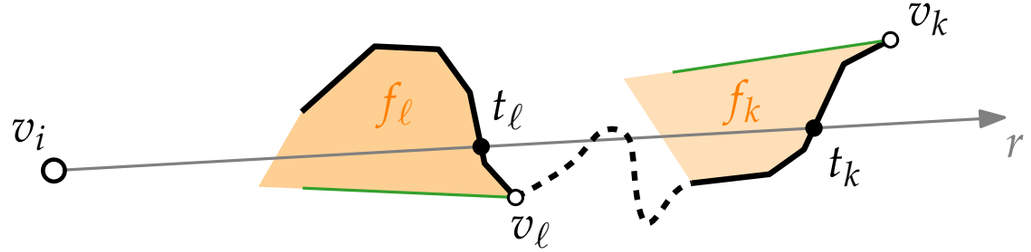
Lemma 2. Jede begrenzte Facette f von S_i ist **θ-monoton** bzgl. v_i, d.h. $|r \cap f| \in \{0, 1\}$ für jeden Strahl r von v_i



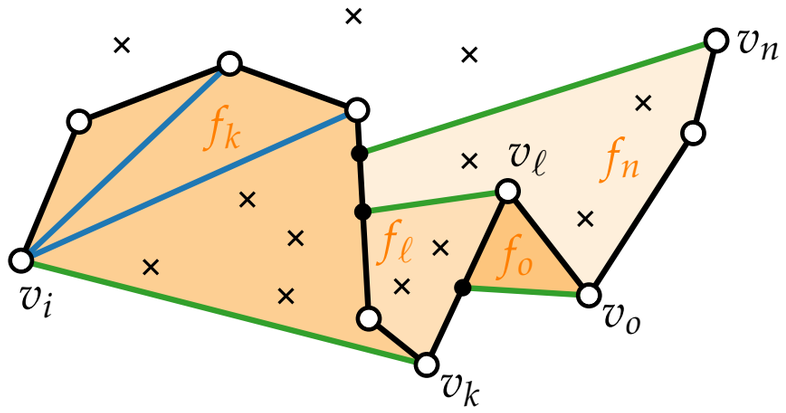
Lemma 3. Jede begrenzte Facette f von S_i besitzt genau einen zshg. Kantenzug, über den Strahlen r von v_i Facette f verlassen



Lemma 4. Jeder Strahl von v_i schneidet die Facetten von S_i in aufsteigender Reihenfolge

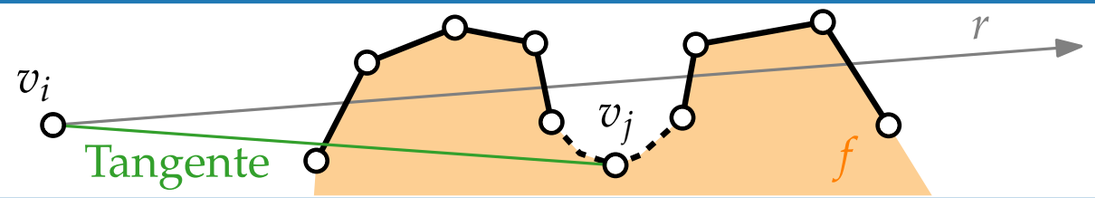


$k < \ell$

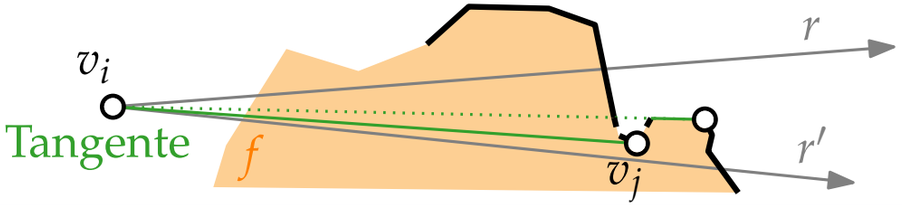


1) Tangentensegmente

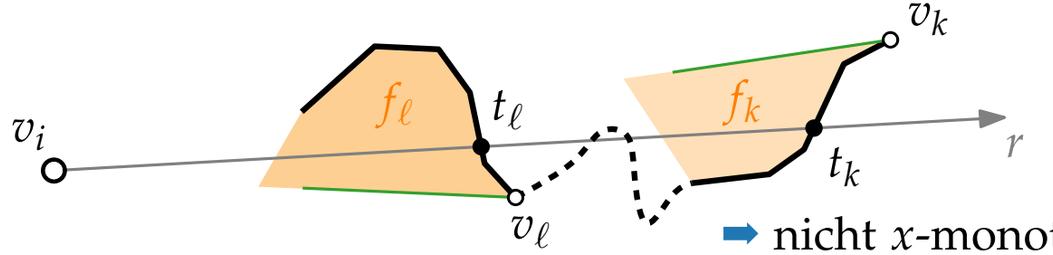
Lemma 2. Jede begrenzte Facette f von S_i ist **θ-monoton** bzgl. v_i, d.h. $|r \cap f| \in \{0, 1\}$ für jeden Strahl r von v_i



Lemma 3. Jede begrenzte Facette f von S_i besitzt genau einen zshg. Kantenzug, über den Strahlen r von v_i Facette f verlassen

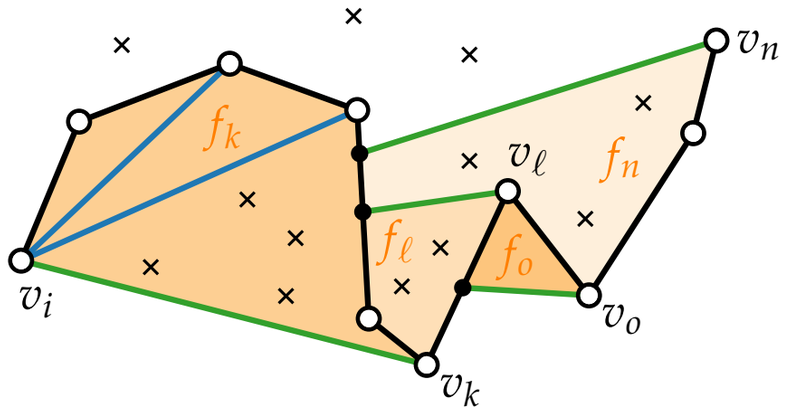


Lemma 4. Jeder Strahl von v_i schneidet die Facetten von S_i in aufsteigender Reihenfolge



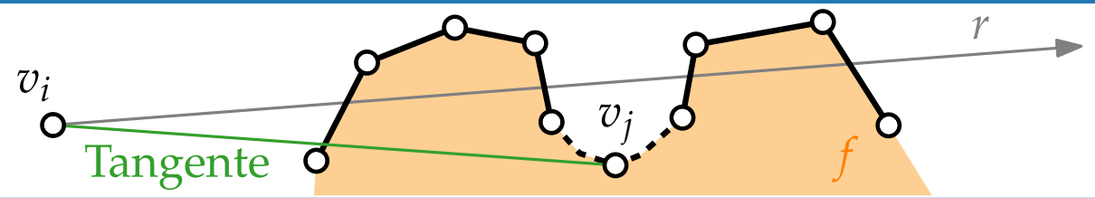
➡ nicht x-monoton

$k < \ell$

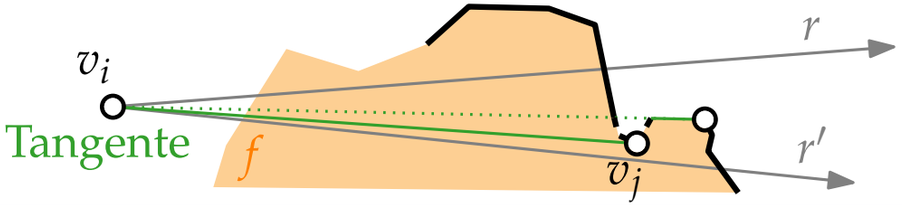


1) Tangentensegmente

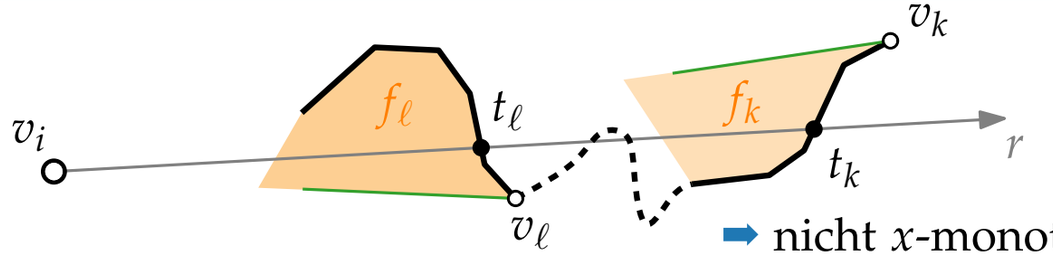
Lemma 2. Jede begrenzte Facette f von S_i ist **θ-monoton** bzgl. v_i, d.h. $|r \cap f| \in \{0,1\}$ für jeden Strahl r von v_i



Lemma 3. Jede begrenzte Facette f von S_i besitzt genau einen zshg. Kantenzug, über den Strahlen r von v_i Facette f verlassen

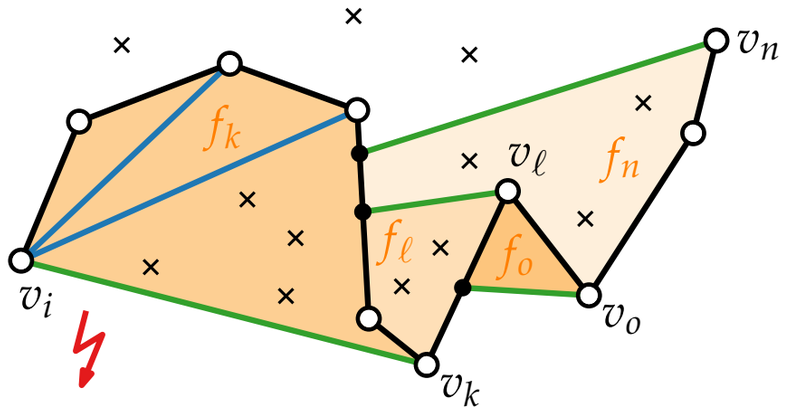


Lemma 4. Jeder Strahl von v_i schneidet die Facetten von S_i in aufsteigender Reihenfolge



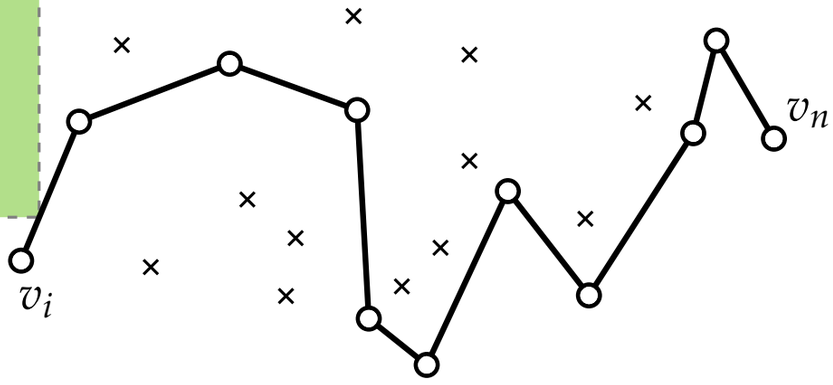
➡ nicht x-monoton

$k < \ell$



1) Tangentensegmente: Algorithmus

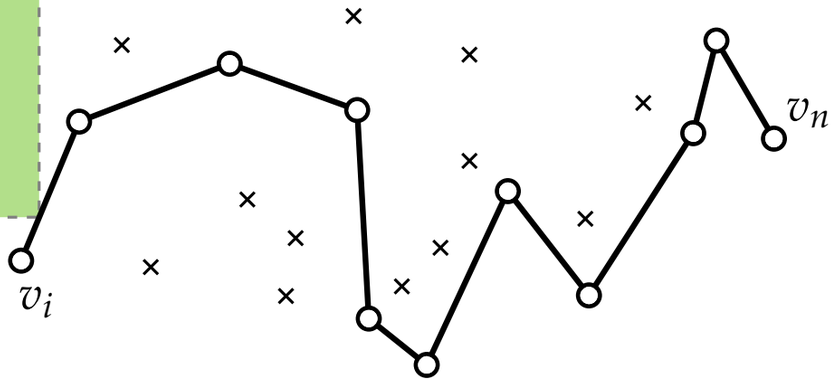
TANGENTENSEGMENTE(C, i)



1) Tangentensegmente: Algorithmus

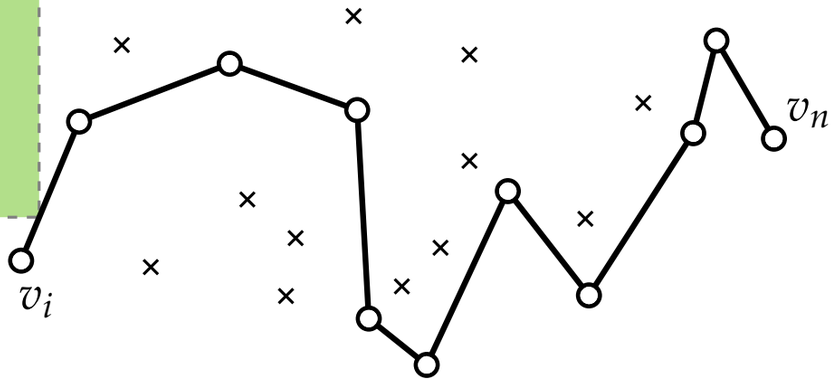
TANGENTENSEGMENTE(C, i)

for $j = i + 1, \ldots, n$ **do**



1) Tangentensegmente: Algorithmus

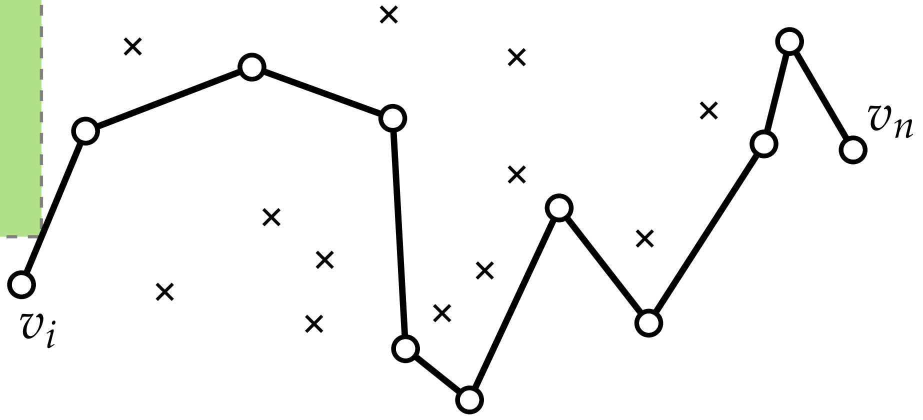
```
TANGENTENSEGMENTE(C, i)
  for j = i + 1, ..., n do
    if (v_i, v_j) maximale Tangente then
      
      
      
      
      
      
      
      
      
    else if (v_i, v_j) minimale Tangente then
      ...                                  // analoges Vorgehen
```



1) Tangentensegmente: Algorithmus

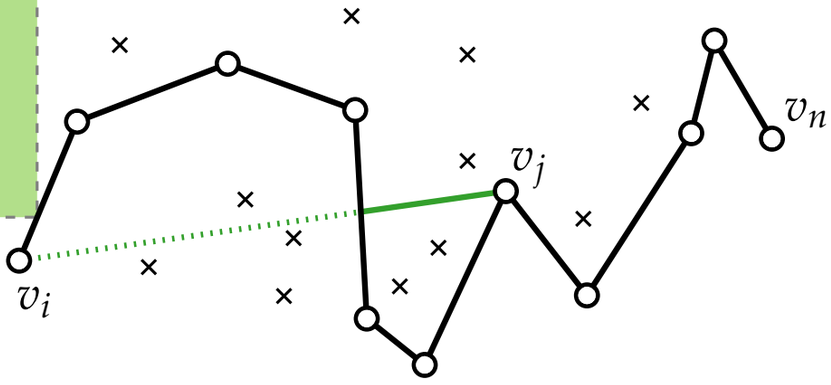
```
TANGENTENSEGMENTE(C, i)
  for j = i+1, ..., n do
    if (v_i, v_j) maximale Tangente then

    else if (v_i, v_j) minimale Tangente then
      ...                      // analoges Vorgehen
```



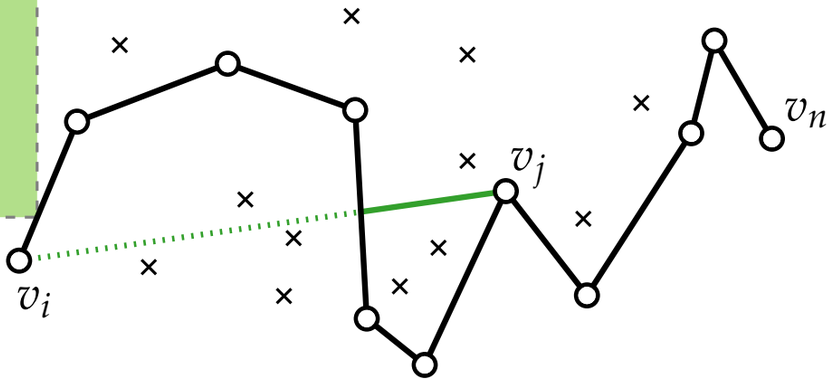
1) Tangentensegmente: Algorithmus

```
TANGENTENSEGMENTE(C, i)
  for j = i + 1, ..., n do
    if (v_i, v_j) maximale Tangente then
      ...
    else if (v_i, v_j) minimale Tangente then
      ...                                   // analoges Vorgehen
```



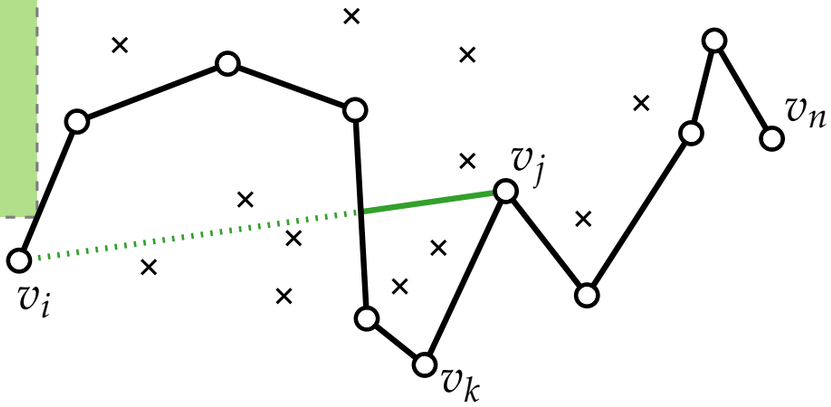
1) Tangentensegmente: Algorithmus

```
TANGENTENSEGMENTE(C, i)
  for j = i + 1, ..., n do
    if (v_i, v_j) maximale Tangente then
      k = j − 1
    else if (v_i, v_j) minimale Tangente then
      ...                                  // analoges Vorgehen
```



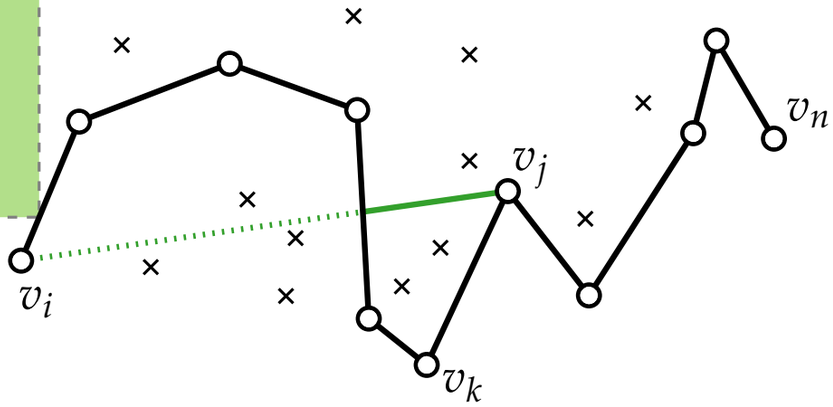
1) Tangentensegmente: Algorithmus

```
TANGENTENSEGMENTE(C, i)
  for j = i + 1, ..., n do
    if (v_i, v_j) maximale Tangente then
      k = j − 1
    else if (v_i, v_j) minimale Tangente then
      ...                                  // analoges Vorgehen
```



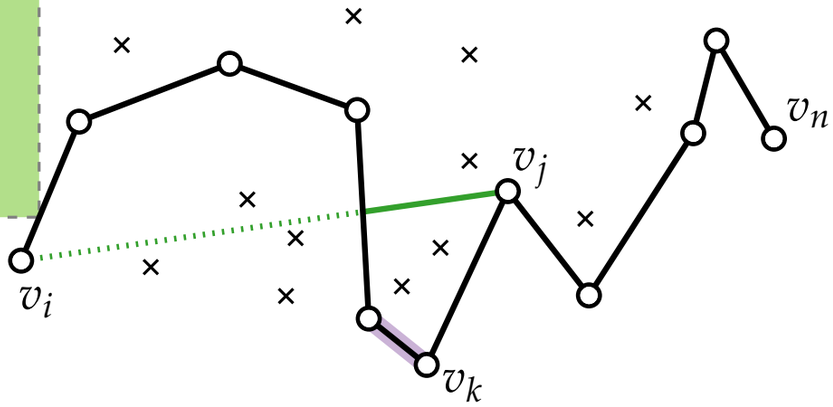
1) Tangentensegmente: Algorithmus

```
TANGENTENSEGMENTE(C, i)
  for j = i + 1, ..., n do
    if (v_i, v_j) maximale Tangente then
      k = j − 1
      while (v_i, v_j) ∩ (v_{k−1}, v_k) = ∅ do
        
    else if (v_i, v_j) minimale Tangente then
      ...                                   // analoges Vorgehen
```



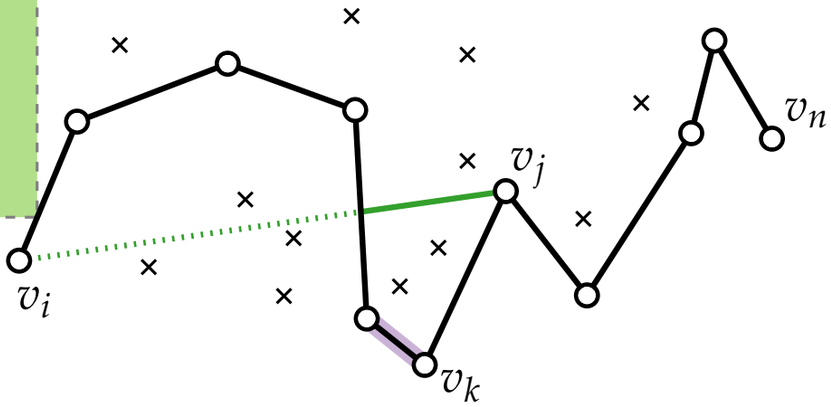
1) Tangentensegmente: Algorithmus

```
TANGENTENSEGMENTE(C, i)
  for j = i + 1, ..., n do
    if (v_i, v_j) maximale Tangente then
      k = j − 1
      while (v_i, v_j) ∩ (v_{k−1}, v_k) = ∅ do
        
    else if (v_i, v_j) minimale Tangente then
      ...                                   // analoges Vorgehen
```



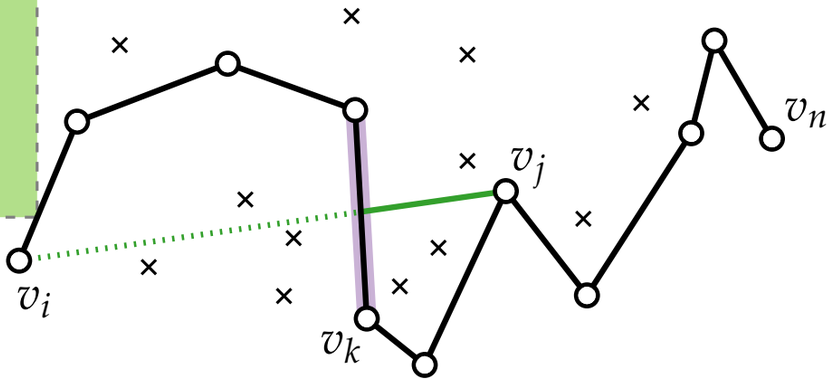
1) Tangentensegmente: Algorithmus

```
TANGENTENSEGMENTE(C, i)
  for j = i + 1, ..., n do
    if (v_i, v_j) maximale Tangente then
      k = j − 1
      while (v_i, v_j) ∩ (v_{k−1}, v_k) = ∅ do
        k = k − 1
    else if (v_i, v_j) minimale Tangente then
      ...                                   // analoges Vorgehen
```



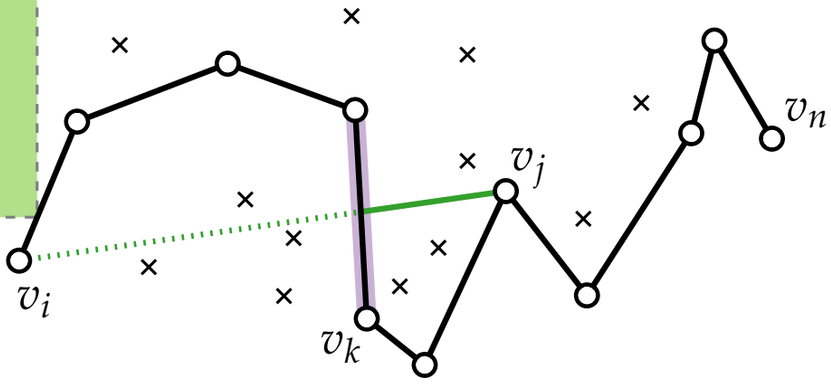
1) Tangentensegmente: Algorithmus

```
TANGENTENSEGMENTE(C, i)
  for j = i + 1, ..., n do
    if (v_i, v_j) maximale Tangente then
      k = j − 1
      while (v_i, v_j) ∩ (v_{k−1}, v_k) = ∅ do
        k = k − 1
    else if (v_i, v_j) minimale Tangente then
      ...                                   // analoges Vorgehen
```



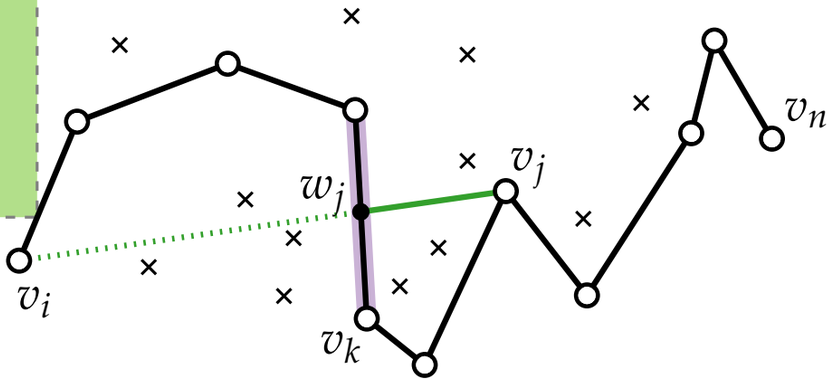
1) Tangentensegmente: Algorithmus

```
TANGENTENSEGMENTE(C, i)
  for j = i + 1, ..., n do
    if (v_i, v_j) maximale Tangente then
      k = j − 1
      while (v_i, v_j) ∩ (v_{k−1}, v_k) = ∅ do
        k = k − 1
      w_j = (v_i, v_j) ∩ (v_{k−1}, v_k)
    else if (v_i, v_j) minimale Tangente then
      ...                                  // analoges Vorgehen
```



1) Tangentensegmente: Algorithmus

```
TANGENTENSEGMENTE(C, i)
  for j = i + 1, ..., n do
    if (v_i, v_j) maximale Tangente then
      k = j − 1
      while (v_i, v_j) ∩ (v_{k−1}, v_k) = ∅ do
        k = k − 1
      w_j = (v_i, v_j) ∩ (v_{k−1}, v_k)
    else if (v_i, v_j) minimale Tangente then
      ...                                  // analoges Vorgehen
```



1) Tangentensegmente: Algorithmus

Tangentensegmente(C, i)

for $j = i+1, \ldots, n$ **do**

  if (v_i, v_j) maximale Tangente **then**

    $k = j - 1$

    while $(v_i, v_j) \cap (v_{k-1}, v_k) = \emptyset$ **do**

      $k = k - 1$

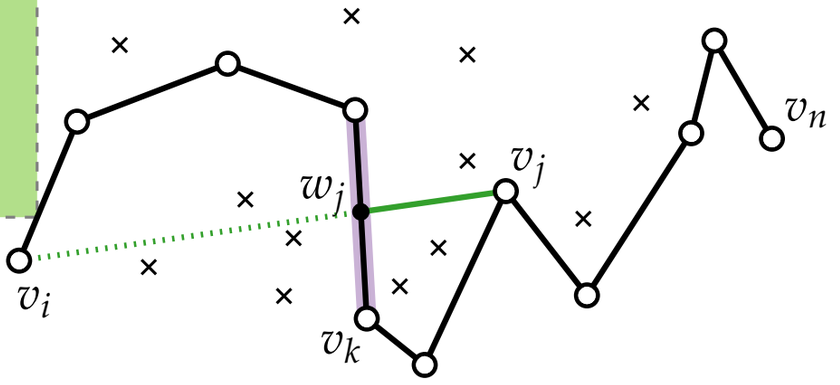
      if (v_i, v_k) maximale Tangente **then**

        $k = h$, wobei (v_{h-1}, v_h) Punkt w_k enthält

    $w_j = (v_i, v_j) \cap (v_{k-1}, v_k)$

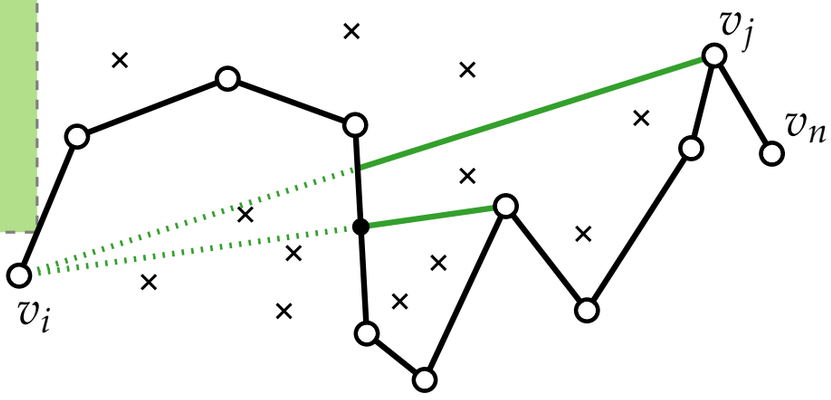
  else if (v_i, v_j) minimale Tangente **then**

    $\ldots$ `// analoges Vorgehen`



1) Tangentensegmente: Algorithmus

```
TANGENTENSEGMENTE(C, i)
  for j = i + 1, ..., n do
    if (v_i, v_j) maximale Tangente then
      k = j − 1
      while (v_i, v_j) ∩ (v_{k−1}, v_k) = ∅ do
        k = k − 1
        if (v_i, v_k) maximale Tangente then
          k = h, wobei (v_{h−1}, v_h) Punkt w_k enthält
      w_j = (v_i, v_j) ∩ (v_{k−1}, v_k)
    else if (v_i, v_j) minimale Tangente then
      ...                          // analoges Vorgehen
```



1) Tangentensegmente: Algorithmus

Tangentensegmente(C, i)

for $j = i+1, \ldots, n$ **do**

  if (v_i, v_j) maximale Tangente **then**

    $k = j - 1$

    while $(v_i, v_j) \cap (v_{k-1}, v_k) = \emptyset$ **do**

      $k = k - 1$

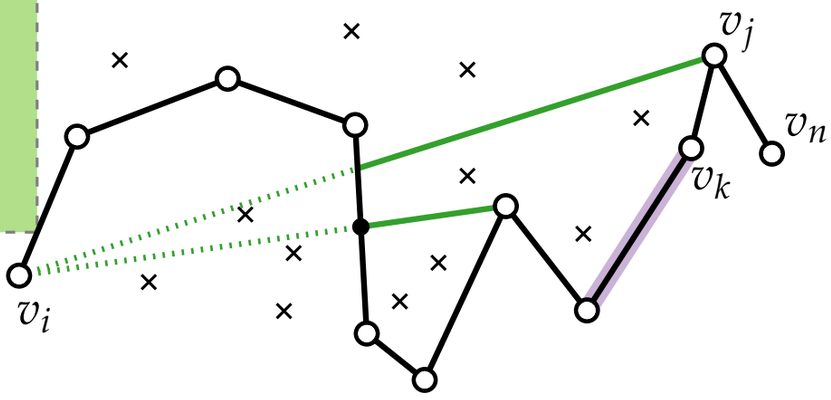
      if (v_i, v_k) maximale Tangente **then**

        $k = h$, wobei (v_{h-1}, v_h) Punkt w_k enthält

    $w_j = (v_i, v_j) \cap (v_{k-1}, v_k)$

  else if (v_i, v_j) minimale Tangente **then**

    $\ldots$ `// analoges Vorgehen`



1) Tangentensegmente: Algorithmus

Tangentensegmente(C, i)

for $j = i+1, \ldots, n$ **do**

  if (v_i, v_j) maximale Tangente **then**

    $k = j - 1$

    while $(v_i, v_j) \cap (v_{k-1}, v_k) = \emptyset$ **do**

      $k = k - 1$

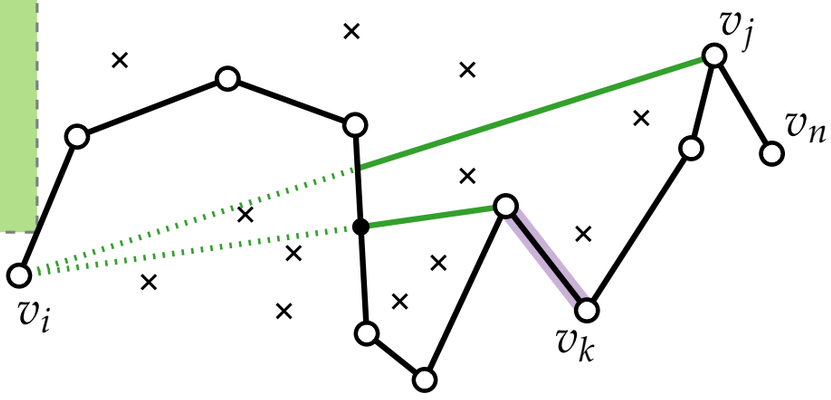
      if (v_i, v_k) maximale Tangente **then**

        $k = h$, wobei (v_{h-1}, v_h) Punkt w_k enthält

    $w_j = (v_i, v_j) \cap (v_{k-1}, v_k)$

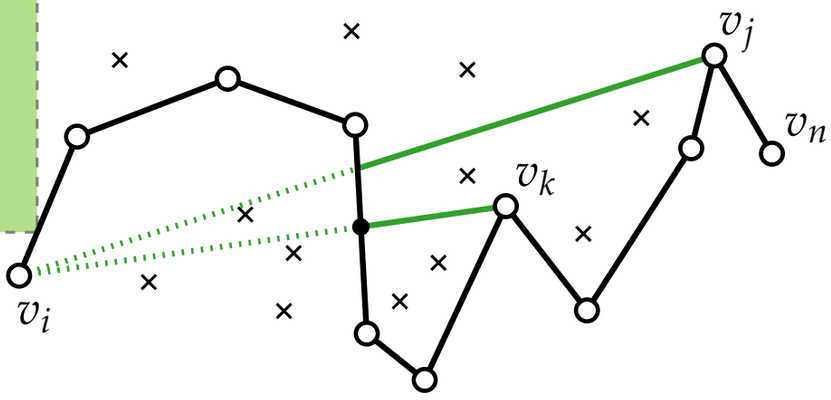
  else if (v_i, v_j) minimale Tangente **then**

    $\ldots$ `// analoges Vorgehen`



1) Tangentensegmente: Algorithmus

```
TANGENTENSEGMENTE(C, i)
  for j = i + 1, ..., n do
    if (v_i, v_j) maximale Tangente then
      k = j − 1
      while (v_i, v_j) ∩ (v_{k−1}, v_k) = ∅ do
        k = k − 1
        if (v_i, v_k) maximale Tangente then
          k = h, wobei (v_{h−1}, v_h) Punkt w_k enthält
      w_j = (v_i, v_j) ∩ (v_{k−1}, v_k)
    else if (v_i, v_j) minimale Tangente then
      ...                          // analoges Vorgehen
```



1) Tangentensegmente: Algorithmus

Tangentensegmente(C, i)

for $j = i+1, \ldots, n$ **do**

  if (v_i, v_j) maximale Tangente **then**

    $k = j - 1$

    while $(v_i, v_j) \cap (v_{k-1}, v_k) = \emptyset$ **do**

      $k = k - 1$

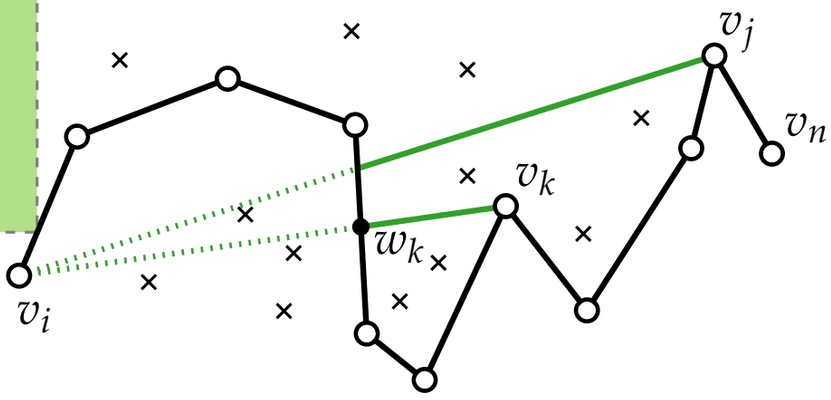
      if (v_i, v_k) maximale Tangente **then**

        $k = h$, wobei (v_{h-1}, v_h) Punkt w_k enthält

    $w_j = (v_i, v_j) \cap (v_{k-1}, v_k)$

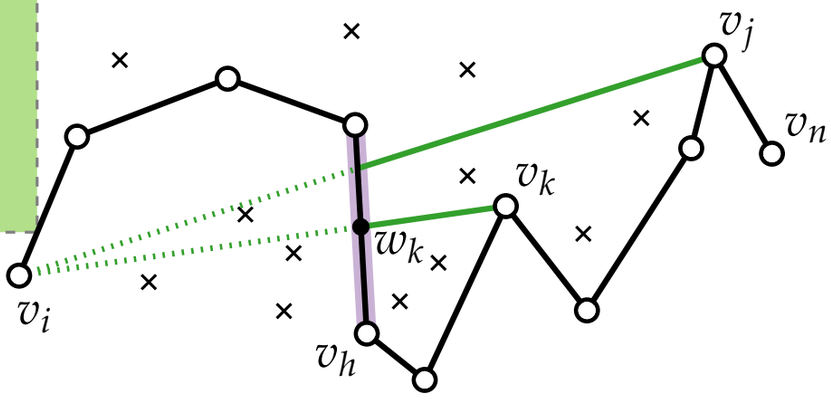
  else if (v_i, v_j) minimale Tangente **then**

    $\ldots$ `// analoges Vorgehen`



1) Tangentensegmente: Algorithmus

```
TANGENTENSEGMENTE(C, i)
  for j = i + 1, ..., n do
    if (v_i, v_j) maximale Tangente then
      k = j − 1
      while (v_i, v_j) ∩ (v_{k−1}, v_k) = ∅ do
        k = k − 1
        if (v_i, v_k) maximale Tangente then
          k = h, wobei (v_{h−1}, v_h) Punkt w_k enthält
      w_j = (v_i, v_j) ∩ (v_{k−1}, v_k)
    else if (v_i, v_j) minimale Tangente then
      ...                          // analoges Vorgehen
```



1) Tangentensegmente: Algorithmus

TANGENTENSEGMENTE(C, i)

for $j = i+1, \ldots, n$ **do**

  if (v_i, v_j) maximale Tangente **then**

    $k = j - 1$

    while $(v_i, v_j) \cap (v_{k-1}, v_k) = \emptyset$ **do**

      $k = k - 1$

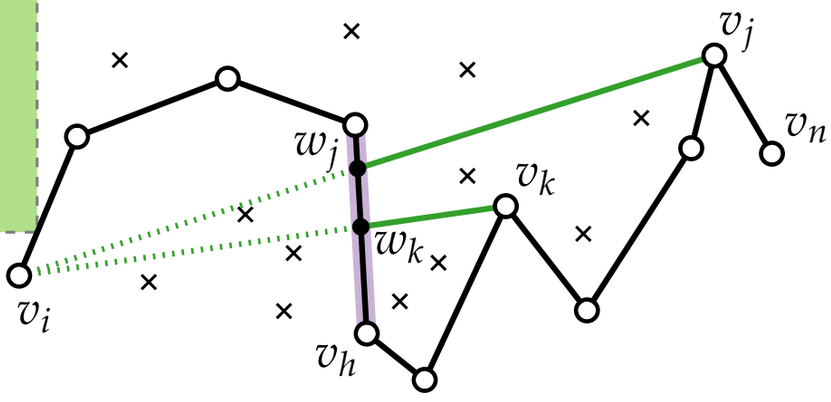
      if (v_i, v_k) maximale Tangente **then**

        $k = h$, wobei (v_{h-1}, v_h) Punkt w_k enthält

    $w_j = (v_i, v_j) \cap (v_{k-1}, v_k)$

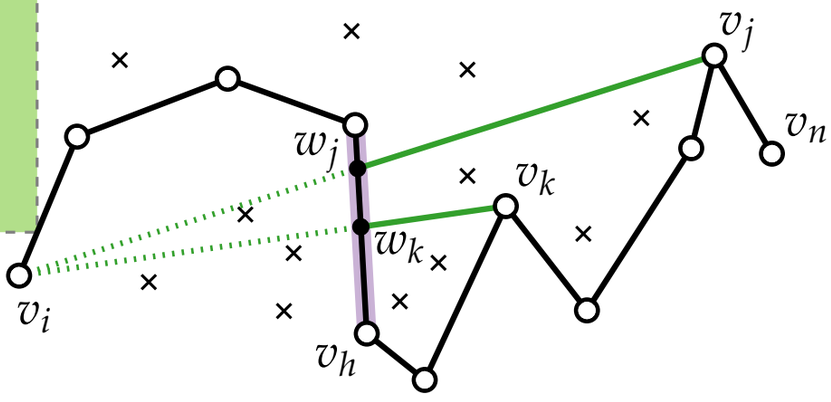
  else if (v_i, v_j) minimale Tangente **then**

    ... `// analoges Vorgehen`



1) Tangentensegmente: Algorithmus

```
TANGENTENSEGMENTE(C, i)
  for j = i + 1, ..., n do
    if (v_i, v_j) maximale Tangente then
      k = j − 1
      while (v_i, v_j) ∩ (v_{k−1}, v_k) = ∅ do
        k = k − 1
        if (v_i, v_k) maximale Tangente then
          k = h, wobei (v_{h−1}, v_h) Punkt w_k enthält
      w_j = (v_i, v_j) ∩ (v_{k−1}, v_k)
    else if (v_i, v_j) minimale Tangente then
      ...                          // analoges Vorgehen
```



Laufzeit?

1) Tangentensegmente: Algorithmus

Tangentensegmente(C, i)

for $j = i+1, \ldots, n$ **do**

  if (v_i, v_j) maximale Tangente **then**

    $k = j - 1$

    while $(v_i, v_j) \cap (v_{k-1}, v_k) = \emptyset$ **do**

      $k = k - 1$

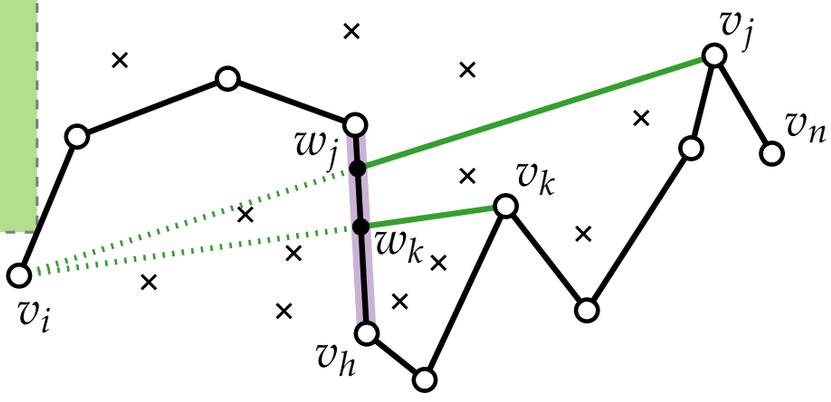
      if (v_i, v_k) maximale Tangente **then**

        $k = h$, wobei (v_{h-1}, v_h) Punkt w_k enthält

    $w_j = (v_i, v_j) \cap (v_{k-1}, v_k)$

  else if (v_i, v_j) minimale Tangente **then**

    $\ldots$ `// analoges Vorgehen`



Laufzeit? Jeder Knoten wird beim Zurücklaufen max. 1 mal „übersprungen“

1) Tangentensegmente: Algorithmus

Tangentensegmente(C, i)

for $j = i+1, \ldots, n$ **do**

  if (v_i, v_j) maximale Tangente **then**

    $k = j - 1$

    while $(v_i, v_j) \cap (v_{k-1}, v_k) = \emptyset$ **do**

      $k = k - 1$

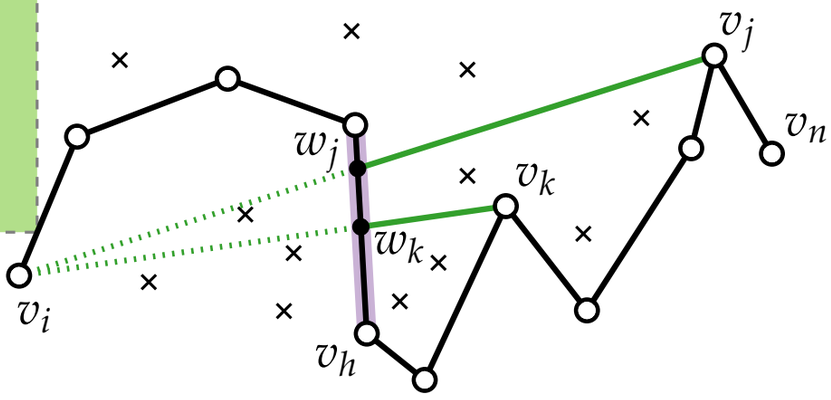
      if (v_i, v_k) maximale Tangente **then**

        $k = h$, wobei (v_{h-1}, v_h) Punkt w_k enthält

    $w_j = (v_i, v_j) \cap (v_{k-1}, v_k)$

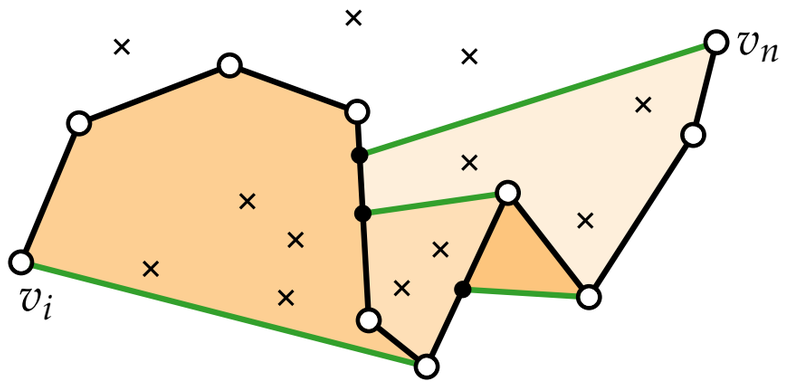
  else if (v_i, v_j) minimale Tangente **then**

    $\ldots$ `// analoges Vorgehen`



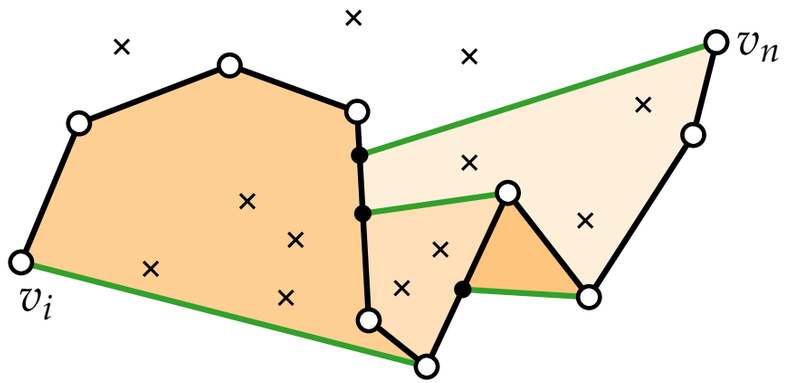
Laufzeit? Jeder Knoten wird beim Zurücklaufen max. 1 mal „übersprungen“ ➡ $\mathcal{O}(n)$

2) Punktzuweisung



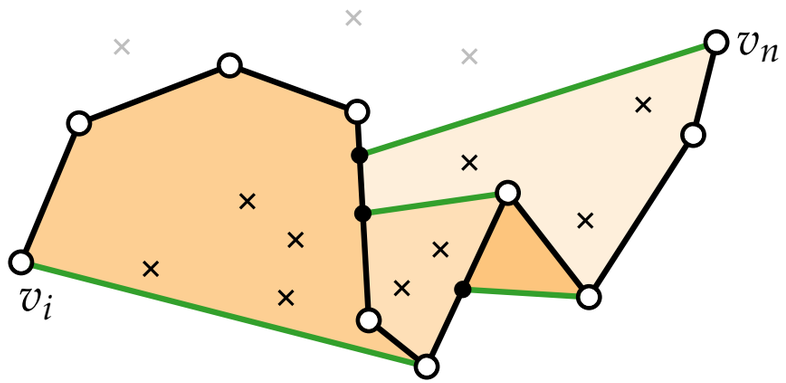
2) Punktzuweisung

- Punkte in äußerer Facette irrelevant



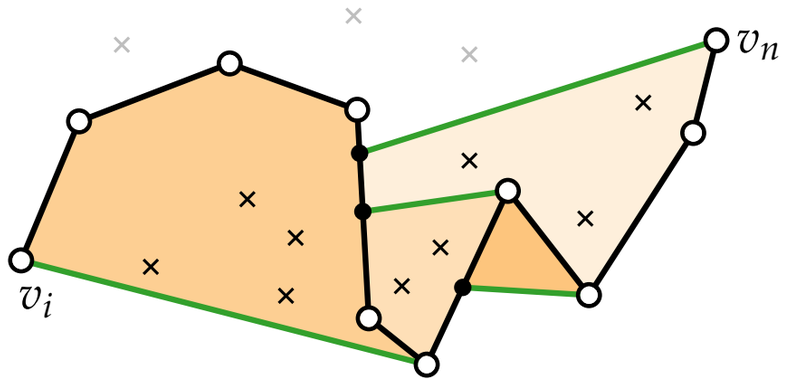
2) Punktzuweisung

- Punkte in äußerer Facette irrelevant



2) Punktzuweisung

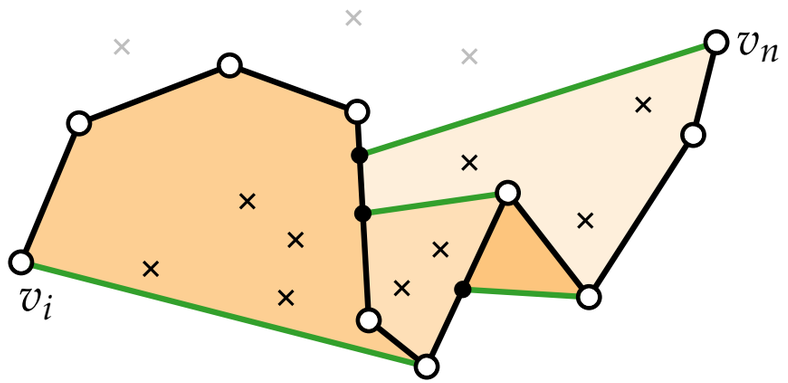
- Punkte in äußerer Facette irrelevant
- weise Punkte in P Facetten von S_i zu



2) Punktzuweisung

- Punkte in äußerer Facette irrelevant
- weise Punkte in P Facetten von S_i zu

Sind alle verbleibenden Punkte nötig?

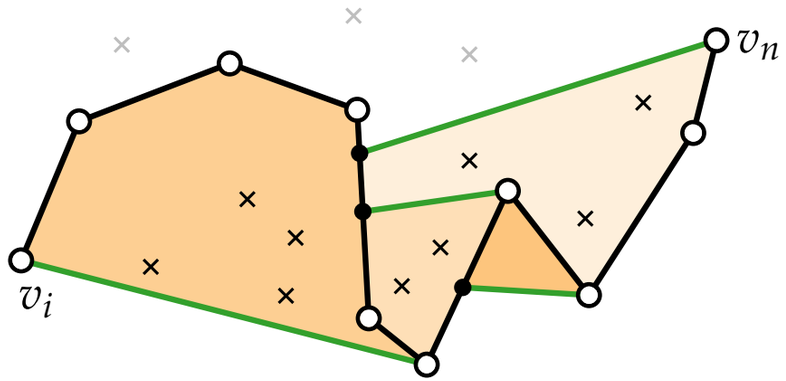


2) Punktzuweisung

- Punkte in äußerer Facette irrelevant
- weise Punkte in P Facetten von S_i zu

Sind alle verbleibenden Punkte nötig?

- für **minimale** Facette f, behalte Punkt p_f mit **maximaler** Steigung von $v_i p_f$

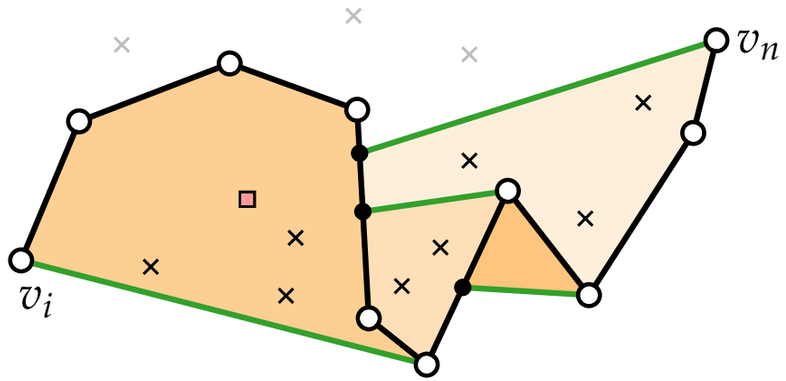


2) Punktzuweisung

- Punkte in äußerer Facette irrelevant
- weise Punkte in P Facetten von S_i zu

Sind alle verbleibenden Punkte nötig?

- für **minimale** Facette f, behalte Punkt p_f mit **maximaler** Steigung von $v_i p_f$

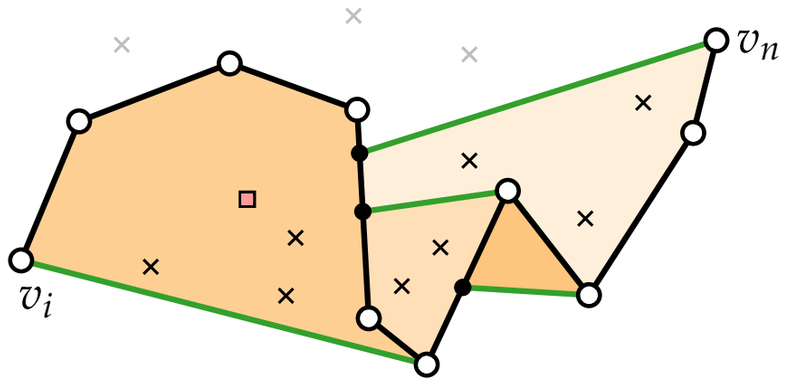


2) Punktzuweisung

- Punkte in äußerer Facette irrelevant
- weise Punkte in P Facetten von S_i zu

Sind alle verbleibenden Punkte nötig?

- für **minimale** Facette f, behalte Punkt p_f mit **maximaler** Steigung von $v_i p_f$
- für **maximale** Facette f, behalte Punkt p_f mit **minimaler** Steigung von $v_i p_f$

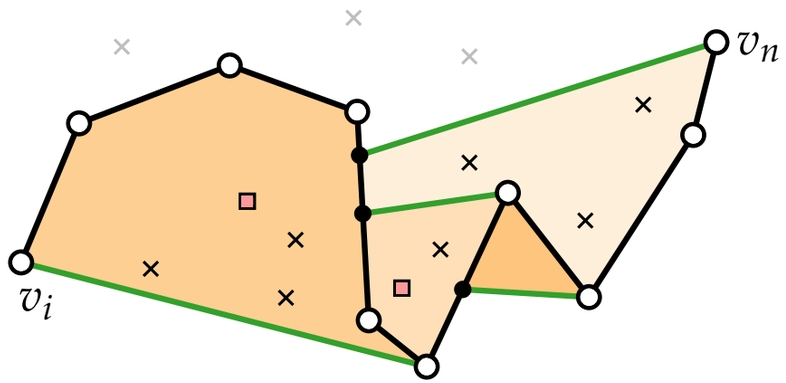


2) Punktzuweisung

- Punkte in äußerer Facette irrelevant
- weise Punkte in P Facetten von S_i zu

Sind alle verbleibenden Punkte nötig?

- für **minimale** Facette f, behalte Punkt p_f mit **maximaler** Steigung von $v_i p_f$
- für **maximale** Facette f, behalte Punkt p_f mit **minimaler** Steigung von $v_i p_f$

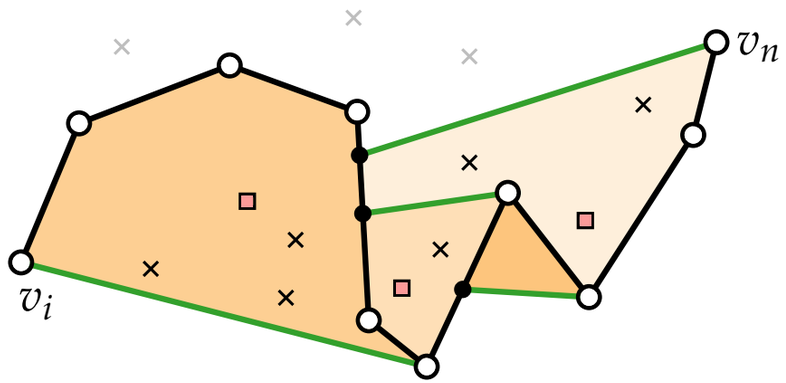


2) Punktzuweisung

- Punkte in äußerer Facette irrelevant
- weise Punkte in P Facetten von S_i zu

Sind alle verbleibenden Punkte nötig?

- für **minimale** Facette f, behalte Punkt p_f mit **maximaler** Steigung von $v_i p_f$
- für **maximale** Facette f, behalte Punkt p_f mit **minimaler** Steigung von $v_i p_f$



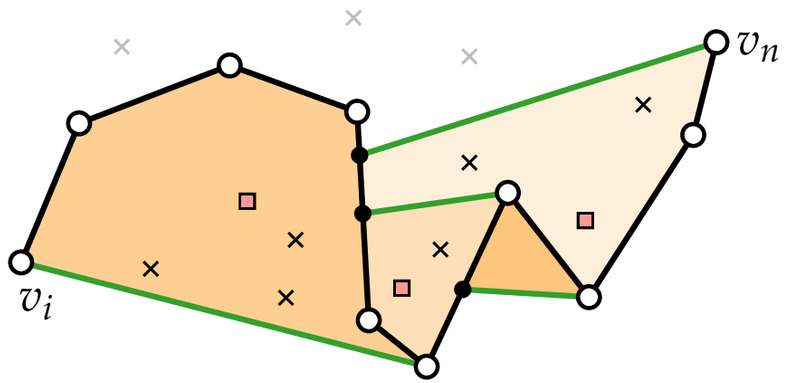
2) Punktzuweisung

- Punkte in äußerer Facette irrelevant
- weise Punkte in P Facetten von S_i zu

Sind alle verbleibenden Punkte nötig?

- für **minimale** Facette f, behalte Punkt p_f mit **maximaler** Steigung von $v_i p_f$
- für **maximale** Facette f, behalte Punkt p_f mit **minimaler** Steigung von $v_i p_f$

➡ verbleibende Punktmenge $P' \subseteq P$



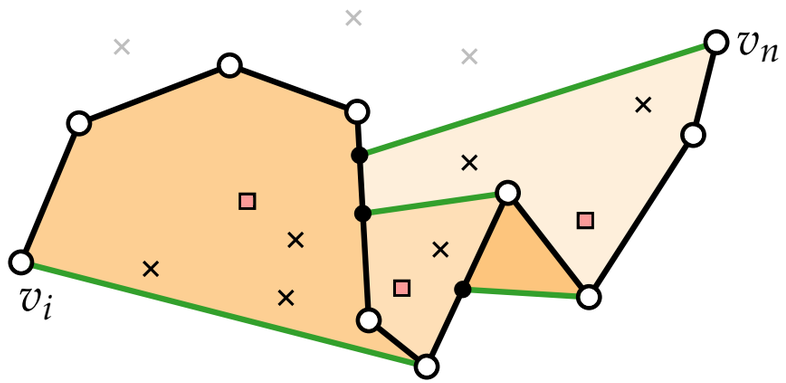
2) Punktzuweisung

- Punkte in äußerer Facette irrelevant
- weise Punkte in P Facetten von S_i zu

Sind alle verbleibenden Punkte nötig?

- für **minimale** Facette f, behalte Punkt p_f mit **maximaler** Steigung von $v_i p_f$
- für **maximale** Facette f, behalte Punkt p_f mit **minimaler** Steigung von $v_i p_f$

➡ verbleibende Punktmenge $P' \subseteq P$



Lemma 5. Eine Abkürzung (v_i, v_j) ist konsistent für bzgl. P $\Leftrightarrow$ sie ist konsistent bzgl. P'.

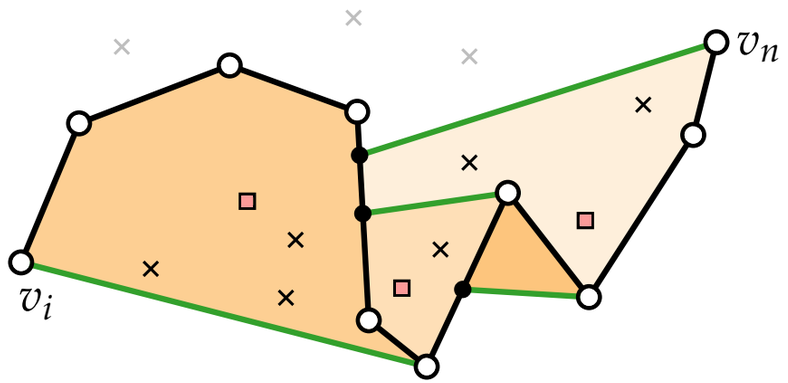
2) Punktzuweisung

- Punkte in äußerer Facette irrelevant
- weise Punkte in P Facetten von S_i zu

Sind alle verbleibenden Punkte nötig?

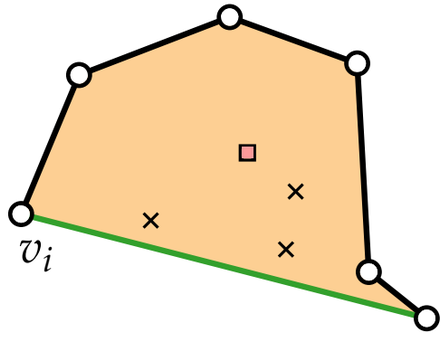
- für **minimale** Facette f, behalte Punkt p_f mit **maximaler** Steigung von $v_i p_f$
- für **maximale** Facette f, behalte Punkt p_f mit **minimaler** Steigung von $v_i p_f$

➡ verbleibende Punktmenge $P' \subseteq P$



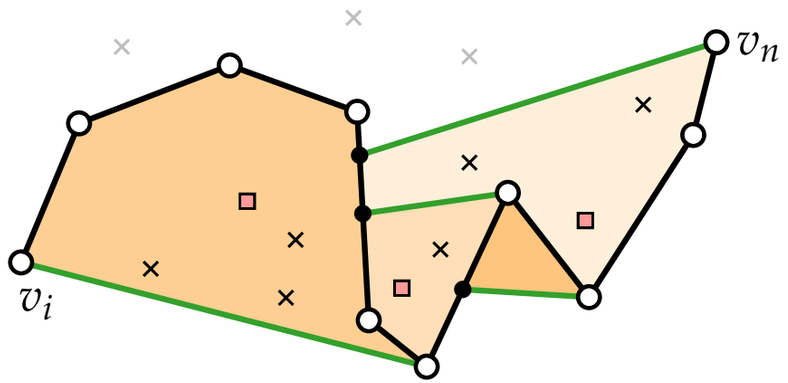
Lemma 5. Eine Abkürzung (v_i, v_j) ist konsistent für bzgl. P $\Leftrightarrow$ sie ist konsistent bzgl. P'.

(v_i, v_j) nicht konsistent bzgl. P



2) Punktzuweisung

- Punkte in äußerer Facette irrelevant
- weise Punkte in P Facetten von S_i zu

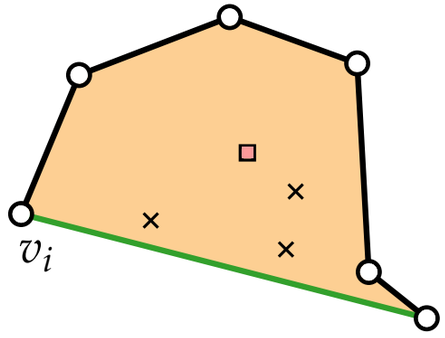


Sind alle verbleibenden Punkte nötig?

- für **minimale** Facette f, behalte Punkt p_f mit **maximaler** Steigung von $v_i p_f$
- für **maximale** Facette f, behalte Punkt p_f mit **minimaler** Steigung von $v_i p_f$

➡ verbleibende Punktmenge $P' \subseteq P$

Lemma 5. Eine Abkürzung (v_i, v_j) ist konsistent für bzgl. P $\Leftrightarrow$ sie ist konsistent bzgl. P'.

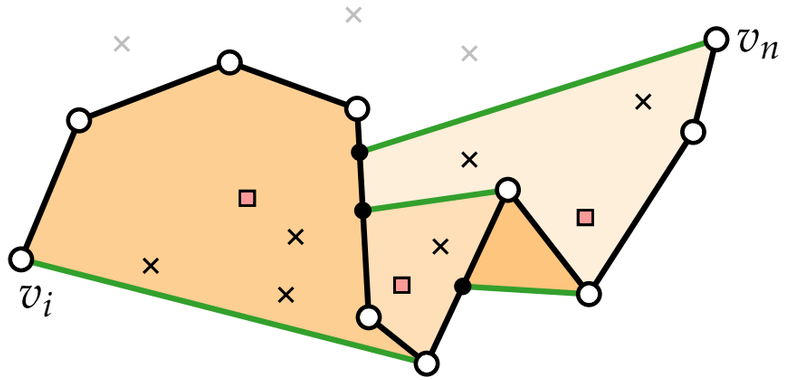


(v_i, v_j) nicht konsistent bzgl. P
$\Leftrightarrow$ ein Punkt $p \in P$ auf einer Facette f liegt zwischen C_{ij} und (v_i, v_j)

2) Punktzuweisung

- Punkte in äußerer Facette irrelevant
- weise Punkte in P Facetten von S_i zu

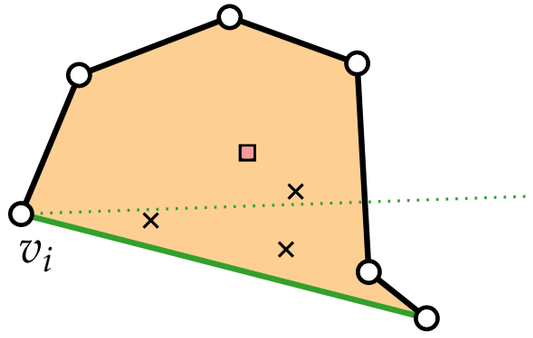
Sind alle verbleibenden Punkte nötig?



- für **minimale** Facette f, behalte Punkt p_f mit **maximaler** Steigung von $v_i p_f$
- für **maximale** Facette f, behalte Punkt p_f mit **minimaler** Steigung von $v_i p_f$

➡ verbleibende Punktmenge $P' \subseteq P$

Lemma 5. Eine Abkürzung (v_i, v_j) ist konsistent für bzgl. P $\Leftrightarrow$ sie ist konsistent bzgl. P'.

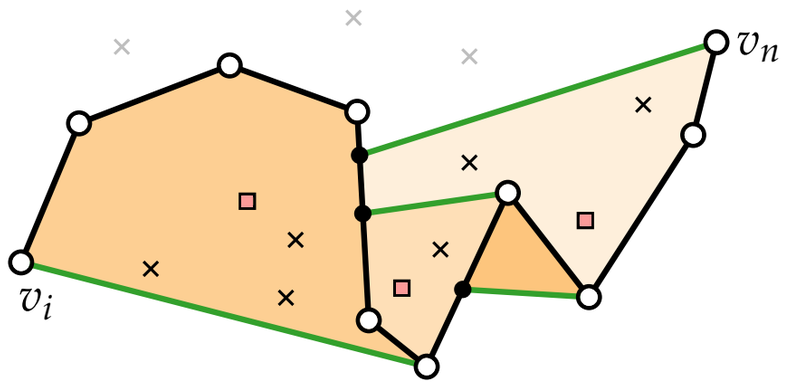


(v_i, v_j) nicht konsistent bzgl. P
$\Leftrightarrow$ ein Punkt $p \in P$ auf einer Facette f liegt zwischen C_{ij} und (v_i, v_j)

2) Punktzuweisung

- Punkte in äußerer Facette irrelevant
- weise Punkte in P Facetten von S_i zu

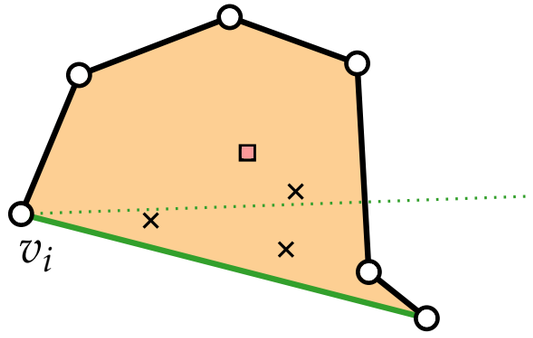
Sind alle verbleibenden Punkte nötig?



- für **minimale** Facette f, behalte Punkt p_f mit **maximaler** Steigung von $v_i p_f$
- für **maximale** Facette f, behalte Punkt p_f mit **minimaler** Steigung von $v_i p_f$

➡ verbleibende Punktmenge $P' \subseteq P$

Lemma 5. Eine Abkürzung (v_i, v_j) ist konsistent für bzgl. P $\Leftrightarrow$ sie ist konsistent bzgl. P'.



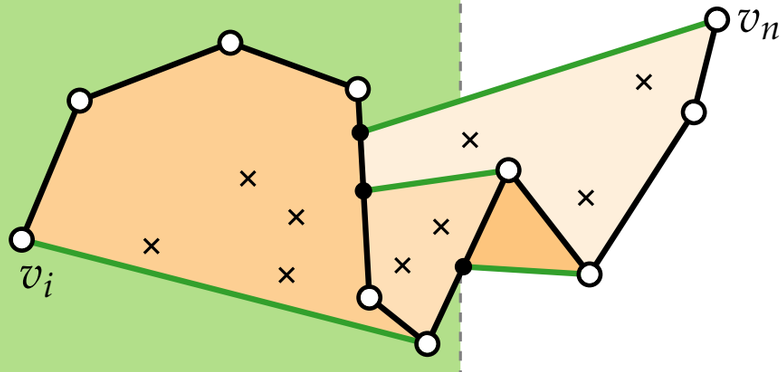
(v_i, v_j) nicht konsistent bzgl. P

$\Leftrightarrow$ ein Punkt $p \in P$ auf einer Facette f liegt zwischen C_{ij} und (v_i, v_j)

$\Leftrightarrow$ p_f liegt zwischen C_{ij} und (v_i, v_j)

2) Punktzuweisung: Algorithmus

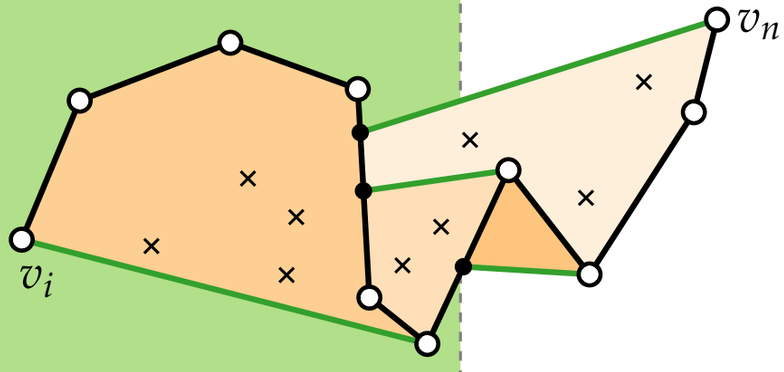
PUNKTZUWEISUNG(S_i, i, P)



2) Punktzuweisung: Algorithmus

PUNKTZUWEISUNG(S_i, i, P)

$P' =$ Punkte aus P rechts von v_i

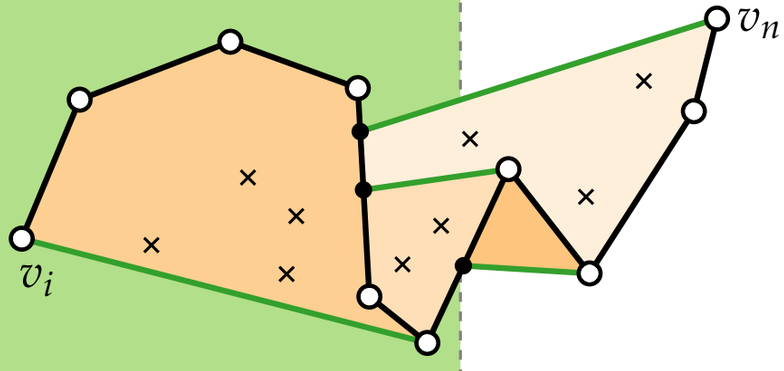


2) Punktzuweisung: Algorithmus

PUNKTZUWEISUNG(S_i, i, P)

$P' =$ Punkte aus P rechts von v_i

$Q =$ PRIORITYQUEUE($P' \cup \{v_{i+1}, \ldots, v_n\}$) sortiert nach Winkel bzgl. v_i



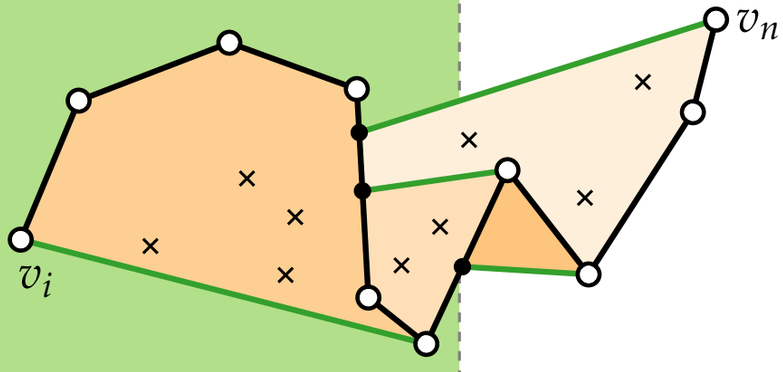
2) Punktzuweisung: Algorithmus

PUNKTZUWEISUNG(S_i, i, P)

$P' =$ Punkte aus P rechts von v_i

$Q =$ PRIORITYQUEUE($P' \cup \{v_{i+1}, \ldots, v_n\}$) sortiert nach Winkel bzgl. v_i

$T =$ balancierter binärer Suchbaum für radialen Sweep



2) Punktzuweisung: Algorithmus

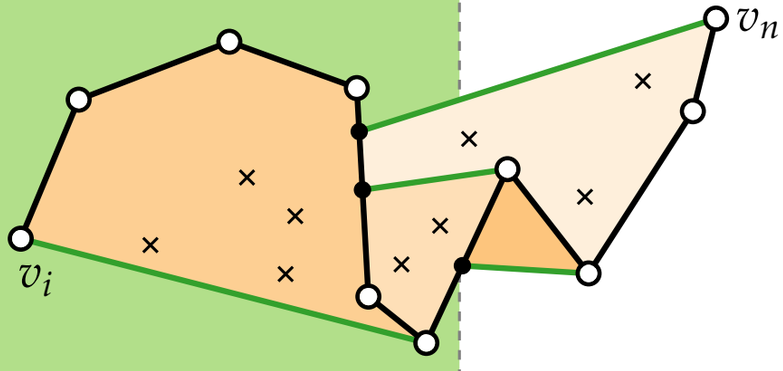
PUNKTZUWEISUNG(S_i, i, P)

$P' =$ Punkte aus P rechts von v_i

$Q =$ PRIORITYQUEUE($P' \cup \{v_{i+1}, \ldots, v_n\}$) sortiert nach Winkel bzgl. v_i

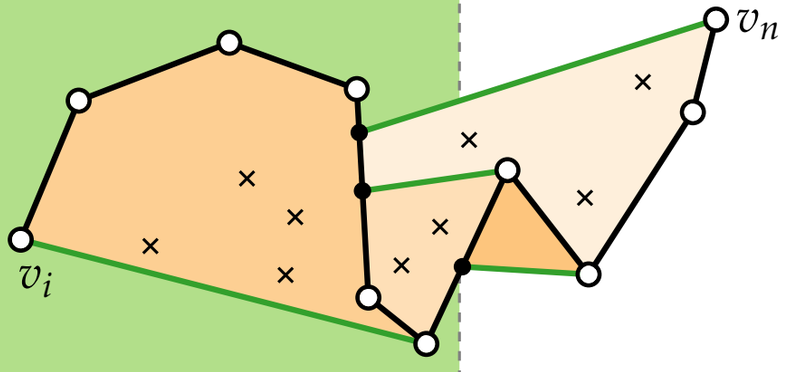
$T =$ balancierter binärer Suchbaum für radialen Sweep

while !Q.ISEMPTY() **do**



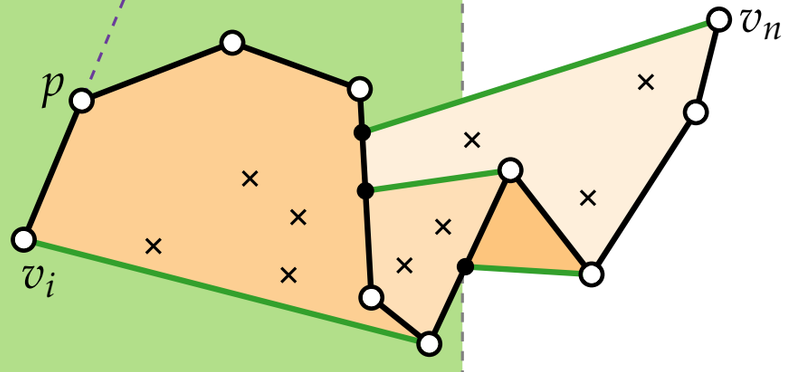
2) Punktzuweisung: Algorithmus

PUNKTZUWEISUNG(S_i, i, P)
  $P' =$ Punkte aus P rechts von v_i
  $Q =$ PRIORITYQUEUE($P' \cup \{v_{i+1}, \ldots, v_n\}$) sortiert nach Winkel bzgl. v_i
  $T =$ balancierter binärer Suchbaum für radialen Sweep
  while !Q.ISEMPTY() **do**
    $p = Q$.EXTRACTMAX()



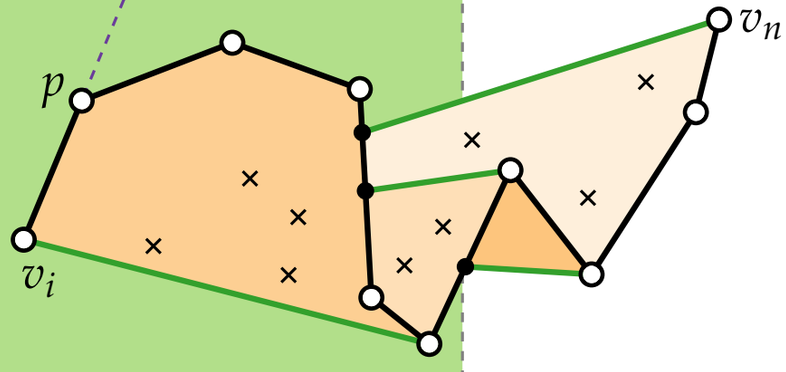
2) Punktzuweisung: Algorithmus

PUNKTZUWEISUNG(S_i, i, P)
  $P' =$ Punkte aus P rechts von v_i
  $Q =$ PRIORITYQUEUE($P' \cup \{v_{i+1}, \ldots, v_n\}$) sortiert nach Winkel bzgl. v_i
  $T =$ balancierter binärer Suchbaum für radialen Sweep
  while !Q.ISEMPTY() **do**
    $p = Q$.EXTRACTMAX()



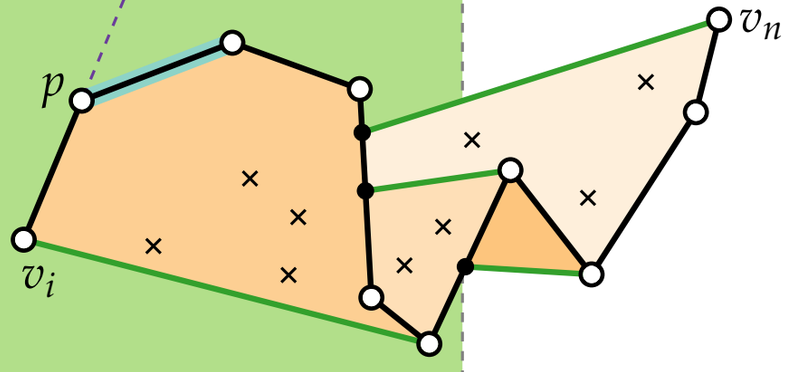
2) Punktzuweisung: Algorithmus

PUNKTZUWEISUNG(S_i, i, P)
  $P' =$ Punkte aus P rechts von v_i
  $Q =$ PRIORITYQUEUE($P' \cup \{v_{i+1}, \ldots, v_n\}$) sortiert nach Winkel bzgl. v_i
  $T =$ balancierter binärer Suchbaum für radialen Sweep
  while !Q.ISEMPTY() **do**
    $p = Q$.EXTRACTMAX()
    if p Knoten von C_{in} **then**
      update T mit Kanten von p
    else



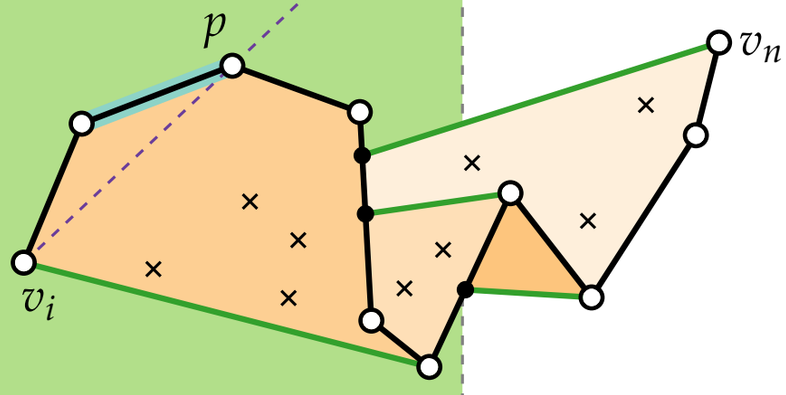
2) Punktzuweisung: Algorithmus

PUNKTZUWEISUNG(S_i, i, P)
  $P' =$ Punkte aus P rechts von v_i
  $Q =$ PRIORITYQUEUE($P' \cup \{v_{i+1}, \ldots, v_n\}$) sortiert nach Winkel bzgl. v_i
  $T =$ balancierter binärer Suchbaum für radialen Sweep
  while !Q.ISEMPTY() **do**
    $p = Q$.EXTRACTMAX()
    if p Knoten von C_{in} **then**
      update T mit Kanten von p
    else



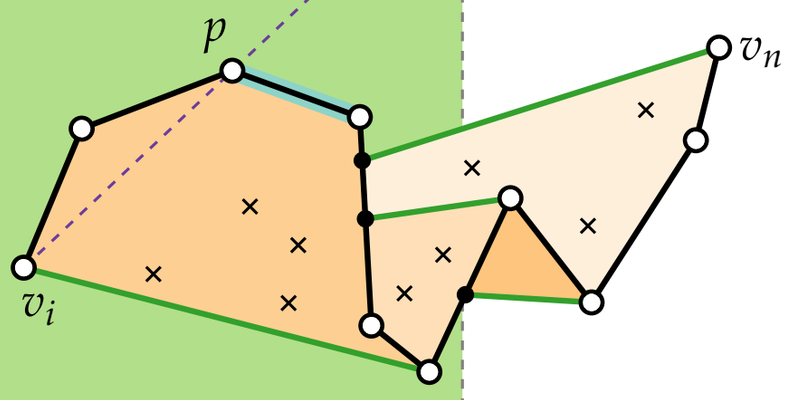
2) Punktzuweisung: Algorithmus

PUNKTZUWEISUNG(S_i, i, P)
  $P' =$ Punkte aus P rechts von v_i
  $Q =$ PRIORITYQUEUE($P' \cup \{v_{i+1}, \ldots, v_n\}$) sortiert nach Winkel bzgl. v_i
  $T =$ balancierter binärer Suchbaum für radialen Sweep
  while !Q.ISEMPTY() **do**
    $p = Q$.EXTRACTMAX()
    if p Knoten von C_{in} **then**
      update T mit Kanten von p
    else



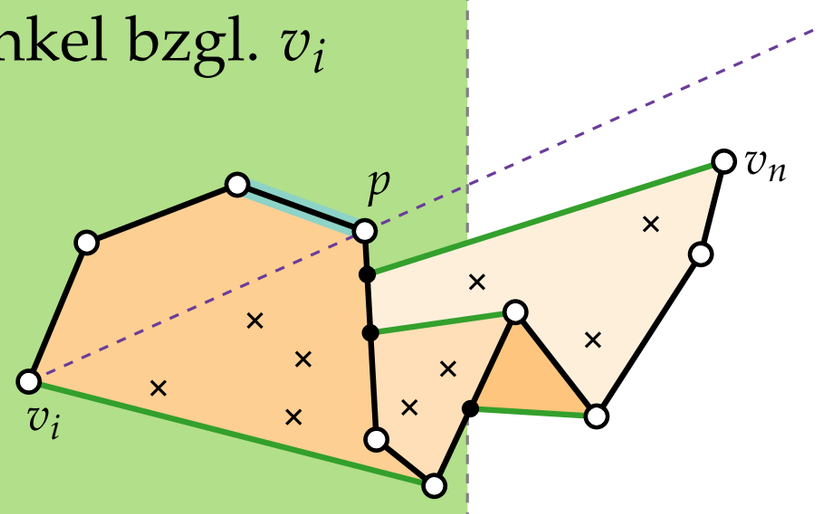
2) Punktzuweisung: Algorithmus

PUNKTZUWEISUNG(S_i, i, P)
  $P' =$ Punkte aus P rechts von v_i
  $Q =$ PRIORITYQUEUE($P' \cup \{v_{i+1}, \ldots, v_n\}$) sortiert nach Winkel bzgl. v_i
  $T =$ balancierter binärer Suchbaum für radialen Sweep
  while !Q.ISEMPTY() **do**
    $p = Q$.EXTRACTMAX()
    if p Knoten von C_{in} **then**
      update T mit Kanten von p
    else



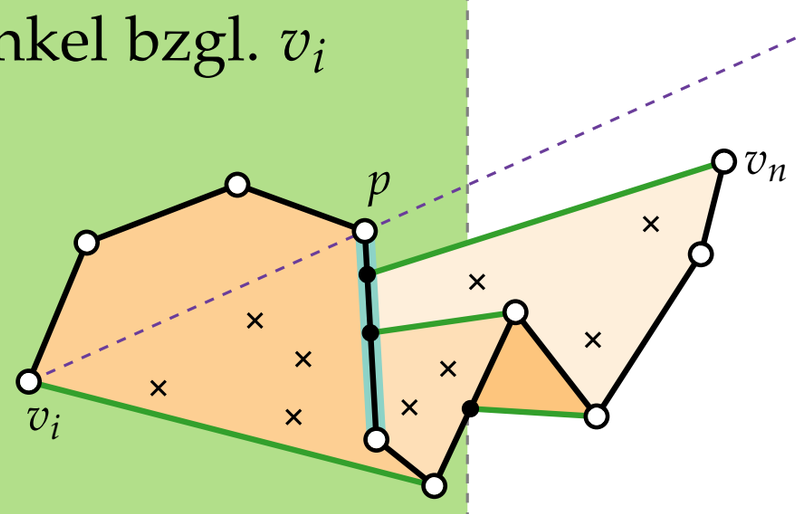
2) Punktzuweisung: Algorithmus

PUNKTZUWEISUNG(S_i, i, P)
  $P' =$ Punkte aus P rechts von v_i
  $Q =$ PRIORITYQUEUE($P' \cup \{v_{i+1}, \ldots, v_n\}$) sortiert nach Winkel bzgl. v_i
  $T =$ balancierter binärer Suchbaum für radialen Sweep
  while !Q.ISEMPTY() **do**
    $p = Q$.EXTRACTMAX()
    if p Knoten von C_{in} **then**
      update T mit Kanten von p
    else



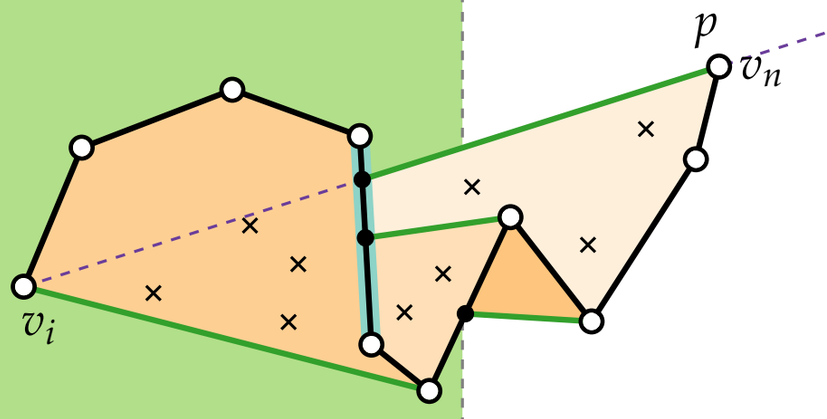
2) Punktzuweisung: Algorithmus

PUNKTZUWEISUNG(S_i, i, P)
  $P' =$ Punkte aus P rechts von v_i
  $Q =$ PRIORITYQUEUE($P' \cup \{v_{i+1}, \ldots, v_n\}$) sortiert nach Winkel bzgl. v_i
  $T =$ balancierter binärer Suchbaum für radialen Sweep
  while !Q.ISEMPTY() **do**
    $p = Q$.EXTRACTMAX()
    if p Knoten von C_{in} **then**
      update T mit Kanten von p
    else



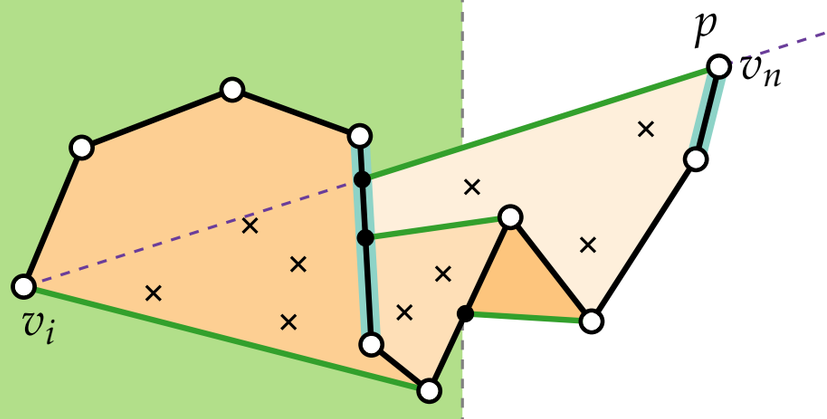
2) Punktzuweisung: Algorithmus

PUNKTZUWEISUNG(S_i, i, P)
  $P' =$ Punkte aus P rechts von v_i
  $Q =$ PRIORITYQUEUE($P' \cup \{v_{i+1}, \ldots, v_n\}$) sortiert nach Winkel bzgl. v_i
  $T =$ balancierter binärer Suchbaum für radialen Sweep
  while !Q.ISEMPTY() **do**
    $p = Q$.EXTRACTMAX()
    if p Knoten von C_{in} **then**
      update T mit Kanten von p
    else



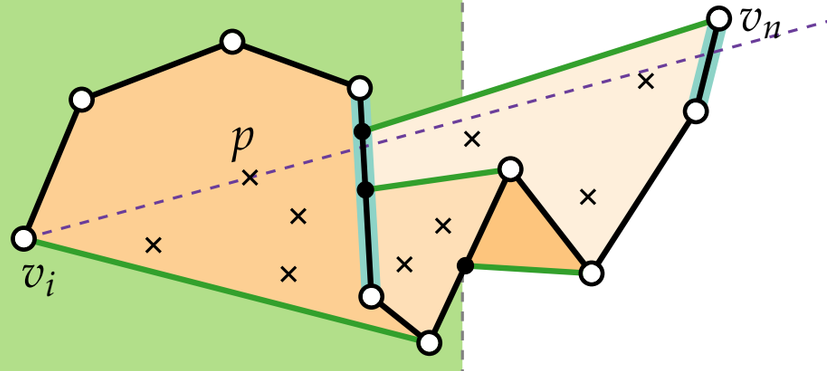
2) Punktzuweisung: Algorithmus

PUNKTZUWEISUNG(S_i, i, P)
  $P' =$ Punkte aus P rechts von v_i
  $Q =$ PRIORITYQUEUE($P' \cup \{v_{i+1}, \ldots, v_n\}$) sortiert nach Winkel bzgl. v_i
  $T =$ balancierter binärer Suchbaum für radialen Sweep
  while !Q.ISEMPTY() **do**
    $p = Q$.EXTRACTMAX()
    if p Knoten von C_{in} **then**
      update T mit Kanten von p
    else



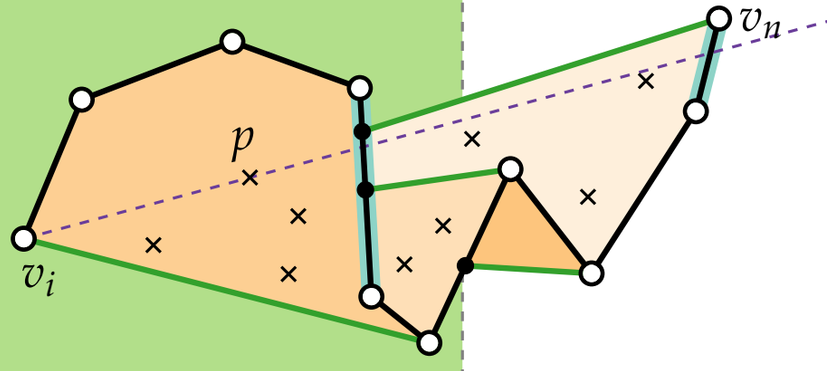
2) Punktzuweisung: Algorithmus

PUNKTZUWEISUNG(S_i, i, P)
  $P' =$ Punkte aus P rechts von v_i
  $Q =$ PRIORITYQUEUE($P' \cup \{v_{i+1}, \ldots, v_n\}$) sortiert nach Winkel bzgl. v_i
  $T =$ balancierter binärer Suchbaum für radialen Sweep
  while !Q.ISEMPTY() **do**
    $p = Q$.EXTRACTMAX()
    if p Knoten von C_{in} **then**
      update T mit Kanten von p
    else



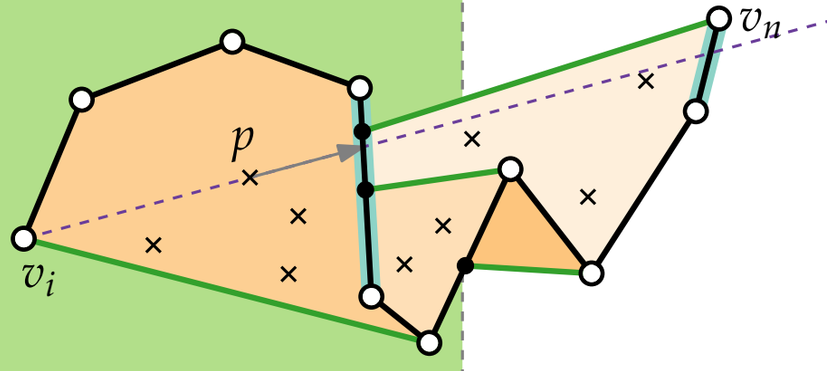
2) Punktzuweisung: Algorithmus

```
PUNKTZUWEISUNG(S_i, i, P)
    P' = Punkte aus P rechts von v_i
    Q = PRIORITYQUEUE(P' ∪ {v_{i+1}, ..., v_n}) sortiert nach Winkel bzgl. v_i
    T = balancierter binärer Suchbaum für radialen Sweep
    while !Q.ISEMPTY() do
        p = Q.EXTRACTMAX()
        if p Knoten von C_in then
            update T mit Kanten von p
        else
            e = linkeste Kante v_k v_{k+1} in T rechts von p auf Sweep-Line
            speichere p an e
```



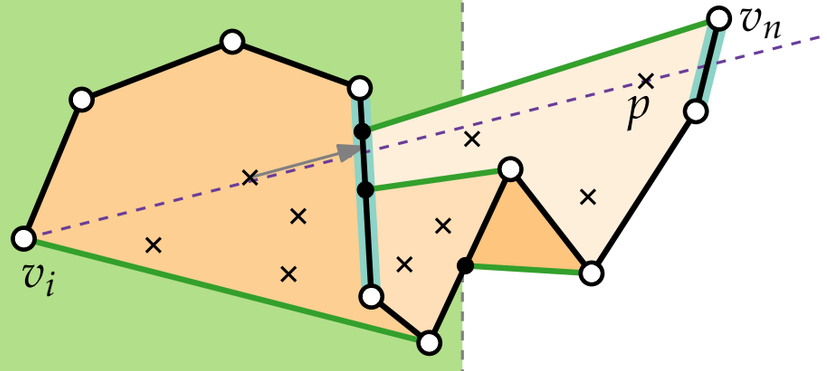
2) Punktzuweisung: Algorithmus

```
PUNKTZUWEISUNG(S_i, i, P)
    P' = Punkte aus P rechts von v_i
    Q = PRIORITYQUEUE(P' ∪ {v_{i+1}, ..., v_n}) sortiert nach Winkel bzgl. v_i
    T = balancierter binärer Suchbaum für radialen Sweep
    while !Q.ISEMPTY() do
        p = Q.EXTRACTMAX()
        if p Knoten von C_in then
            update T mit Kanten von p
        else
            e = linkeste Kante v_k v_{k+1} in T rechts von p auf Sweep-Line
            speichere p an e
```



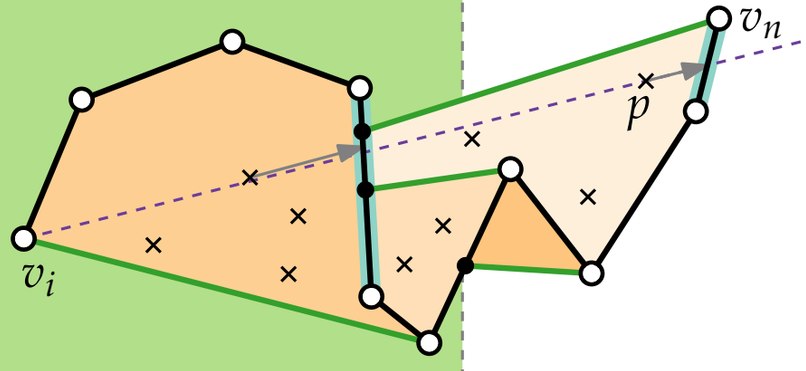
2) Punktzuweisung: Algorithmus

PUNKTZUWEISUNG(S_i, i, P)
  $P' =$ Punkte aus P rechts von v_i
  $Q =$ PRIORITYQUEUE($P' \cup \{v_{i+1}, \ldots, v_n\}$) sortiert nach Winkel bzgl. v_i
  $T =$ balancierter binärer Suchbaum für radialen Sweep
  while !Q.ISEMPTY() **do**
    $p = Q$.EXTRACTMAX()
    if p Knoten von C_{in} **then**
      update T mit Kanten von p
    else
      $e =$ linkeste Kante $v_k v_{k+1}$ in T rechts von p auf Sweep-Line
      speichere p an e



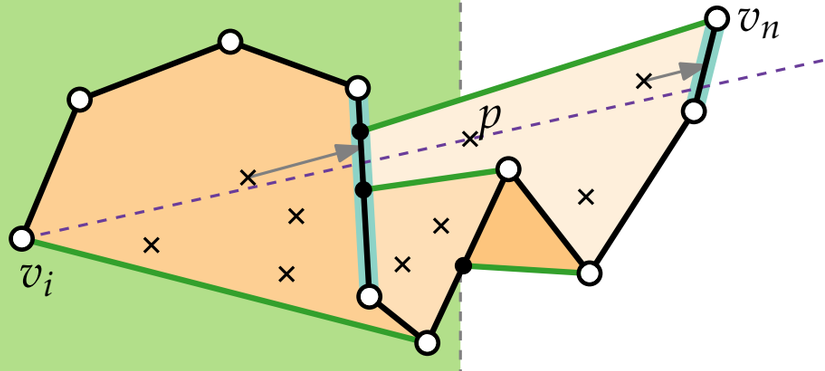
2) Punktzuweisung: Algorithmus

PUNKTZUWEISUNG(S_i, i, P)
  $P' =$ Punkte aus P rechts von v_i
  $Q =$ PRIORITYQUEUE($P' \cup \{v_{i+1}, \ldots, v_n\}$) sortiert nach Winkel bzgl. v_i
  $T =$ balancierter binärer Suchbaum für radialen Sweep
  while !Q.ISEMPTY() **do**
    $p = Q$.EXTRACTMAX()
    if p Knoten von C_{in} **then**
      update T mit Kanten von p
    else
      $e =$ linkeste Kante $v_k v_{k+1}$ in T rechts von p auf Sweep-Line
      speichere p an e



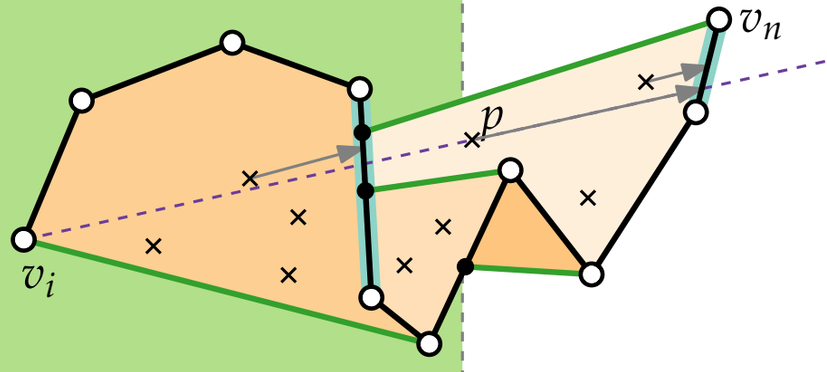
2) Punktzuweisung: Algorithmus

PUNKTZUWEISUNG(S_i, i, P)
  $P' =$ Punkte aus P rechts von v_i
  $Q =$ PRIORITYQUEUE($P' \cup \{v_{i+1}, \ldots, v_n\}$) sortiert nach Winkel bzgl. v_i
  $T =$ balancierter binärer Suchbaum für radialen Sweep
  while !Q.ISEMPTY() **do**
    $p = Q$.EXTRACTMAX()
    if p Knoten von C_{in} **then**
      update T mit Kanten von p
    else
      $e =$ linkeste Kante $v_k v_{k+1}$ in T rechts von p auf Sweep-Line
      speichere p an e



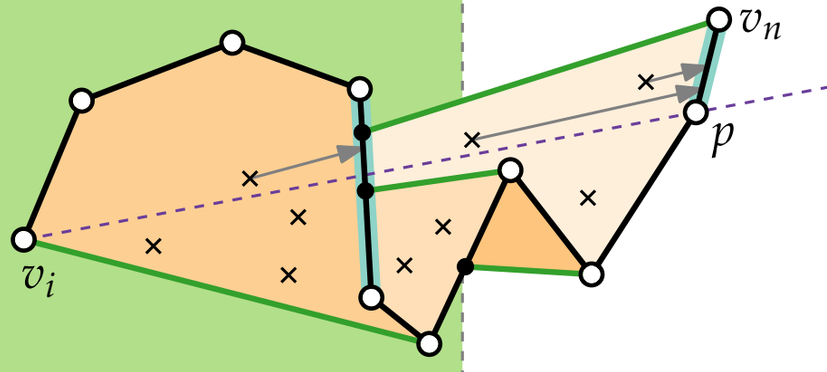
2) Punktzuweisung: Algorithmus

PUNKTZUWEISUNG(S_i, i, P)
  $P' =$ Punkte aus P rechts von v_i
  $Q =$ PRIORITYQUEUE($P' \cup \{v_{i+1}, \ldots, v_n\}$) sortiert nach Winkel bzgl. v_i
  $T =$ balancierter binärer Suchbaum für radialen Sweep
  while !Q.ISEMPTY() **do**
    $p = Q$.EXTRACTMAX()
    if p Knoten von C_{in} **then**
      update T mit Kanten von p
    else
      $e =$ linkeste Kante $v_k v_{k+1}$ in T rechts von p auf Sweep-Line
      speichere p an e



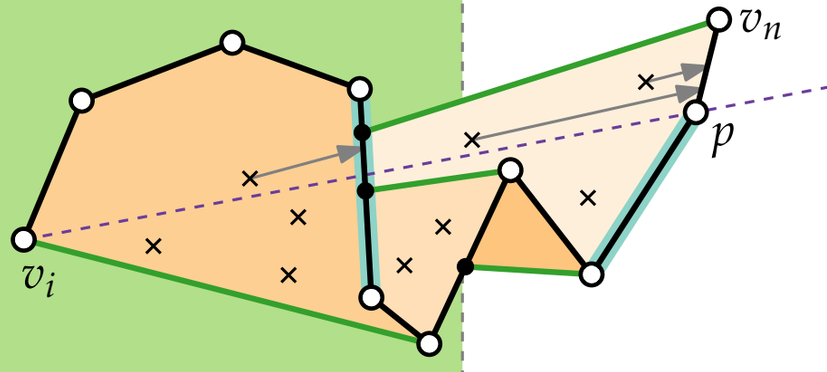
2) Punktzuweisung: Algorithmus

PUNKTZUWEISUNG(S_i, i, P)
  $P' =$ Punkte aus P rechts von v_i
  $Q =$ PRIORITYQUEUE($P' \cup \{v_{i+1}, \ldots, v_n\}$) sortiert nach Winkel bzgl. v_i
  $T =$ balancierter binärer Suchbaum für radialen Sweep
  while !Q.ISEMPTY() **do**
    $p = Q$.EXTRACTMAX()
    if p Knoten von C_{in} **then**
      update T mit Kanten von p
    else
      $e =$ linkeste Kante $v_k v_{k+1}$ in T rechts von p auf Sweep-Line
      speichere p an e



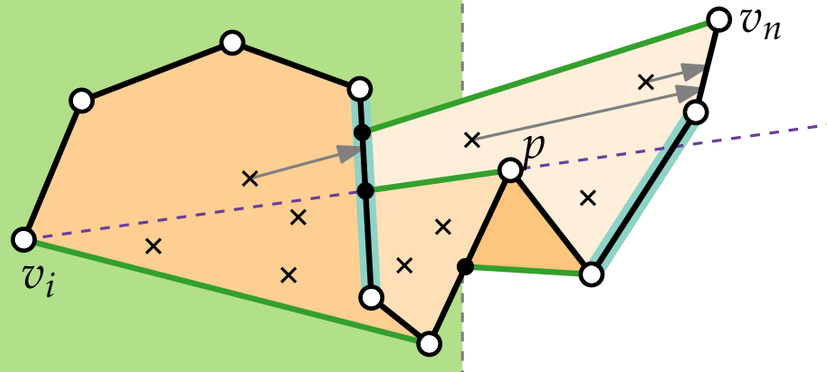
2) Punktzuweisung: Algorithmus

```
PUNKTZUWEISUNG(S_i, i, P)
    P' = Punkte aus P rechts von v_i
    Q = PRIORITYQUEUE(P' ∪ {v_{i+1}, ..., v_n}) sortiert nach Winkel bzgl. v_i
    T = balancierter binärer Suchbaum für radialen Sweep
    while !Q.ISEMPTY() do
        p = Q.EXTRACTMAX()
        if p Knoten von C_in then
            update T mit Kanten von p
        else
            e = linkeste Kante v_k v_{k+1} in T rechts von p auf Sweep-Line
            speichere p an e
```



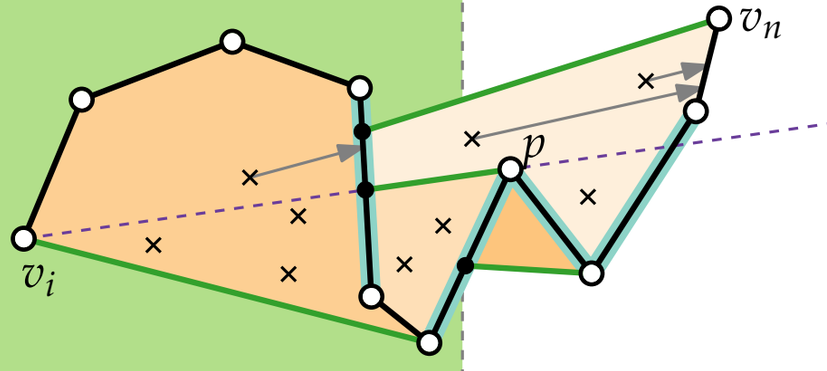
2) Punktzuweisung: Algorithmus

PUNKTZUWEISUNG(S_i, i, P)
  $P' =$ Punkte aus P rechts von v_i
  $Q =$ PRIORITYQUEUE($P' \cup \{v_{i+1}, \ldots, v_n\}$) sortiert nach Winkel bzgl. v_i
  $T =$ balancierter binärer Suchbaum für radialen Sweep
  while !Q.ISEMPTY() **do**
    $p = Q$.EXTRACTMAX()
    if p Knoten von C_{in} **then**
      update T mit Kanten von p
    else
      $e =$ linkeste Kante $v_k v_{k+1}$ in T rechts von p auf Sweep-Line
      speichere p an e



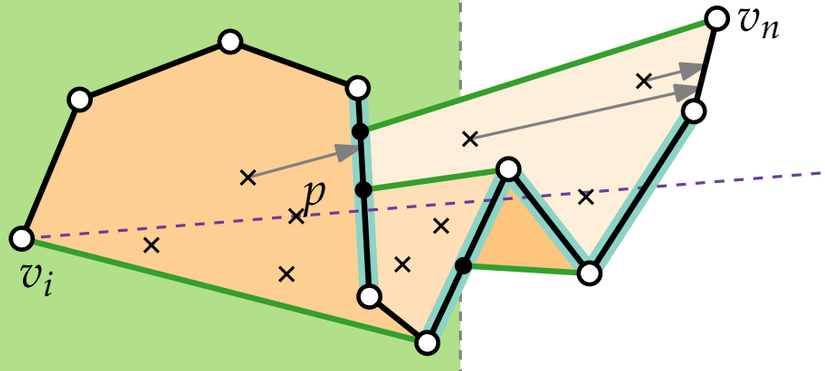
2) Punktzuweisung: Algorithmus

PUNKTZUWEISUNG(S_i, i, P)
  $P' =$ Punkte aus P rechts von v_i
  $Q =$ PRIORITYQUEUE($P' \cup \{v_{i+1}, \ldots, v_n\}$) sortiert nach Winkel bzgl. v_i
  $T =$ balancierter binärer Suchbaum für radialen Sweep
  while !Q.ISEMPTY() **do**
    $p = Q$.EXTRACTMAX()
    if p Knoten von C_{in} **then**
      update T mit Kanten von p
    else
      $e =$ linkeste Kante $v_k v_{k+1}$ in T rechts von p auf Sweep-Line
      speichere p an e



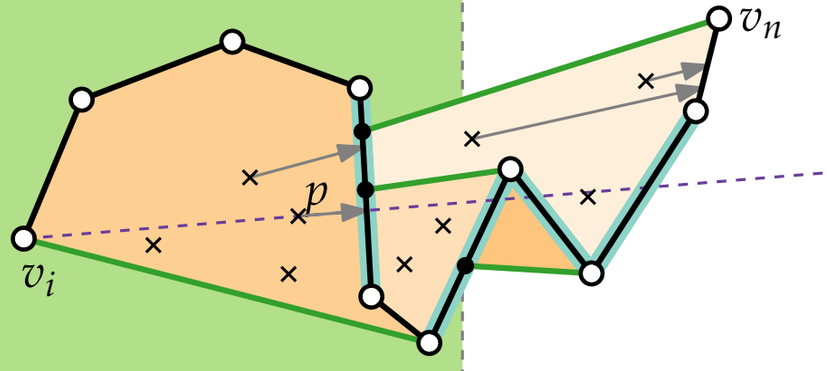
2) Punktzuweisung: Algorithmus

PUNKTZUWEISUNG(S_i, i, P)
  $P' =$ Punkte aus P rechts von v_i
  $Q =$ PRIORITYQUEUE($P' \cup \{v_{i+1}, \ldots, v_n\}$) sortiert nach Winkel bzgl. v_i
  $T =$ balancierter binärer Suchbaum für radialen Sweep
  while !Q.ISEMPTY() **do**
    $p = Q$.EXTRACTMAX()
    if p Knoten von C_{in} **then**
      update T mit Kanten von p
    else
      $e =$ linkeste Kante $v_k v_{k+1}$ in T rechts von p auf Sweep-Line
      speichere p an e



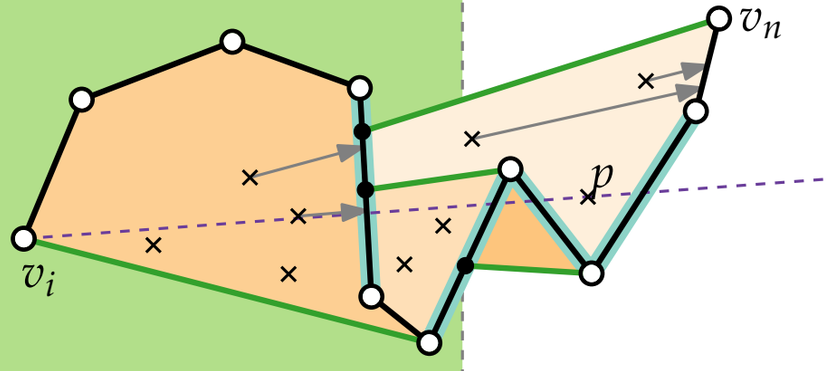
2) Punktzuweisung: Algorithmus

PUNKTZUWEISUNG(S_i, i, P)
  $P' =$ Punkte aus P rechts von v_i
  $Q =$ PRIORITYQUEUE($P' \cup \{v_{i+1}, \ldots, v_n\}$) sortiert nach Winkel bzgl. v_i
  $T =$ balancierter binärer Suchbaum für radialen Sweep
  while !Q.ISEMPTY() **do**
    $p = Q$.EXTRACTMAX()
    if p Knoten von C_{in} **then**
      update T mit Kanten von p
    else
      $e =$ linkeste Kante $v_k v_{k+1}$ in T rechts von p auf Sweep-Line
      speichere p an e



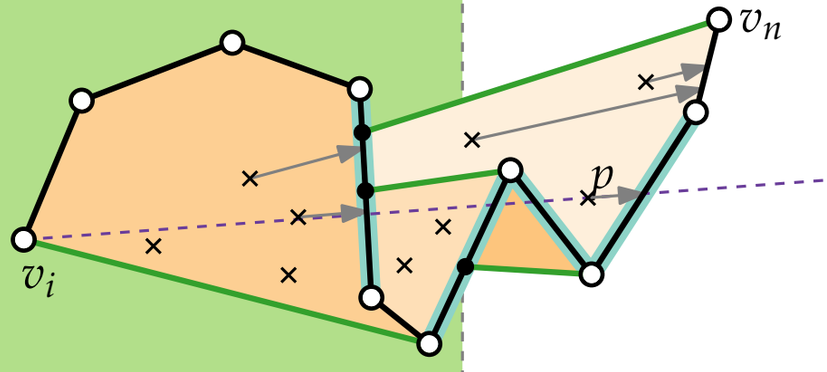
2) Punktzuweisung: Algorithmus

PUNKTZUWEISUNG(S_i, i, P)
  $P' =$ Punkte aus P rechts von v_i
  $Q =$ PRIORITYQUEUE($P' \cup \{v_{i+1}, \ldots, v_n\}$) sortiert nach Winkel bzgl. v_i
  $T =$ balancierter binärer Suchbaum für radialen Sweep
  while !Q.ISEMPTY() **do**
    $p = Q$.EXTRACTMAX()
    if p Knoten von C_{in} **then**
      update T mit Kanten von p
    else
      $e =$ linkeste Kante $v_k v_{k+1}$ in T rechts von p auf Sweep-Line
      speichere p an e



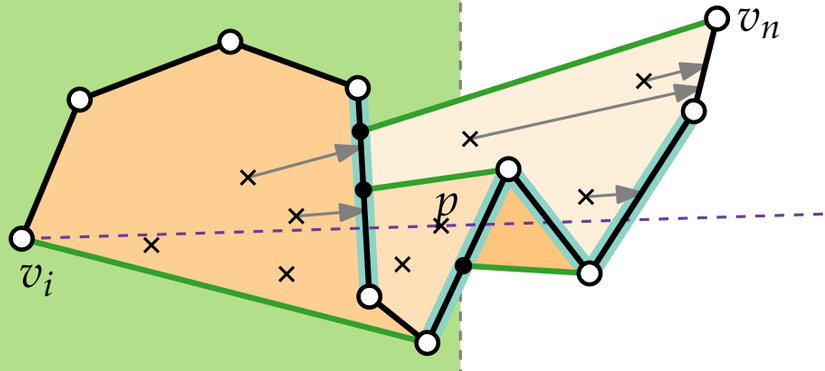
2) Punktzuweisung: Algorithmus

PUNKTZUWEISUNG(S_i, i, P)
  $P' =$ Punkte aus P rechts von v_i
  $Q =$ PRIORITYQUEUE($P' \cup \{v_{i+1}, \ldots, v_n\}$) sortiert nach Winkel bzgl. v_i
  $T =$ balancierter binärer Suchbaum für radialen Sweep
  while !Q.ISEMPTY() **do**
    $p = Q$.EXTRACTMAX()
    if p Knoten von C_{in} **then**
      update T mit Kanten von p
    else
      $e =$ linkeste Kante $v_k v_{k+1}$ in T rechts von p auf Sweep-Line
      speichere p an e



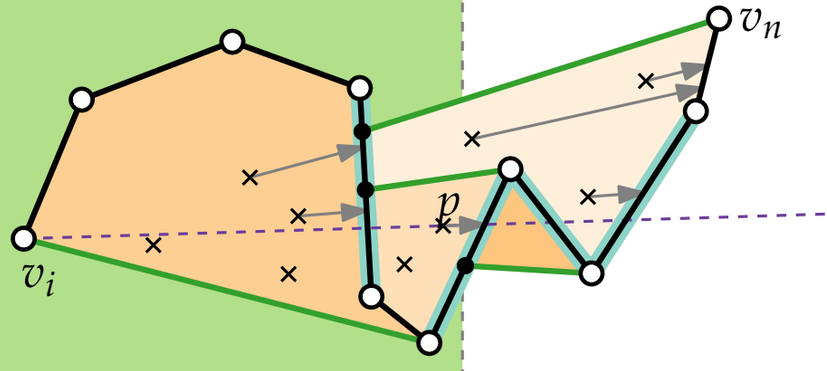
2) Punktzuweisung: Algorithmus

PUNKTZUWEISUNG(S_i, i, P)
  $P' =$ Punkte aus P rechts von v_i
  $Q =$ PRIORITYQUEUE($P' \cup \{v_{i+1}, \ldots, v_n\}$) sortiert nach Winkel bzgl. v_i
  $T =$ balancierter binärer Suchbaum für radialen Sweep
  while !Q.ISEMPTY() **do**
    $p = Q$.EXTRACTMAX()
    if p Knoten von C_{in} **then**
      update T mit Kanten von p
    else
      $e =$ linkeste Kante $v_k v_{k+1}$ in T rechts von p auf Sweep-Line
      speichere p an e



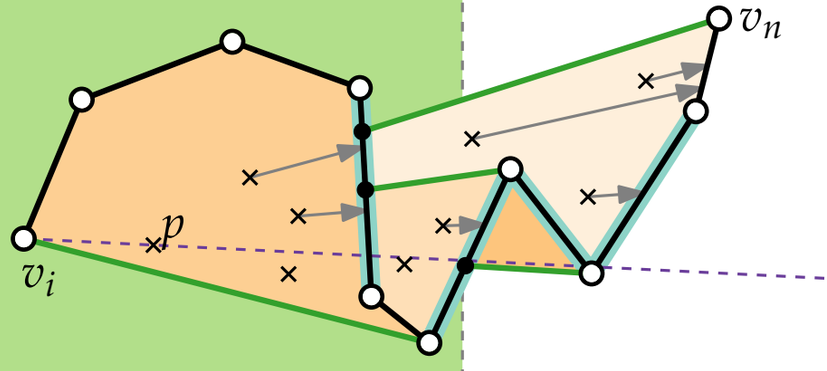
2) Punktzuweisung: Algorithmus

PUNKTZUWEISUNG(S_i, i, P)
  $P' =$ Punkte aus P rechts von v_i
  $Q =$ PRIORITYQUEUE($P' \cup \{v_{i+1}, \ldots, v_n\}$) sortiert nach Winkel bzgl. v_i
  $T =$ balancierter binärer Suchbaum für radialen Sweep
  while !Q.ISEMPTY() **do**
    $p = Q$.EXTRACTMAX()
    if p Knoten von C_{in} **then**
      update T mit Kanten von p
    else
      $e =$ linkeste Kante $v_k v_{k+1}$ in T rechts von p auf Sweep-Line
      speichere p an e



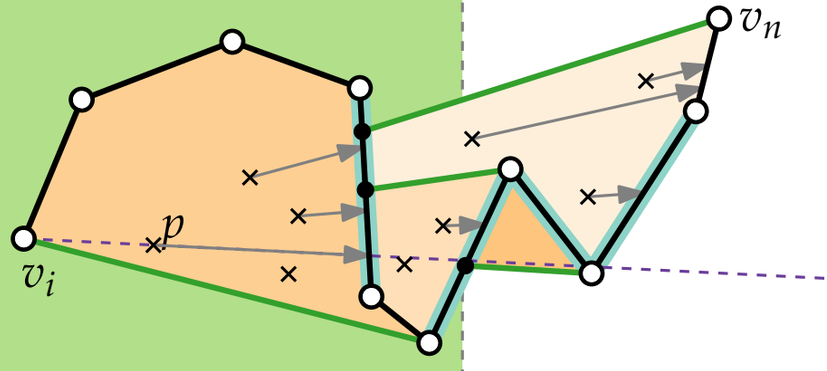
2) Punktzuweisung: Algorithmus

PUNKTZUWEISUNG(S_i, i, P)
  $P' =$ Punkte aus P rechts von v_i
  $Q =$ PRIORITYQUEUE($P' \cup \{v_{i+1}, \ldots, v_n\}$) sortiert nach Winkel bzgl. v_i
  $T =$ balancierter binärer Suchbaum für radialen Sweep
  while !Q.ISEMPTY() **do**
    $p = Q$.EXTRACTMAX()
    if p Knoten von C_{in} **then**
      update T mit Kanten von p
    else
      $e =$ linkeste Kante $v_k v_{k+1}$ in T rechts von p auf Sweep-Line
      speichere p an e



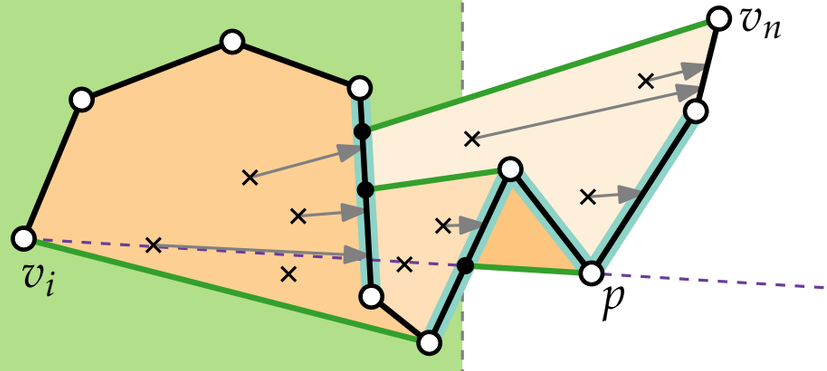
2) Punktzuweisung: Algorithmus

PUNKTZUWEISUNG(S_i, i, P)
  $P' =$ Punkte aus P rechts von v_i
  $Q =$ PRIORITYQUEUE($P' \cup \{v_{i+1}, \ldots, v_n\}$) sortiert nach Winkel bzgl. v_i
  $T =$ balancierter binärer Suchbaum für radialen Sweep
  while !Q.ISEMPTY() **do**
    $p = Q$.EXTRACTMAX()
    if p Knoten von C_{in} **then**
      update T mit Kanten von p
    else
      $e =$ linkeste Kante $v_k v_{k+1}$ in T rechts von p auf Sweep-Line
      speichere p an e



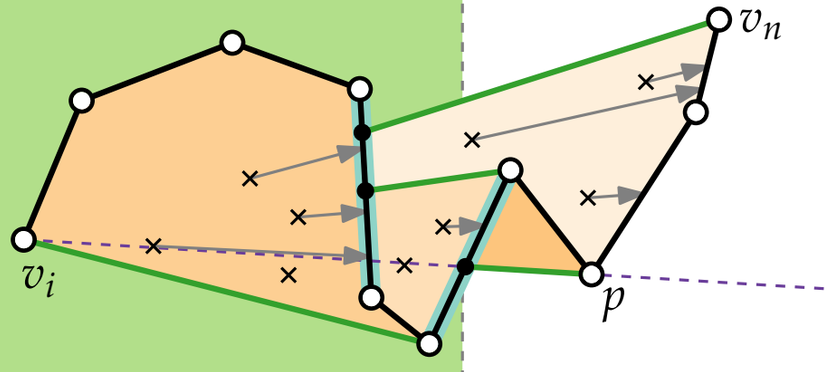
2) Punktzuweisung: Algorithmus

PUNKTZUWEISUNG(S_i, i, P)
  $P' =$ Punkte aus P rechts von v_i
  $Q =$ PRIORITYQUEUE($P' \cup \{v_{i+1}, \ldots, v_n\}$) sortiert nach Winkel bzgl. v_i
  $T =$ balancierter binärer Suchbaum für radialen Sweep
  while !Q.ISEMPTY() **do**
    $p = Q$.EXTRACTMAX()
    if p Knoten von C_{in} **then**
      update T mit Kanten von p
    else
      $e =$ linkeste Kante $v_k v_{k+1}$ in T rechts von p auf Sweep-Line
      speichere p an e



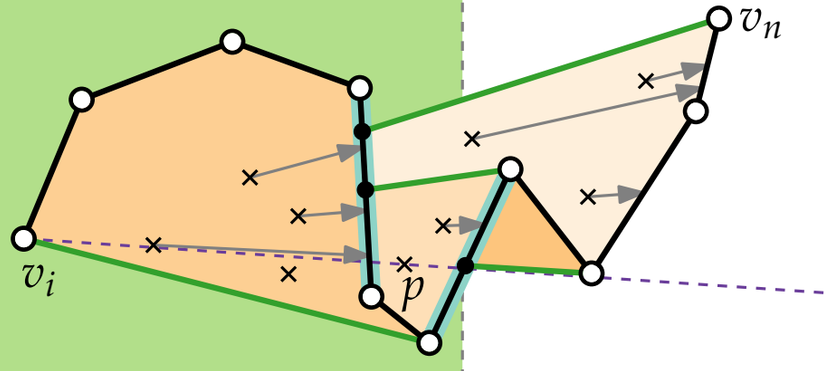
2) Punktzuweisung: Algorithmus

PUNKTZUWEISUNG(S_i, i, P)
  $P' =$ Punkte aus P rechts von v_i
  $Q =$ PRIORITYQUEUE($P' \cup \{v_{i+1}, \ldots, v_n\}$) sortiert nach Winkel bzgl. v_i
  $T =$ balancierter binärer Suchbaum für radialen Sweep
  while !Q.ISEMPTY() **do**
    $p = Q$.EXTRACTMAX()
    if p Knoten von C_{in} **then**
      update T mit Kanten von p
    else
      $e =$ linkeste Kante $v_k v_{k+1}$ in T rechts von p auf Sweep-Line
      speichere p an e



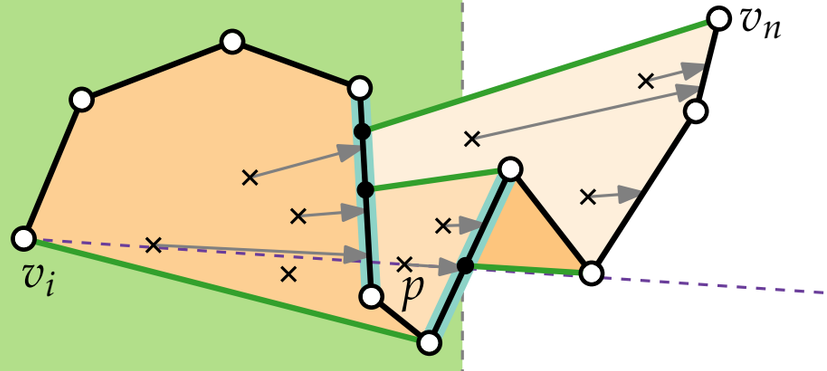
2) Punktzuweisung: Algorithmus

PUNKTZUWEISUNG(S_i, i, P)
  $P' =$ Punkte aus P rechts von v_i
  $Q =$ PRIORITYQUEUE($P' \cup \{v_{i+1}, \ldots, v_n\}$) sortiert nach Winkel bzgl. v_i
  $T =$ balancierter binärer Suchbaum für radialen Sweep
  while !Q.ISEMPTY() **do**
    $p = Q$.EXTRACTMAX()
    if p Knoten von C_{in} **then**
      update T mit Kanten von p
    else
      $e =$ linkeste Kante $v_k v_{k+1}$ in T rechts von p auf Sweep-Line
      speichere p an e



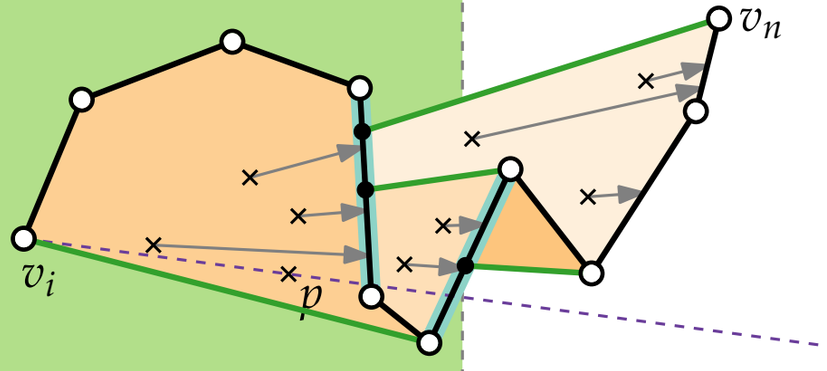
2) Punktzuweisung: Algorithmus

```
PUNKTZUWEISUNG(S_i, i, P)
    P' = Punkte aus P rechts von v_i
    Q = PRIORITYQUEUE(P' ∪ {v_{i+1}, ..., v_n}) sortiert nach Winkel bzgl. v_i
    T = balancierter binärer Suchbaum für radialen Sweep
    while !Q.ISEMPTY() do
        p = Q.EXTRACTMAX()
        if p Knoten von C_in then
            update T mit Kanten von p
        else
            e = linkeste Kante v_k v_{k+1} in T rechts von p auf Sweep-Line
            speichere p an e
```



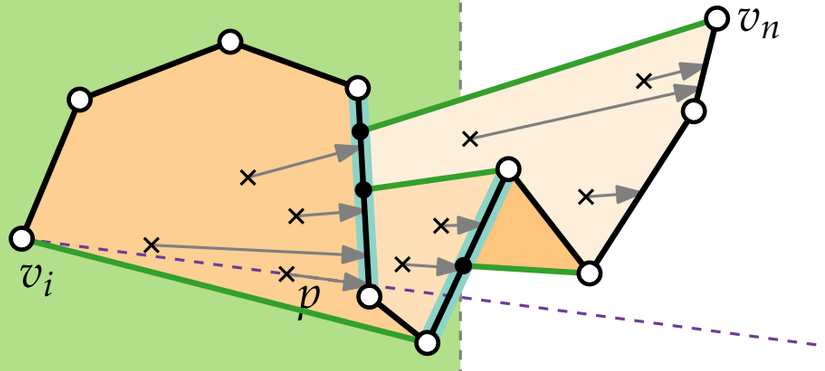
2) Punktzuweisung: Algorithmus

PUNKTZUWEISUNG(S_i, i, P)
  $P' =$ Punkte aus P rechts von v_i
  $Q =$ PRIORITYQUEUE($P' \cup \{v_{i+1}, \ldots, v_n\}$) sortiert nach Winkel bzgl. v_i
  $T =$ balancierter binärer Suchbaum für radialen Sweep
  while !Q.ISEMPTY() **do**
    $p = Q$.EXTRACTMAX()
    if p Knoten von C_{in} **then**
      update T mit Kanten von p
    else
      $e =$ linkeste Kante $v_k v_{k+1}$ in T rechts von p auf Sweep-Line
      speichere p an e



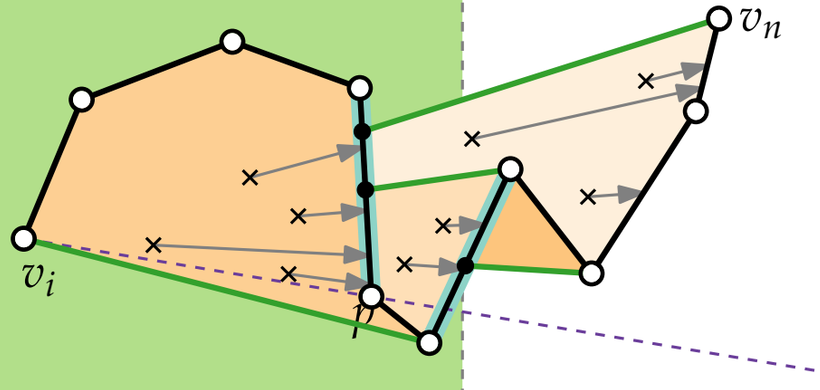
2) Punktzuweisung: Algorithmus

PUNKTZUWEISUNG(S_i, i, P)
  $P' =$ Punkte aus P rechts von v_i
  $Q =$ PRIORITYQUEUE($P' \cup \{v_{i+1}, \ldots, v_n\}$) sortiert nach Winkel bzgl. v_i
  $T =$ balancierter binärer Suchbaum für radialen Sweep
  while !Q.ISEMPTY() **do**
    $p = Q$.EXTRACTMAX()
    if p Knoten von C_{in} **then**
      update T mit Kanten von p
    else
      $e =$ linkeste Kante $v_k v_{k+1}$ in T rechts von p auf Sweep-Line
      speichere p an e



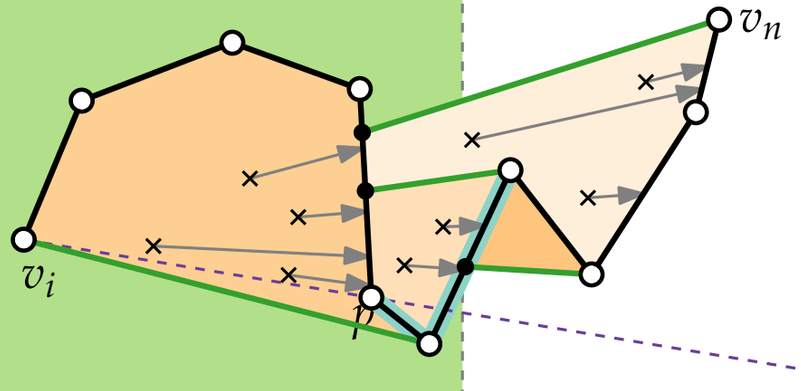
2) Punktzuweisung: Algorithmus

PUNKTZUWEISUNG(S_i, i, P)
  $P' =$ Punkte aus P rechts von v_i
  $Q =$ PRIORITYQUEUE($P' \cup \{v_{i+1}, \ldots, v_n\}$) sortiert nach Winkel bzgl. v_i
  $T =$ balancierter binärer Suchbaum für radialen Sweep
  while !Q.ISEMPTY() **do**
    $p = Q$.EXTRACTMAX()
    if p Knoten von C_{in} **then**
      update T mit Kanten von p
    else
      $e =$ linkeste Kante $v_k v_{k+1}$ in T rechts von p auf Sweep-Line
      speichere p an e



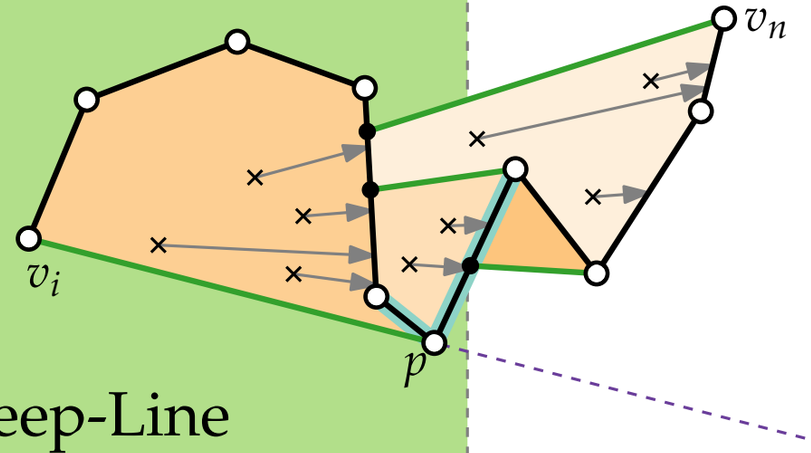
2) Punktzuweisung: Algorithmus

PUNKTZUWEISUNG(S_i, i, P)
  $P' =$ Punkte aus P rechts von v_i
  $Q =$ PRIORITYQUEUE($P' \cup \{v_{i+1}, \ldots, v_n\}$) sortiert nach Winkel bzgl. v_i
  $T =$ balancierter binärer Suchbaum für radialen Sweep
  while !Q.ISEMPTY() **do**
    $p = Q$.EXTRACTMAX()
    if p Knoten von C_{in} **then**
      update T mit Kanten von p
    else
      $e =$ linkeste Kante $v_k v_{k+1}$ in T rechts von p auf Sweep-Line
      speichere p an e



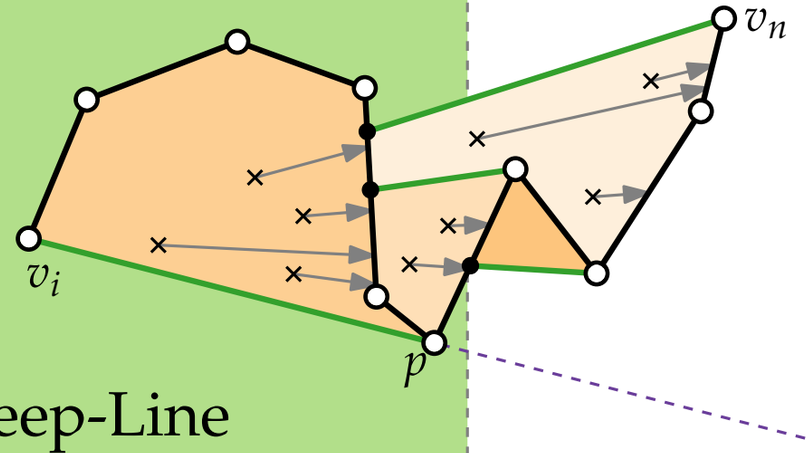
2) Punktzuweisung: Algorithmus

PUNKTZUWEISUNG(S_i, i, P)
  $P' =$ Punkte aus P rechts von v_i
  $Q =$ PRIORITYQUEUE($P' \cup \{v_{i+1}, \ldots, v_n\}$) sortiert nach Winkel bzgl. v_i
  $T =$ balancierter binärer Suchbaum für radialen Sweep
  while !Q.ISEMPTY() **do**
    $p = Q$.EXTRACTMAX()
    if p Knoten von C_{in} **then**
      update T mit Kanten von p
    else
      $e =$ linkeste Kante $v_k v_{k+1}$ in T rechts von p auf Sweep-Line
      speichere p an e



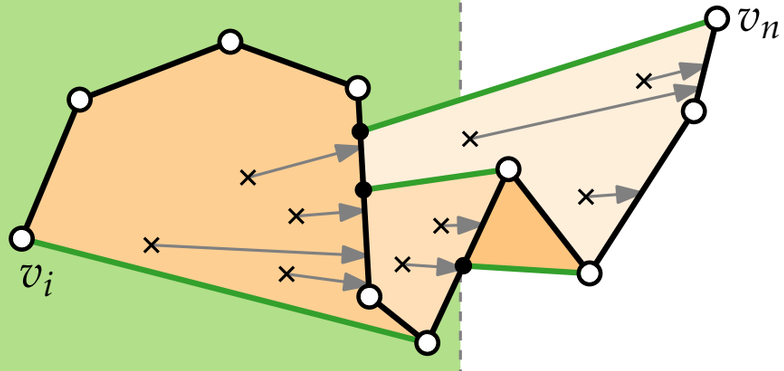
2) Punktzuweisung: Algorithmus

PUNKTZUWEISUNG(S_i, i, P)
  $P' =$ Punkte aus P rechts von v_i
  $Q =$ PRIORITYQUEUE($P' \cup \{v_{i+1}, \ldots, v_n\}$) sortiert nach Winkel bzgl. v_i
  $T =$ balancierter binärer Suchbaum für radialen Sweep
  while !Q.ISEMPTY() **do**
    $p = Q$.EXTRACTMAX()
    if p Knoten von C_{in} **then**
      update T mit Kanten von p
    else
      $e =$ linkeste Kante $v_k v_{k+1}$ in T rechts von p auf Sweep-Line
      speichere p an e



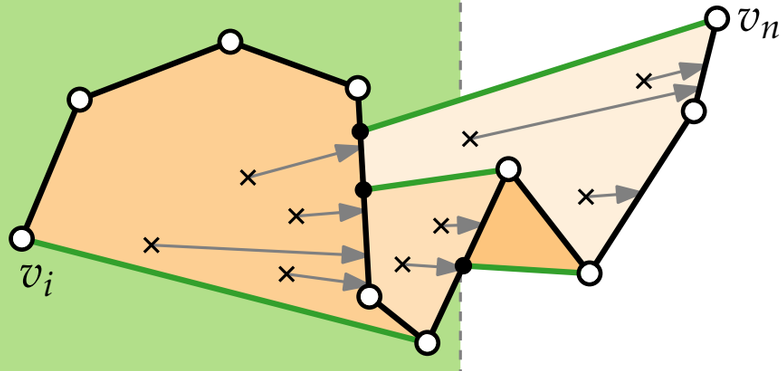
2) Punktzuweisung: Algorithmus

PUNKTZUWEISUNG(S_i, i, P)
  $P' =$ Punkte aus P rechts von v_i
  $Q =$ PRIORITYQUEUE($P' \cup \{v_{i+1}, \ldots, v_n\}$) sortiert nach Winkel bzgl. v_i
  $T =$ balancierter binärer Suchbaum für radialen Sweep
  while !Q.ISEMPTY() **do**
    $p = Q$.EXTRACTMAX()
    if p Knoten von C_{in} **then**
      update T mit Kanten von p
    else
      $e =$ linkeste Kante $v_k v_{k+1}$ in T rechts von p auf Sweep-Line
      speichere p an e



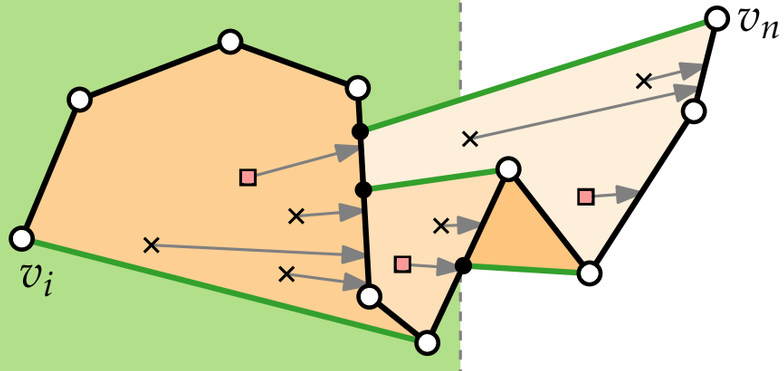
2) Punktzuweisung: Algorithmus

PUNKTZUWEISUNG(S_i, i, P)
  $P' =$ Punkte aus P rechts von v_i
  $Q =$ PRIORITYQUEUE($P' \cup \{v_{i+1}, \ldots, v_n\}$) sortiert nach Winkel bzgl. v_i
  $T =$ balancierter binärer Suchbaum für radialen Sweep
  while !Q.ISEMPTY() **do**
    $p = Q$.EXTRACTMAX()
    if p Knoten von C_{in} **then**
      update T mit Kanten von p
    else
      $e =$ linkeste Kante $v_k v_{k+1}$ in T rechts von p auf Sweep-Line
      speichere p an e
  gehe über alle Kanten von C_{in}, betrachte an Kanten gespeicherte Punkte und weise Facetten f maximalen/minimalen Punkt p_f zu



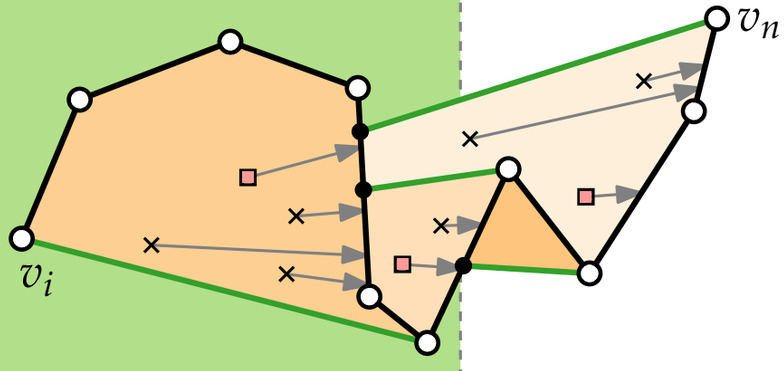
2) Punktzuweisung: Algorithmus

PUNKTZUWEISUNG(S_i, i, P)
  $P' =$ Punkte aus P rechts von v_i
  $Q =$ PRIORITYQUEUE($P' \cup \{v_{i+1}, \ldots, v_n\}$) sortiert nach Winkel bzgl. v_i
  $T =$ balancierter binärer Suchbaum für radialen Sweep
  while !Q.ISEMPTY() **do**
    $p = Q$.EXTRACTMAX()
    if p Knoten von C_{in} **then**
      update T mit Kanten von p
    else
      $e =$ linkeste Kante $v_k v_{k+1}$ in T rechts von p auf Sweep-Line
      speichere p an e
  gehe über alle Kanten von C_{in}, betrachte an Kanten gespeicherte Punkte und weise Facetten f maximalen/minimalen Punkt p_f zu



2) Punktzuweisung: Algorithmus

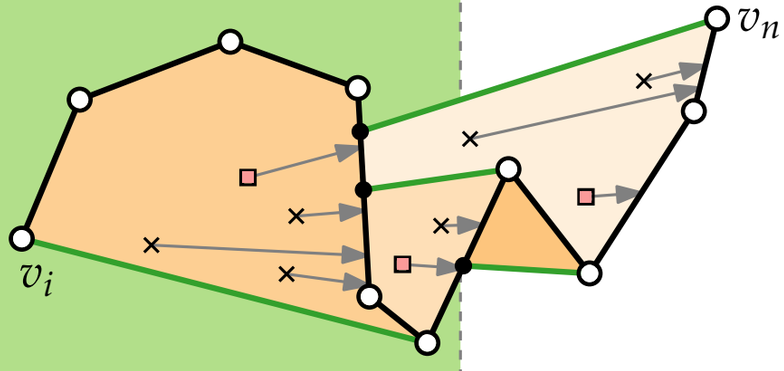
PUNKTZUWEISUNG(S_i, i, P)
  $P' =$ Punkte aus P rechts von v_i
  $Q =$ PRIORITYQUEUE($P' \cup \{v_{i+1}, \ldots, v_n\}$) sortiert nach Winkel bzgl. v_i
  $T =$ balancierter binärer Suchbaum für radialen Sweep
  while !Q.ISEMPTY() **do**
    $p = Q$.EXTRACTMAX()
    if p Knoten von C_{in} **then**
      update T mit Kanten von p
    else
      $e =$ linkeste Kante $v_k v_{k+1}$ in T rechts von p auf Sweep-Line
      speichere p an e
  gehe über alle Kanten von C_{in}, betrachte an Kanten gespeicherte Punkte und weise Facetten f maximalen/minimalen Punkt p_f zu



Laufzeit?

2) Punktzuweisung: Algorithmus

PUNKTZUWEISUNG(S_i, i, P)
  $P' =$ Punkte aus P rechts von v_i
  $Q =$ PRIORITYQUEUE($P' \cup \{v_{i+1}, \ldots, v_n\}$) sortiert nach Winkel bzgl. v_i
  $T =$ balancierter binärer Suchbaum für radialen Sweep
  while !Q.ISEMPTY() **do**
    $p = Q$.EXTRACTMAX()
    if p Knoten von C_{in} **then**
      update T mit Kanten von p
    else
      $e =$ linkeste Kante $v_k v_{k+1}$ in T rechts von p auf Sweep-Line
      speichere p an e
  gehe über alle Kanten von C_{in}, betrachte an Kanten gespeicherte Punkte und weise Facetten f maximalen/minimalen Punkt p_f zu



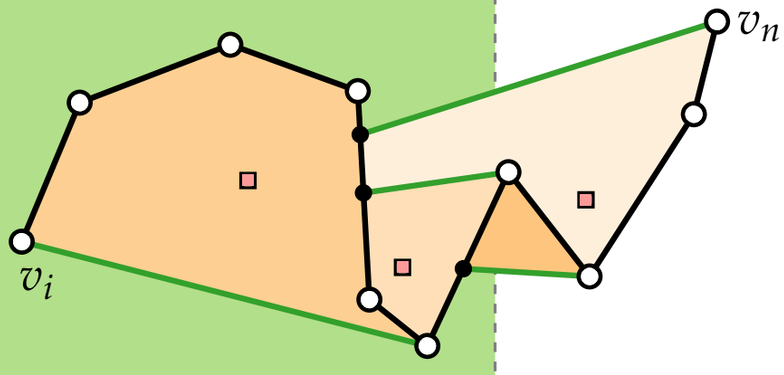
Laufzeit? $\mathcal{O}((n+m)\log n)$

3) Konsistenzberechnung

KONSISTENZBERECHNUNG(S_i, i, P')

3) Konsistenzberechnung

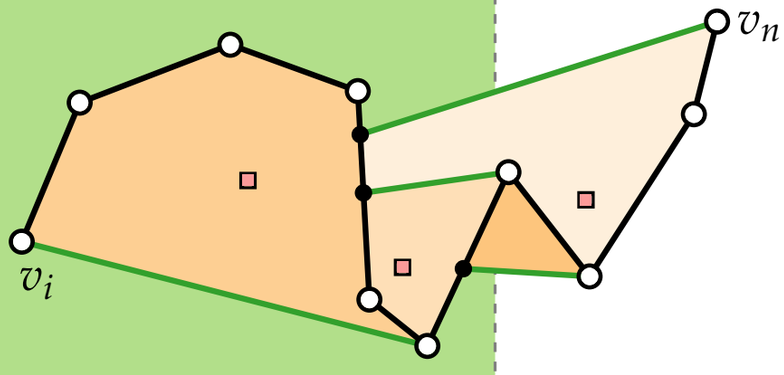
KONSISTENZBERECHNUNG(S_i, i, P')



3) Konsistenzberechnung

KONSISTENZBERECHNUNG(S_i, i, P')

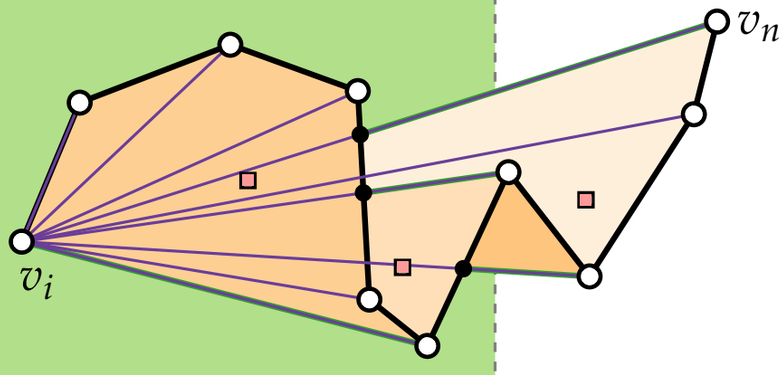
Q = DEQUE für Abkürzungen (v_i, v_j) absteigend sortiert nach Steigung



3) Konsistenzberechnung

KONSISTENZBERECHNUNG(S_i, i, P')

Q = DEQUE für Abkürzungen (v_i, v_j) absteigend sortiert nach Steigung

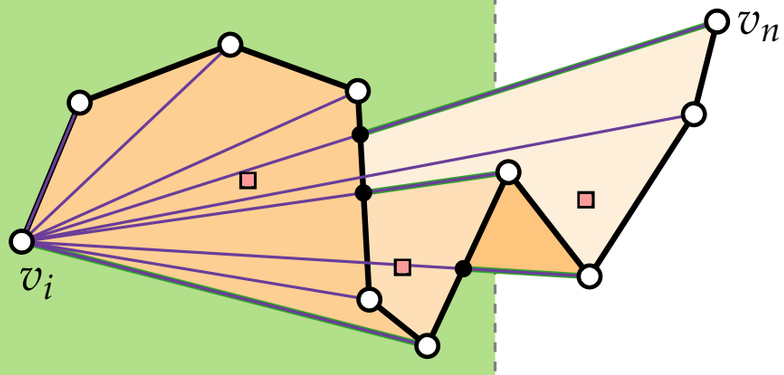


3) Konsistenzberechnung

KONSISTENZBERECHNUNG(S_i, i, P')

Q = DEQUE für Abkürzungen (v_i, v_j) absteigend sortiert nach Steigung

$j = i + 1$

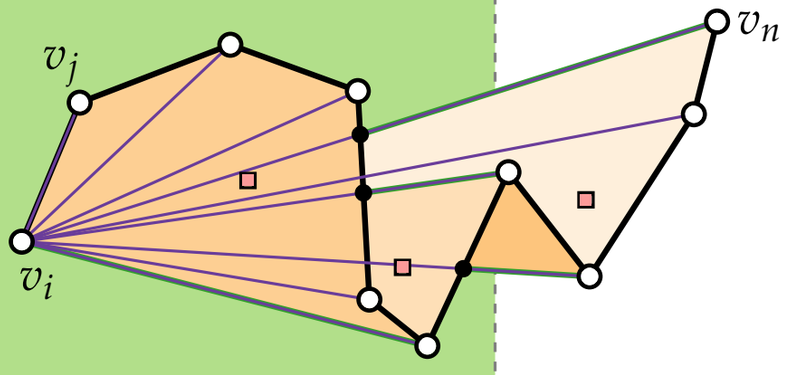


3) Konsistenzberechnung

KONSISTENZBERECHNUNG(S_i, i, P')

Q = DEQUE für Abkürzungen (v_i, v_j) absteigend sortiert nach Steigung

$j = i + 1$



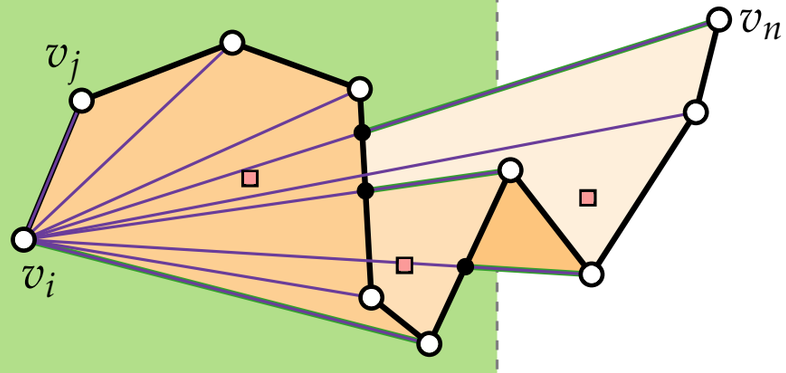
3) Konsistenzberechnung

KONSISTENZBERECHNUNG(S_i, i, P')

Q = DEQUE für Abkürzungen (v_i, v_j) absteigend sortiert nach Steigung

$j = i + 1$

for $f_h \in S_i$ **do**



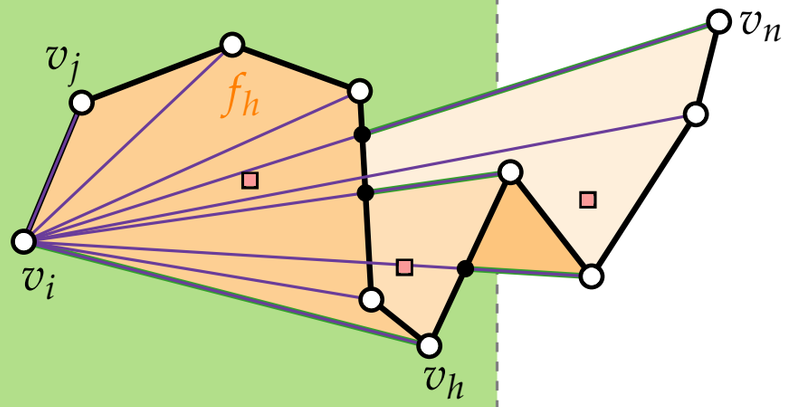
3) Konsistenzberechnung

KONSISTENZBERECHNUNG(S_i, i, P')

Q = DEQUE für Abkürzungen (v_i, v_j) absteigend sortiert nach Steigung

$j = i + 1$

for $f_h \in S_i$ **do**



3) Konsistenzberechnung

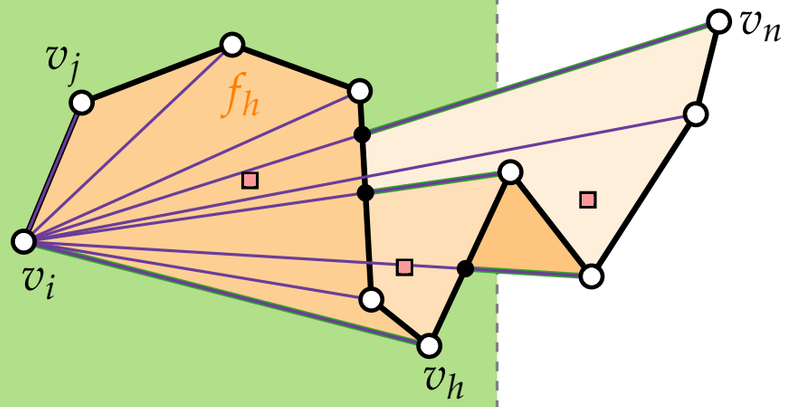
KONSISTENZBERECHNUNG(S_i, i, P')

Q = DEQUE für Abkürzungen (v_i, v_j) absteigend sortiert nach Steigung

$j = i + 1$

for $f_h \in S_i$ **do**

  if $p_f \neq$ **nil then**



3) Konsistenzberechnung

KONSISTENZBERECHNUNG(S_i, i, P')

Q = DEQUE für Abkürzungen (v_i, v_j) absteigend sortiert nach Steigung

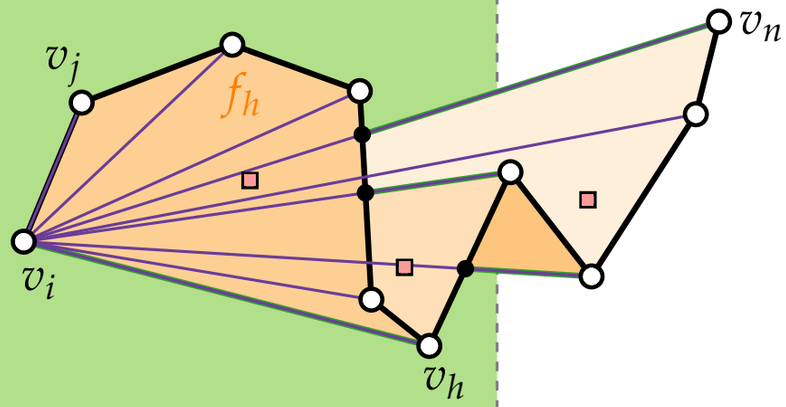
$j = i + 1$

for $f_h \in S_i$ **do**

  if $p_f \neq$ **nil then**

    if f maximale Facette **then**

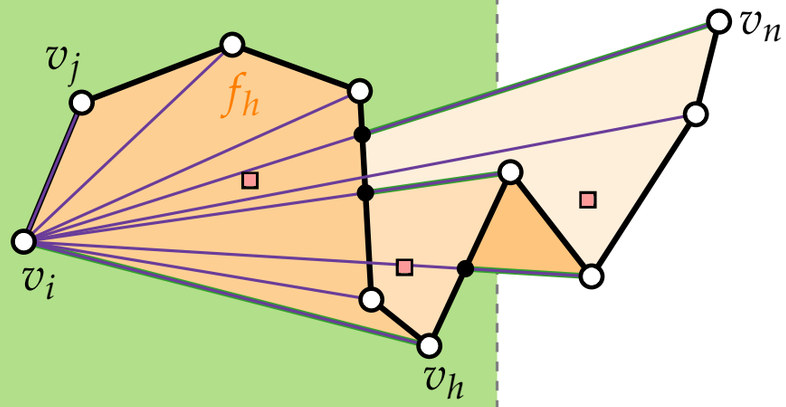
    else



3) Konsistenzberechnung

KONSISTENZBERECHNUNG(S_i, i, P')

Q = DEQUE für Abkürzungen (v_i, v_j) absteigend sortiert nach Steigung
$j = i + 1$
for $f_h \in S_i$ **do**
  if $p_f \neq$ **nil then**
    if f maximale Facette **then**
      else
        Entferne Abkürzungen vom Ende von Q
        solange Steigung kleiner als $v_i p_f$



3) Konsistenzberechnung

KONSISTENZBERECHNUNG(S_i, i, P')

Q = DEQUE für Abkürzungen (v_i, v_j) absteigend sortiert nach Steigung

$j = i + 1$

for $f_h \in S_i$ **do**

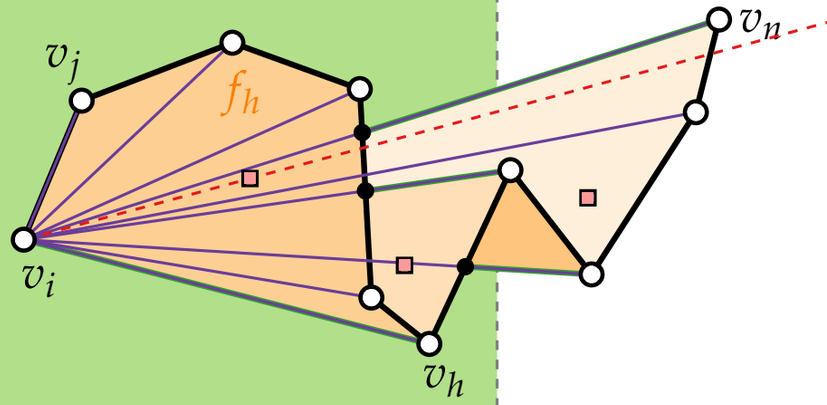
  if $p_f \neq$ **nil then**

    if f maximale Facette **then**

      else

        Entferne Abkürzungen vom Ende von Q

        solange Steigung kleiner als $v_i p_f$



3) Konsistenzberechnung

KONSISTENZBERECHNUNG(S_i, i, P')

Q = DEQUE für Abkürzungen (v_i, v_j) absteigend sortiert nach Steigung

$j = i + 1$

for $f_h \in S_i$ **do**

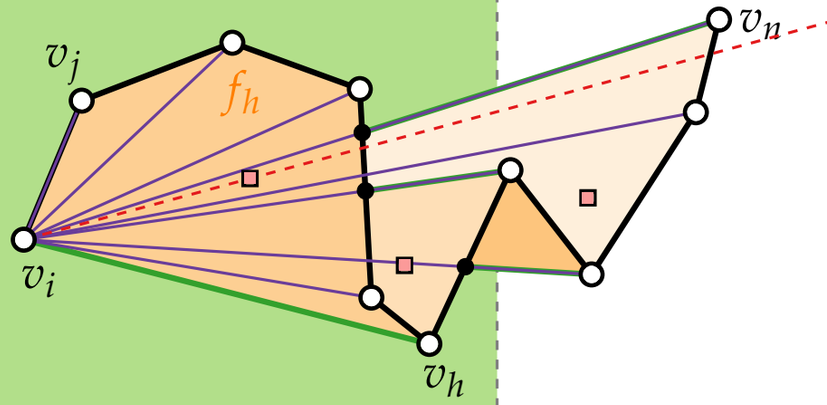
  if $p_f \neq$ **nil then**

    if f maximale Facette **then**

      else

        Entferne Abkürzungen vom Ende von Q

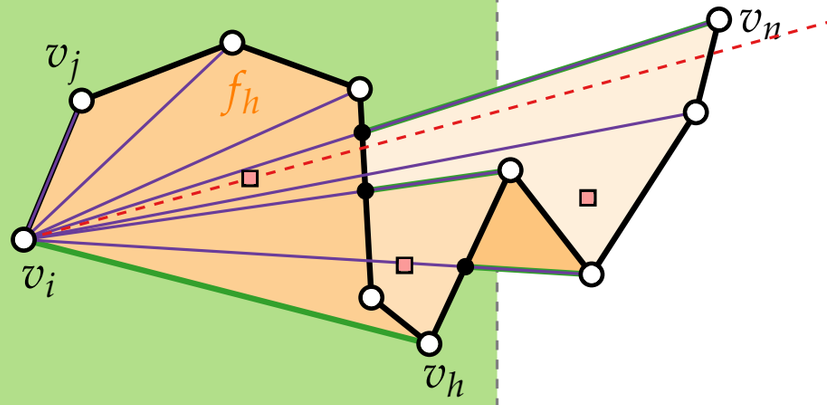
        solange Steigung kleiner als $v_i p_f$



3) Konsistenzberechnung

KONSISTENZBERECHNUNG(S_i, i, P')

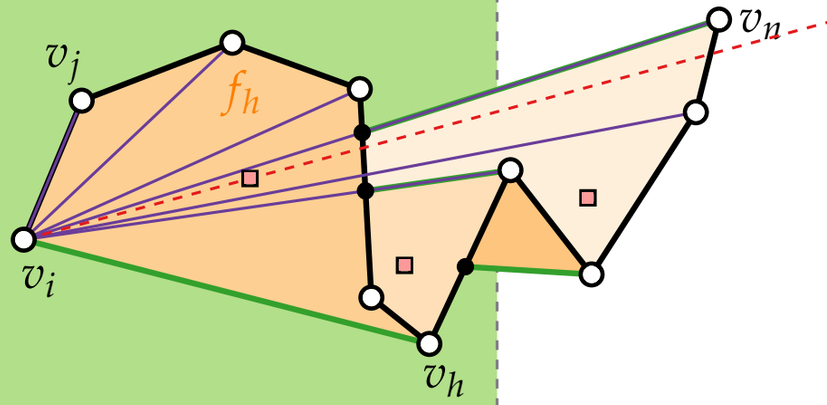
Q = DEQUE für Abkürzungen (v_i, v_j) absteigend sortiert nach Steigung
$j = i + 1$
for $f_h \in S_i$ **do**
  if $p_f \neq$ **nil then**
    if f maximale Facette **then**
      else
        Entferne Abkürzungen vom Ende von Q
        solange Steigung kleiner als $v_i p_f$



3) Konsistenzberechnung

KONSISTENZBERECHNUNG(S_i, i, P')

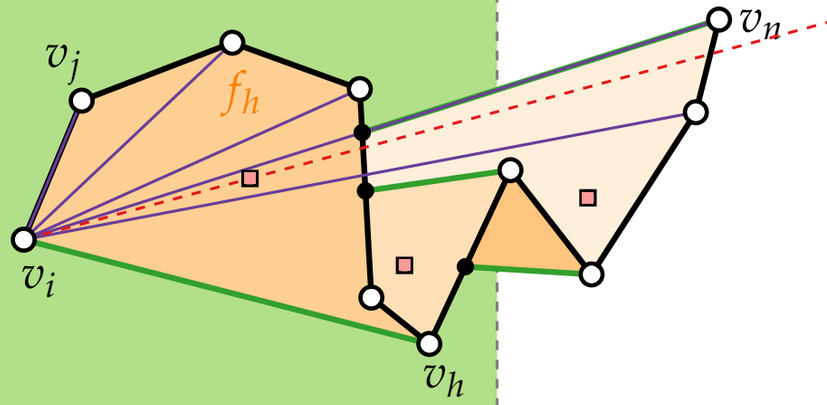
Q = DEQUE für Abkürzungen (v_i, v_j) absteigend sortiert nach Steigung
$j = i + 1$
for $f_h \in S_i$ **do**
  if $p_f \neq$ **nil then**
    if f maximale Facette **then**
      else
        Entferne Abkürzungen vom Ende von Q
        solange Steigung kleiner als $v_i p_f$



3) Konsistenzberechnung

KONSISTENZBERECHNUNG(S_i, i, P')

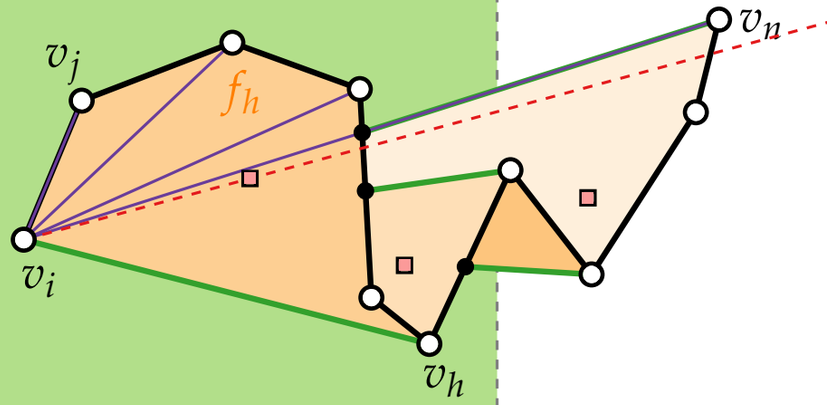
Q = DEQUE für Abkürzungen (v_i, v_j) absteigend sortiert nach Steigung
$j = i + 1$
for $f_h \in S_i$ **do**
  if $p_f \neq$ **nil then**
    if f maximale Facette **then**
      else
        Entferne Abkürzungen vom Ende von Q
        solange Steigung kleiner als $v_i p_f$



3) Konsistenzberechnung

KONSISTENZBERECHNUNG(S_i, i, P')

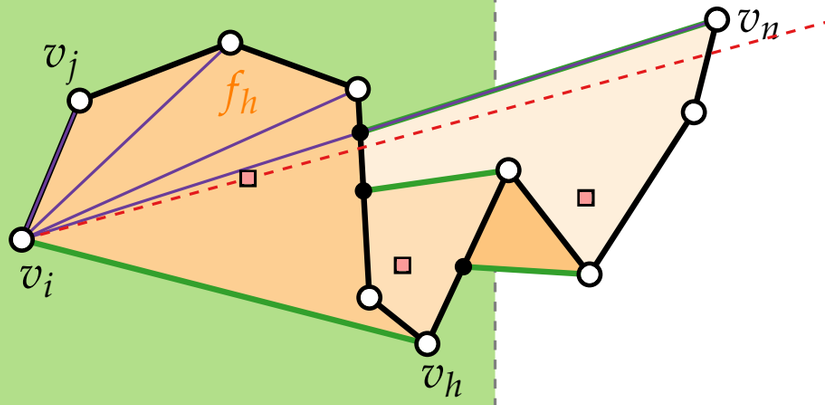
Q = DEQUE für Abkürzungen (v_i, v_j) absteigend sortiert nach Steigung
$j = i + 1$
for $f_h \in S_i$ **do**
  if $p_f \neq$ **nil then**
    if f maximale Facette **then**
      else
        Entferne Abkürzungen vom Ende von Q
        solange Steigung kleiner als $v_i p_f$



3) Konsistenzberechnung

KONSISTENZBERECHNUNG(S_i, i, P')

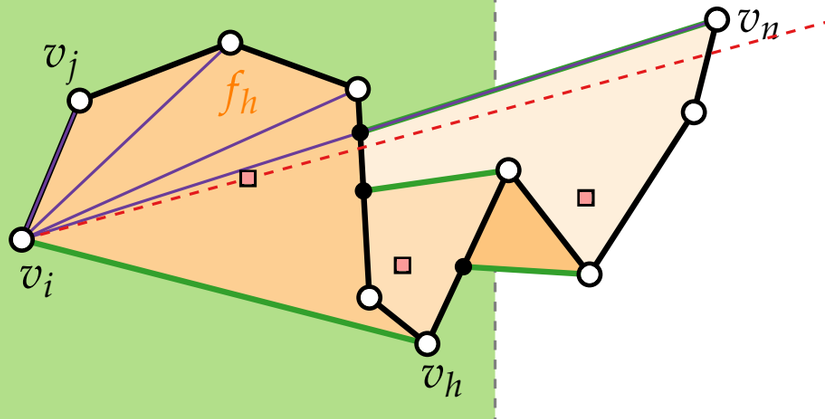
Q = DEQUE für Abkürzungen (v_i, v_j) absteigend sortiert nach Steigung
$j = i + 1$
for $f_h \in S_i$ **do**
  if $p_f \neq$ **nil then**
    if f maximale Facette **then**
      Entferne Abkürzungen vom Anfang von Q
      solange Steigung größer als $v_i p_f$
    else
      Entferne Abkürzungen vom Ende von Q
      solange Steigung kleiner als $v_i p_f$



3) Konsistenzberechnung

KONSISTENZBERECHNUNG(S_i, i, P')

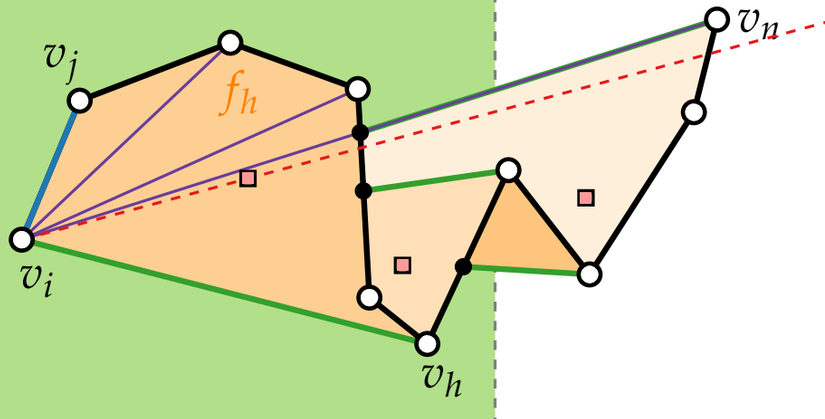
Q = DEQUE für Abkürzungen (v_i, v_j) absteigend sortiert nach Steigung
$j = i + 1$
for $f_h \in S_i$ **do**
  if $p_f \neq$ **nil then**
    if f maximale Facette **then**
      Entferne Abkürzungen vom Anfang von Q
      solange Steigung größer als $v_i p_f$
    else
      Entferne Abkürzungen vom Ende von Q
      solange Steigung kleiner als $v_i p_f$
  while $v_j \neq v_h$ **do**
    if $v_i v_j \in Q$ **then** A.add((v_i, v_j)); Q.remove((v_i, v_j))
    $j = j + 1$



3) Konsistenzberechnung

KONSISTENZBERECHNUNG(S_i, i, P')

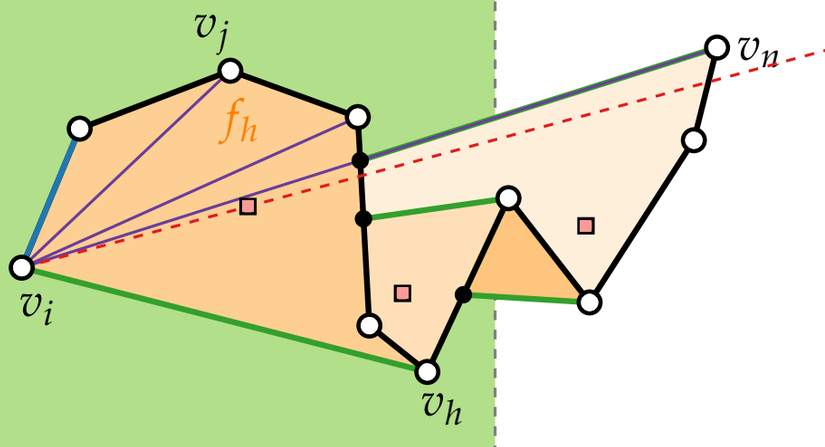
Q = DEQUE für Abkürzungen (v_i, v_j) absteigend sortiert nach Steigung
$j = i + 1$
for $f_h \in S_i$ **do**
  if $p_f \neq$ **nil then**
    if f maximale Facette **then**
      Entferne Abkürzungen vom Anfang von Q
      solange Steigung größer als $v_i p_f$
    else
      Entferne Abkürzungen vom Ende von Q
      solange Steigung kleiner als $v_i p_f$
  while $v_j \neq v_h$ **do**
    if $v_i v_j \in Q$ **then** A.add((v_i, v_j)); Q.remove((v_i, v_j))
    $j = j + 1$



3) Konsistenzberechnung

KONSISTENZBERECHNUNG(S_i, i, P')

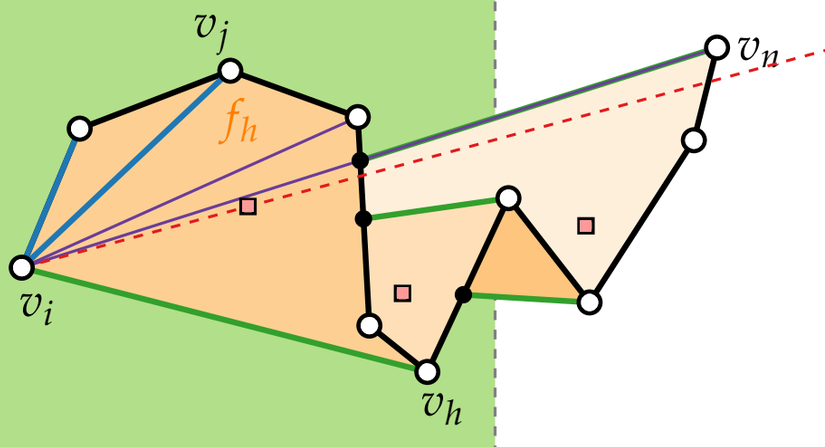
Q = DEQUE für Abkürzungen (v_i, v_j) absteigend sortiert nach Steigung
$j = i + 1$
for $f_h \in S_i$ **do**
  if $p_f \neq$ **nil then**
    if f maximale Facette **then**
      Entferne Abkürzungen vom Anfang von Q
      solange Steigung größer als $v_i p_f$
    else
      Entferne Abkürzungen vom Ende von Q
      solange Steigung kleiner als $v_i p_f$
  while $v_j \neq v_h$ **do**
    if $v_i v_j \in Q$ **then** A.add((v_i, v_j)); Q.remove((v_i, v_j))
    $j = j + 1$



3) Konsistenzberechnung

KONSISTENZBERECHNUNG(S_i, i, P')

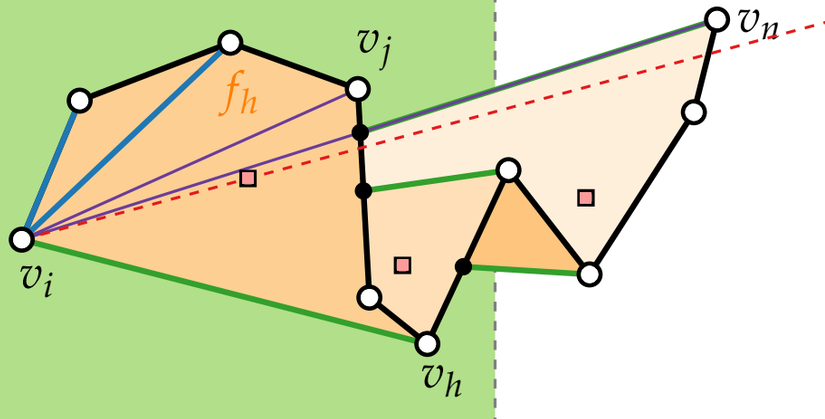
Q = DEQUE für Abkürzungen (v_i, v_j) absteigend sortiert nach Steigung
$j = i + 1$
for $f_h \in S_i$ **do**
  if $p_f \neq$ **nil then**
    if f maximale Facette **then**
      Entferne Abkürzungen vom Anfang von Q
      solange Steigung größer als $v_i p_f$
    else
      Entferne Abkürzungen vom Ende von Q
      solange Steigung kleiner als $v_i p_f$
  while $v_j \neq v_h$ **do**
    if $v_i v_j \in Q$ **then** A.add((v_i, v_j)); Q.remove((v_i, v_j))
    $j = j + 1$



3) Konsistenzberechnung

KONSISTENZBERECHNUNG(S_i, i, P')

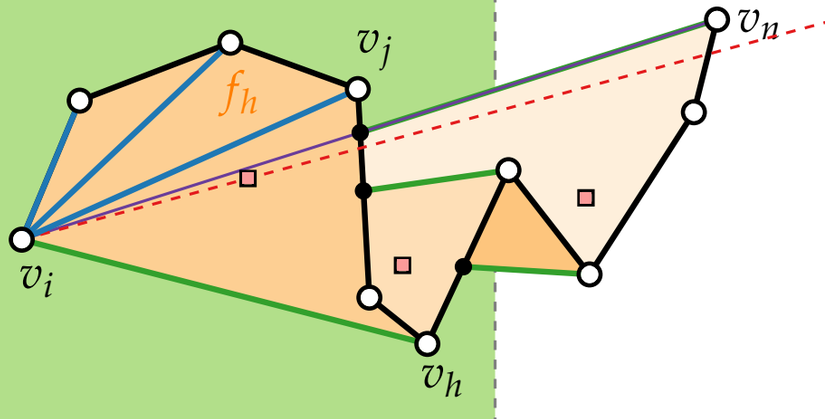
Q = DEQUE für Abkürzungen (v_i, v_j) absteigend sortiert nach Steigung
$j = i + 1$
for $f_h \in S_i$ **do**
  if $p_f \neq$ **nil then**
    if f maximale Facette **then**
      Entferne Abkürzungen vom Anfang von Q
      solange Steigung größer als $v_i p_f$
    else
      Entferne Abkürzungen vom Ende von Q
      solange Steigung kleiner als $v_i p_f$
  while $v_j \neq v_h$ **do**
    if $v_i v_j \in Q$ **then** A.add((v_i, v_j)); Q.remove((v_i, v_j))
    $j = j + 1$



3) Konsistenzberechnung

KONSISTENZBERECHNUNG(S_i, i, P')

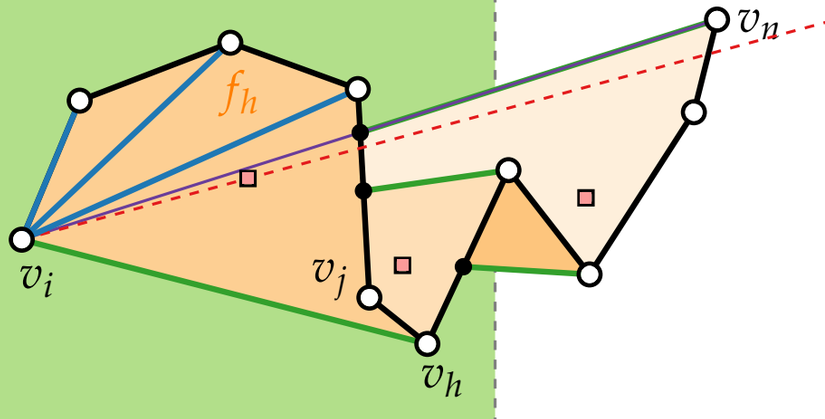
Q = DEQUE für Abkürzungen (v_i, v_j) absteigend sortiert nach Steigung
$j = i + 1$
for $f_h \in S_i$ **do**
  if $p_f \neq$ **nil then**
    if f maximale Facette **then**
      Entferne Abkürzungen vom Anfang von Q
      solange Steigung größer als $v_i p_f$
    else
      Entferne Abkürzungen vom Ende von Q
      solange Steigung kleiner als $v_i p_f$
  while $v_j \neq v_h$ **do**
    if $v_i v_j \in Q$ **then** A.add((v_i, v_j)); Q.remove((v_i, v_j))
    $j = j + 1$



3) Konsistenzberechnung

KONSISTENZBERECHNUNG(S_i, i, P')

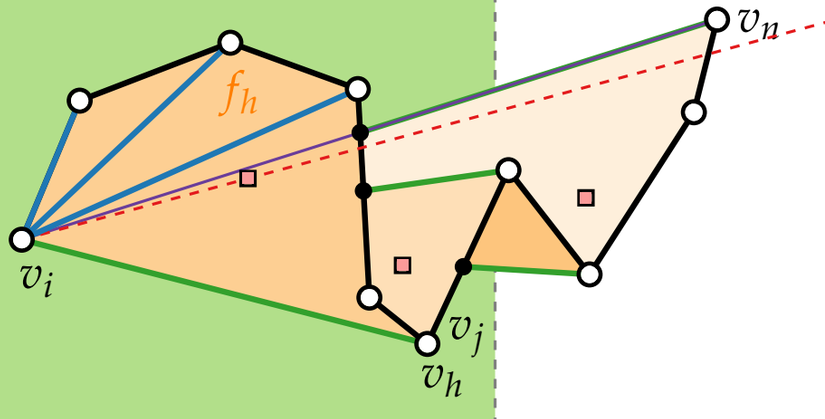
Q = DEQUE für Abkürzungen (v_i, v_j) absteigend sortiert nach Steigung
$j = i + 1$
for $f_h \in S_i$ **do**
  if $p_f \neq$ **nil then**
    if f maximale Facette **then**
      Entferne Abkürzungen vom Anfang von Q
      solange Steigung größer als $v_i p_f$
    else
      Entferne Abkürzungen vom Ende von Q
      solange Steigung kleiner als $v_i p_f$
  while $v_j \neq v_h$ **do**
    if $v_i v_j \in Q$ **then** A.add((v_i, v_j)); Q.remove((v_i, v_j))
    $j = j + 1$



3) Konsistenzberechnung

KONSISTENZBERECHNUNG(S_i, i, P')

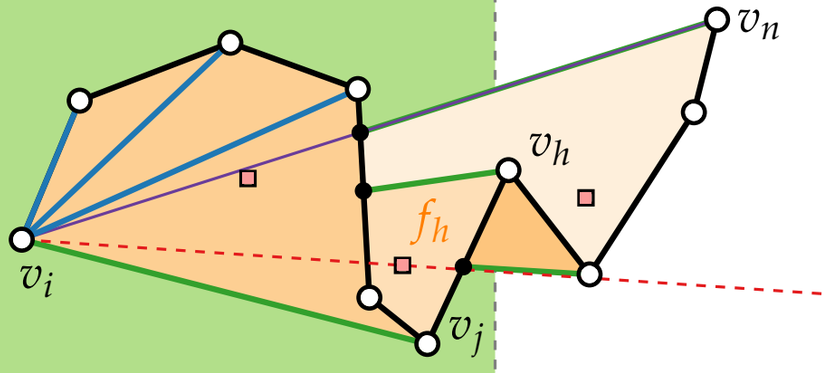
Q = DEQUE für Abkürzungen (v_i, v_j) absteigend sortiert nach Steigung
$j = i + 1$
for $f_h \in S_i$ **do**
  if $p_f \neq$ **nil then**
    if f maximale Facette **then**
      Entferne Abkürzungen vom Anfang von Q
      solange Steigung größer als $v_i p_f$
    else
      Entferne Abkürzungen vom Ende von Q
      solange Steigung kleiner als $v_i p_f$
  while $v_j \neq v_h$ **do**
    if $v_i v_j \in Q$ **then** A.add((v_i, v_j)); Q.remove((v_i, v_j))
    $j = j + 1$



3) Konsistenzberechnung

KONSISTENZBERECHNUNG(S_i, i, P')

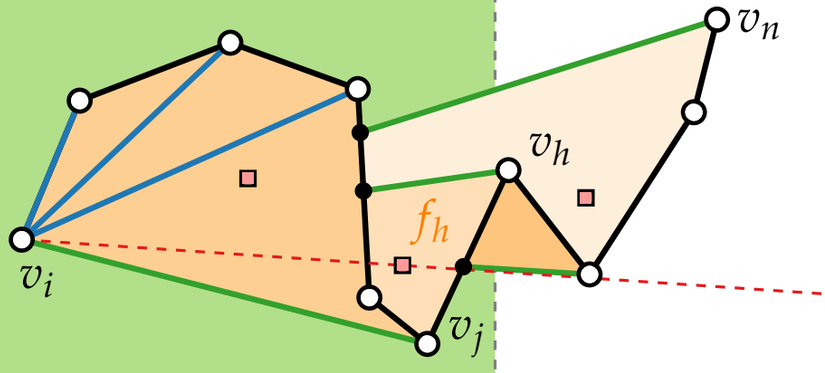
Q = DEQUE für Abkürzungen (v_i, v_j) absteigend sortiert nach Steigung
$j = i + 1$
for $f_h \in S_i$ **do**
  if $p_f \neq$ **nil then**
    if f maximale Facette **then**
      Entferne Abkürzungen vom Anfang von Q
      solange Steigung größer als $v_i p_f$
    else
      Entferne Abkürzungen vom Ende von Q
      solange Steigung kleiner als $v_i p_f$
  while $v_j \neq v_h$ **do**
    if $v_i v_j \in Q$ **then** A.add((v_i, v_j)); Q.remove((v_i, v_j))
    $j = j + 1$



3) Konsistenzberechnung

KONSISTENZBERECHNUNG(S_i, i, P')

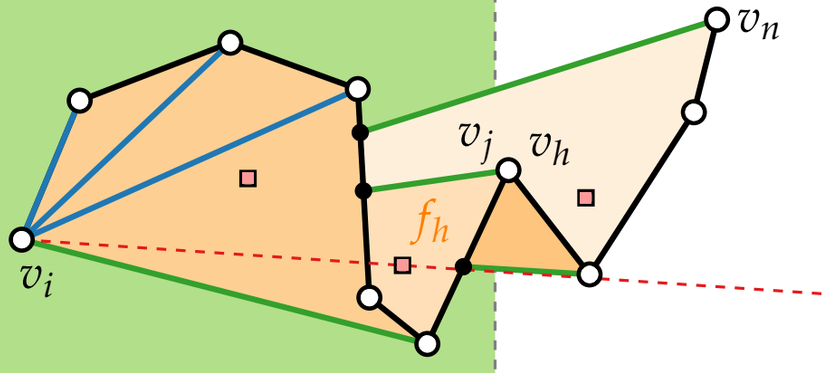
Q = DEQUE für Abkürzungen (v_i, v_j) absteigend sortiert nach Steigung
$j = i + 1$
for $f_h \in S_i$ **do**
  if $p_f \neq$ **nil then**
    if f maximale Facette **then**
      Entferne Abkürzungen vom Anfang von Q
      solange Steigung größer als $v_i p_f$
    else
      Entferne Abkürzungen vom Ende von Q
      solange Steigung kleiner als $v_i p_f$
  while $v_j \neq v_h$ **do**
    if $v_i v_j \in Q$ **then** A.add((v_i, v_j)); Q.remove((v_i, v_j))
    $j = j + 1$



3) Konsistenzberechnung

KONSISTENZBERECHNUNG(S_i, i, P')

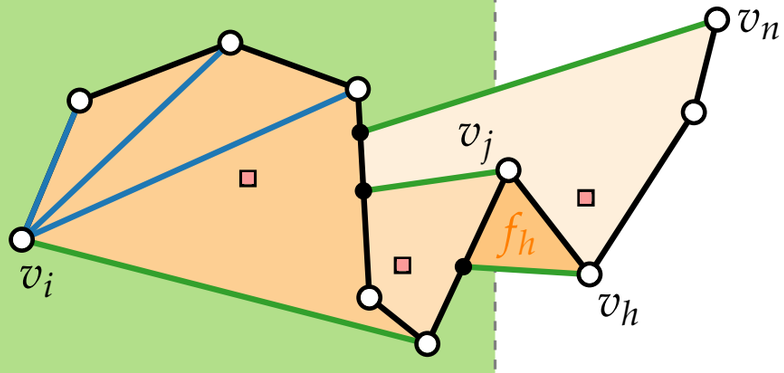
Q = DEQUE für Abkürzungen (v_i, v_j) absteigend sortiert nach Steigung
$j = i + 1$
for $f_h \in S_i$ **do**
  if $p_f \neq$ **nil then**
    if f maximale Facette **then**
      Entferne Abkürzungen vom Anfang von Q
      solange Steigung größer als $v_i p_f$
    else
      Entferne Abkürzungen vom Ende von Q
      solange Steigung kleiner als $v_i p_f$
  while $v_j \neq v_h$ **do**
    if $v_i v_j \in Q$ **then** A.add((v_i, v_j)); Q.remove((v_i, v_j))
    $j = j + 1$



3) Konsistenzberechnung

KONSISTENZBERECHNUNG(S_i, i, P')

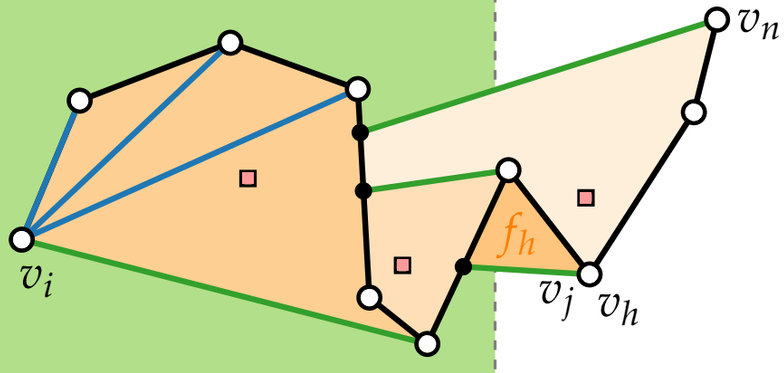
Q = DEQUE für Abkürzungen (v_i, v_j) absteigend sortiert nach Steigung
$j = i + 1$
for $f_h \in S_i$ **do**
  if $p_f \neq$ **nil then**
    if f maximale Facette **then**
      Entferne Abkürzungen vom Anfang von Q
      solange Steigung größer als $v_i p_f$
    else
      Entferne Abkürzungen vom Ende von Q
      solange Steigung kleiner als $v_i p_f$
  while $v_j \neq v_h$ **do**
    if $v_i v_j \in Q$ **then** A.add((v_i, v_j)); Q.remove((v_i, v_j))
    $j = j + 1$



3) Konsistenzberechnung

KONSISTENZBERECHNUNG(S_i, i, P')

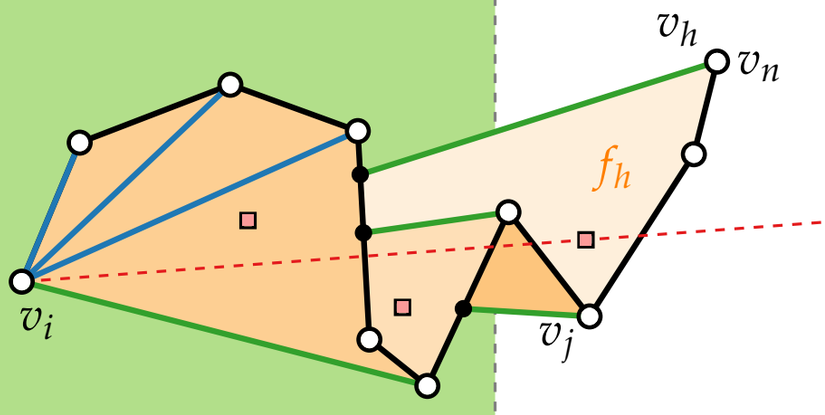
Q = DEQUE für Abkürzungen (v_i, v_j) absteigend sortiert nach Steigung
$j = i + 1$
for $f_h \in S_i$ **do**
  if $p_f \neq$ **nil then**
    if f maximale Facette **then**
      Entferne Abkürzungen vom Anfang von Q
      solange Steigung größer als $v_i p_f$
    else
      Entferne Abkürzungen vom Ende von Q
      solange Steigung kleiner als $v_i p_f$
  while $v_j \neq v_h$ **do**
    if $v_i v_j \in Q$ **then** A.add((v_i, v_j)); Q.remove((v_i, v_j))
    $j = j + 1$



3) Konsistenzberechnung

KONSISTENZBERECHNUNG(S_i, i, P')

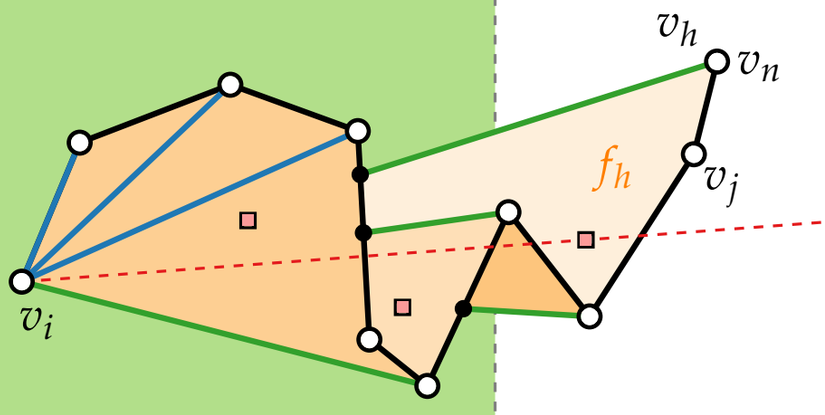
Q = DEQUE für Abkürzungen (v_i, v_j) absteigend sortiert nach Steigung
$j = i + 1$
for $f_h \in S_i$ **do**
  if $p_f \neq$ **nil then**
    if f maximale Facette **then**
      Entferne Abkürzungen vom Anfang von Q
      solange Steigung größer als $v_i p_f$
    else
      Entferne Abkürzungen vom Ende von Q
      solange Steigung kleiner als $v_i p_f$
  while $v_j \neq v_h$ **do**
    if $v_i v_j \in Q$ **then** A.add((v_i, v_j)); Q.remove((v_i, v_j))
    $j = j + 1$



3) Konsistenzberechnung

KONSISTENZBERECHNUNG(S_i, i, P')

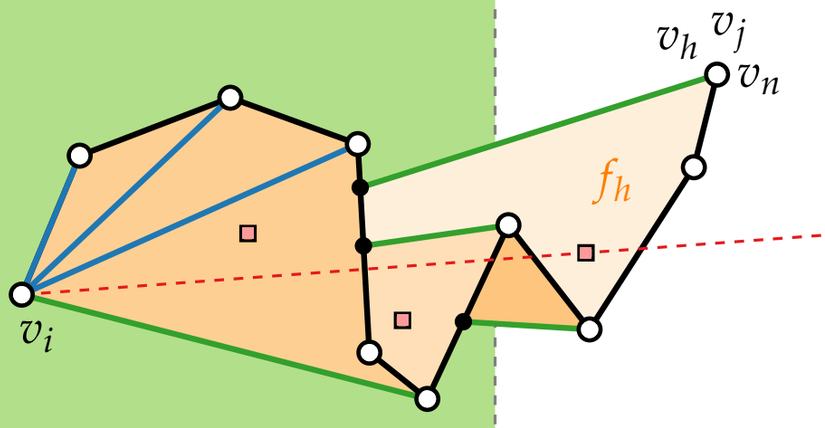
Q = DEQUE für Abkürzungen (v_i, v_j) absteigend sortiert nach Steigung
$j = i + 1$
for $f_h \in S_i$ **do**
  if $p_f \neq$ **nil then**
    if f maximale Facette **then**
      Entferne Abkürzungen vom Anfang von Q
      solange Steigung größer als $v_i p_f$
    else
      Entferne Abkürzungen vom Ende von Q
      solange Steigung kleiner als $v_i p_f$
  while $v_j \neq v_h$ **do**
    if $v_i v_j \in Q$ **then** A.add((v_i, v_j)); Q.remove((v_i, v_j))
    $j = j + 1$



3) Konsistenzberechnung

KONSISTENZBERECHNUNG(S_i, i, P')

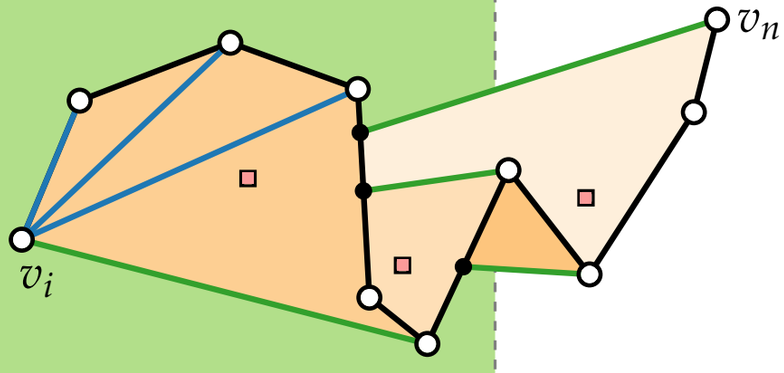
Q = DEQUE für Abkürzungen (v_i, v_j) absteigend sortiert nach Steigung
$j = i + 1$
for $f_h \in S_i$ **do**
  if $p_f \neq$ **nil then**
    if f maximale Facette **then**
      Entferne Abkürzungen vom Anfang von Q
      solange Steigung größer als $v_i p_f$
    else
      Entferne Abkürzungen vom Ende von Q
      solange Steigung kleiner als $v_i p_f$
  while $v_j \neq v_h$ **do**
    if $v_i v_j \in Q$ **then** A.add((v_i, v_j)); Q.remove((v_i, v_j))
    $j = j + 1$



3) Konsistenzberechnung

KONSISTENZBERECHNUNG(S_i, i, P')

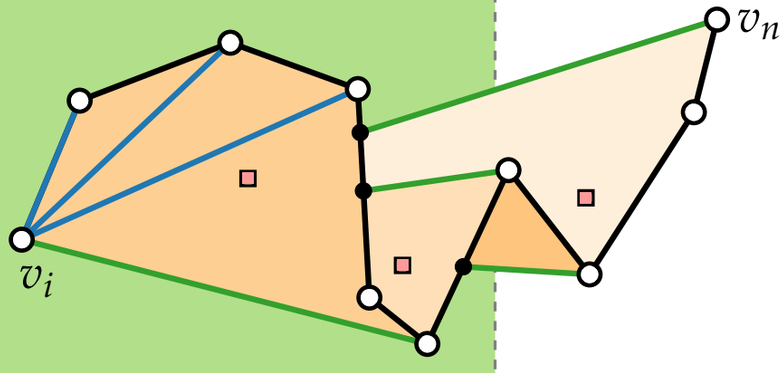
Q = DEQUE für Abkürzungen (v_i, v_j) absteigend sortiert nach Steigung
$j = i + 1$
for $f_h \in S_i$ **do**
  if $p_f \neq$ **nil then**
    if f maximale Facette **then**
      Entferne Abkürzungen vom Anfang von Q
      solange Steigung größer als $v_i p_f$
    else
      Entferne Abkürzungen vom Ende von Q
      solange Steigung kleiner als $v_i p_f$
  while $v_j \neq v_h$ **do**
    if $v_i v_j \in Q$ **then** A.add((v_i, v_j)); Q.remove((v_i, v_j))
    $j = j + 1$



3) Konsistenzberechnung

KONSISTENZBERECHNUNG(S_i, i, P')

Q = DEQUE für Abkürzungen (v_i, v_j) absteigend sortiert nach Steigung
$j = i + 1$
for $f_h \in S_i$ **do**
  if $p_f \neq$ **nil then**
    if f maximale Facette **then**
      Entferne Abkürzungen vom Anfang von Q
      solange Steigung größer als $v_i p_f$
    else
      Entferne Abkürzungen vom Ende von Q
      solange Steigung kleiner als $v_i p_f$
  while $v_j \neq v_h$ **do**
    if $v_i v_j \in Q$ **then** A.add((v_i, v_j)); Q.remove((v_i, v_j))
    $j = j + 1$

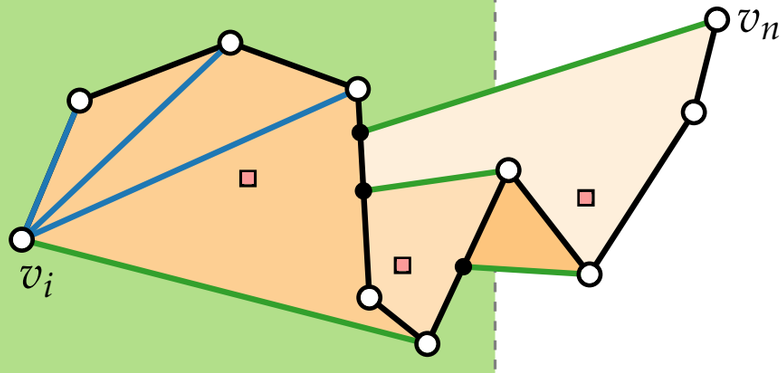


Laufzeit?

3) Konsistenzberechnung

KONSISTENZBERECHNUNG(S_i, i, P')

Q = DEQUE für Abkürzungen (v_i, v_j) absteigend sortiert nach Steigung
$j = i + 1$
for $f_h \in S_i$ **do**
  if $p_f \neq$ **nil then**
    if f maximale Facette **then**
      Entferne Abkürzungen vom Anfang von Q
      solange Steigung größer als $v_i p_f$
    else
      Entferne Abkürzungen vom Ende von Q
      solange Steigung kleiner als $v_i p_f$
  while $v_j \neq v_h$ **do**
    if $v_i v_j \in Q$ **then** A.add((v_i, v_j)); Q.remove((v_i, v_j))
    $j = j + 1$



Laufzeit? $\mathcal{O}(n \log n)$

Zusammenfassung

Lemma 6. Für x-monotonen Kantenzug C mit n Knoten und Menge P von m Punkten können alle konsistenten Abkürzungen für einen Knoten v von C in Zeit $\mathcal{O}((n+m)\log n)$ berechnet werden.

Zusammenfassung

Lemma 6. Für x-monotonen Kantenzug C mit n Knoten und Menge P von m Punkten können alle konsistenten Abkürzungen für einen Knoten v von C in Zeit $\mathcal{O}((n+m)\log n)$ berechnet werden.

Beweisskizze.

- Schritt 1: Tangentensegmente berechnen $O(n)$
- Schritt 2: Punktzuweisung $O((n+m)\log n)$
- Schritt 3: Konsistenzberechnung $O(n \log n)$

Zusammenfassung

Lemma 6. Für x-monotonen Kantenzug C mit n Knoten und Menge P von m Punkten können alle konsistenten Abkürzungen für einen Knoten v von C in Zeit $\mathcal{O}((n+m)\log n)$ berechnet werden.

Satz. Für x-monotonen Kantenzug C mit n Knoten, Menge P von m Punkten und Fehlerschranke ε kann die kleinste konsistente Vereinfachung C' mit Fehler $\leq \varepsilon$ in Zeit $\mathcal{O}(n(n+m)\log n)$ berechnet werden.

Zusammenfassung

Lemma 6. Für x-monotonen Kantenzug C mit n Knoten und Menge P von m Punkten können alle konsistenten Abkürzungen für einen Knoten v von C in Zeit $\mathcal{O}((n+m)\log n)$ berechnet werden.

Satz. Für x-monotonen Kantenzug C mit n Knoten, Menge P von m Punkten und Fehlerschranke ε kann die kleinste konsistente Vereinfachung C' mit Fehler $\leq \varepsilon$ in Zeit $\mathcal{O}(n(n+m)\log n)$ berechnet werden.

Beweisskizze.

Zusammenfassung

Lemma 6. Für x-monotonen Kantenzug C mit n Knoten und Menge P von m Punkten können alle konsistenten Abkürzungen für einen Knoten v von C in Zeit $\mathcal{O}((n+m)\log n)$ berechnet werden.

Satz. Für x-monotonen Kantenzug C mit n Knoten, Menge P von m Punkten und Fehlerschranke ε kann die kleinste konsistente Vereinfachung C' mit Fehler $\leq \varepsilon$ in Zeit $\mathcal{O}(n(n+m)\log n)$ berechnet werden.

Beweisskizze.

- Graph $G_1 = (V, A_1)$ mit Algorithmus von Imai&Iri / Chan&Chin

Zusammenfassung

Lemma 6. Für x-monotonen Kantenzug C mit n Knoten und Menge P von m Punkten können alle konsistenten Abkürzungen für einen Knoten v von C in Zeit $\mathcal{O}((n+m)\log n)$ berechnet werden.

Satz. Für x-monotonen Kantenzug C mit n Knoten, Menge P von m Punkten und Fehlerschranke ε kann die kleinste konsistente Vereinfachung C' mit Fehler $\leq \varepsilon$ in Zeit $\mathcal{O}(n(n+m)\log n)$ berechnet werden.

Beweisskizze.

- Graph $G_1 = (V, A_1)$ mit Algorithmus von Imai&Iri / Chan&Chin $\mathcal{O}(n^2)$

Zusammenfassung

Lemma 6. Für x-monotonen Kantenzug C mit n Knoten und Menge P von m Punkten können alle konsistenten Abkürzungen für einen Knoten v von C in Zeit $\mathcal{O}((n+m)\log n)$ berechnet werden.

Satz. Für x-monotonen Kantenzug C mit n Knoten, Menge P von m Punkten und Fehlerschranke ε kann die kleinste konsistente Vereinfachung C' mit Fehler $\leq \varepsilon$ in Zeit $\mathcal{O}(n(n+m)\log n)$ berechnet werden.

Beweisskizze.

- Graph $G_1 = (V, A_1)$ mit Algorithmus von Imai&Iri / Chan&Chin $\mathcal{O}(n^2)$
- Graph $G_2 = (V, A_2)$: n-fache Anwendung von Lemma 6

Zusammenfassung

Lemma 6. Für x-monotonen Kantenzug C mit n Knoten und Menge P von m Punkten können alle konsistenten Abkürzungen für einen Knoten v von C in Zeit $\mathcal{O}((n+m)\log n)$ berechnet werden.

Satz. Für x-monotonen Kantenzug C mit n Knoten, Menge P von m Punkten und Fehlerschranke ε kann die kleinste konsistente Vereinfachung C' mit Fehler $\leq \varepsilon$ in Zeit $\mathcal{O}(n(n+m)\log n)$ berechnet werden.

Beweisskizze.

- Graph $G_1 = (V, A_1)$ mit Algorithmus von Imai&Iri / Chan&Chin $\mathcal{O}(n^2)$
- Graph $G_2 = (V, A_2)$: n-fache Anwendung von Lemma 6 $\mathcal{O}(n(n+m)\log n)$

Zusammenfassung

Lemma 6. Für x-monotonen Kantenzug C mit n Knoten und Menge P von m Punkten können alle konsistenten Abkürzungen für einen Knoten v von C in Zeit $\mathcal{O}((n+m)\log n)$ berechnet werden.

Satz. Für x-monotonen Kantenzug C mit n Knoten, Menge P von m Punkten und Fehlerschranke ε kann die kleinste konsistente Vereinfachung C' mit Fehler $\leq \varepsilon$ in Zeit $\mathcal{O}(n(n+m)\log n)$ berechnet werden.

Beweisskizze.

- Graph $G_1 = (V, A_1)$ mit Algorithmus von Imai&Iri / Chan&Chin $\mathcal{O}(n^2)$
- Graph $G_2 = (V, A_2)$: n-fache Anwendung von Lemma 6 $\mathcal{O}(n(n+m)\log n)$
- Graph $G = (V, A_1 \cap A_2)$ bilden

Zusammenfassung

Lemma 6. Für x-monotonen Kantenzug C mit n Knoten und Menge P von m Punkten können alle konsistenten Abkürzungen für einen Knoten v von C in Zeit $\mathcal{O}((n+m)\log n)$ berechnet werden.

Satz. Für x-monotonen Kantenzug C mit n Knoten, Menge P von m Punkten und Fehlerschranke ε kann die kleinste konsistente Vereinfachung C' mit Fehler $\leq \varepsilon$ in Zeit $\mathcal{O}(n(n+m)\log n)$ berechnet werden.

Beweisskizze.

- Graph $G_1 = (V, A_1)$ mit Algorithmus von Imai&Iri / Chan&Chin $\mathcal{O}(n^2)$
- Graph $G_2 = (V, A_2)$: n-fache Anwendung von Lemma 6 $\mathcal{O}(n(n+m)\log n)$
- Graph $G = (V, A_1 \cap A_2)$ bilden $\mathcal{O}(n^2)$

Zusammenfassung

Lemma 6. Für x-monotonen Kantenzug C mit n Knoten und Menge P von m Punkten können alle konsistenten Abkürzungen für einen Knoten v von C in Zeit $\mathcal{O}((n+m)\log n)$ berechnet werden.

Satz. Für x-monotonen Kantenzug C mit n Knoten, Menge P von m Punkten und Fehlerschranke ε kann die kleinste konsistente Vereinfachung C' mit Fehler $\leq \varepsilon$ in Zeit $\mathcal{O}(n(n+m)\log n)$ berechnet werden.

Beweisskizze.

- Graph $G_1 = (V, A_1)$ mit Algorithmus von Imai&Iri / Chan&Chin $\mathcal{O}(n^2)$
- Graph $G_2 = (V, A_2)$: n-fache Anwendung von Lemma 6 $\mathcal{O}(n(n+m)\log n)$
- Graph $G = (V, A_1 \cap A_2)$ bilden $\mathcal{O}(n^2)$
- kürzesten $v_1 v_n$-Pfad in G suchen

Zusammenfassung

Lemma 6. Für x-monotonen Kantenzug C mit n Knoten und Menge P von m Punkten können alle konsistenten Abkürzungen für einen Knoten v von C in Zeit $\mathcal{O}((n+m)\log n)$ berechnet werden.

Satz. Für x-monotonen Kantenzug C mit n Knoten, Menge P von m Punkten und Fehlerschranke ε kann die kleinste konsistente Vereinfachung C' mit Fehler $\leq \varepsilon$ in Zeit $\mathcal{O}(n(n+m)\log n)$ berechnet werden.

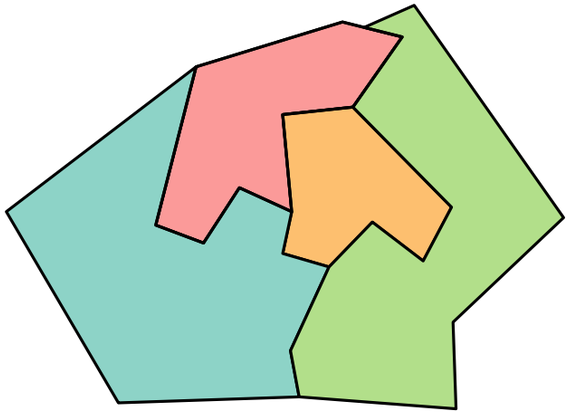
Beweisskizze.

- Graph $G_1 = (V, A_1)$ mit Algorithmus von Imai&Iri / Chan&Chin $\mathcal{O}(n^2)$
- Graph $G_2 = (V, A_2)$: n-fache Anwendung von Lemma 6 $\mathcal{O}(n(n+m)\log n)$
- Graph $G = (V, A_1 \cap A_2)$ bilden $\mathcal{O}(n^2)$
- kürzesten $v_1 v_n$-Pfad in G suchen $\mathcal{O}(n^2)$

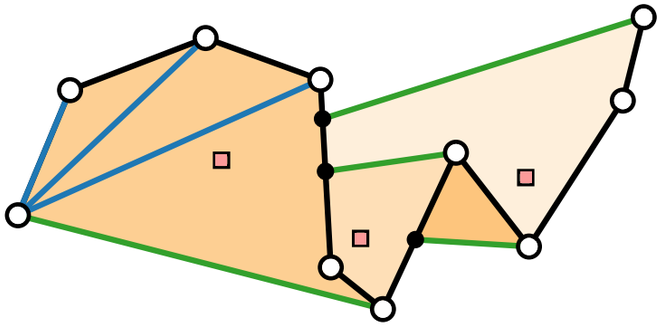
Universität Trier

Algorithmen für geographische Informationssysteme

10. Vorlesung
Flächenvereinfachung



Teil IV:
Allgemeine Linienzüge



Philipp Kindermann

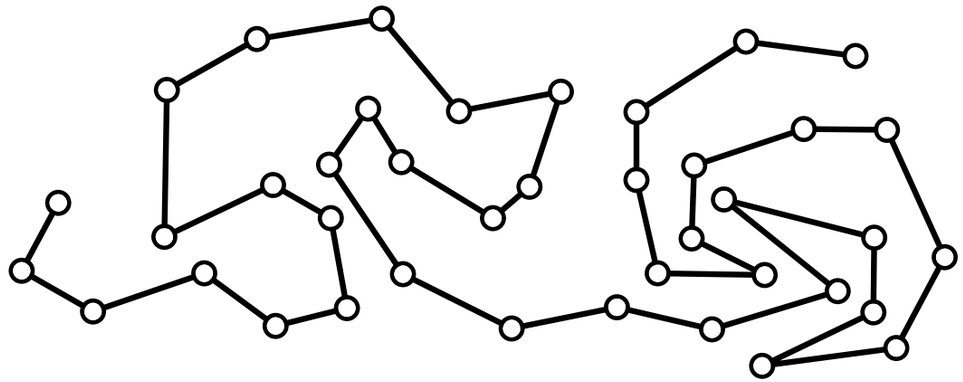
Wintersemester 2024/25

Allgemeine Linienzüge

Was machen wir, wenn C **nicht** x-monoton ist?

Allgemeine Linienzüge

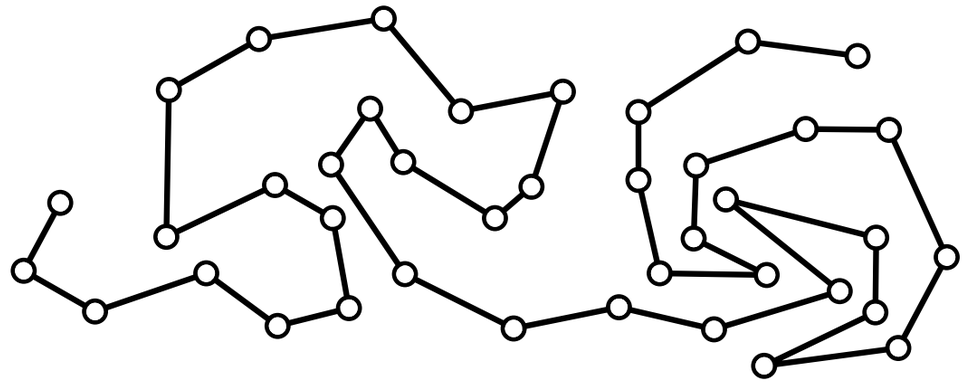
Was machen wir, wenn C **nicht** x-monoton ist?



Allgemeine Linienzüge

Was machen wir, wenn C **nicht** x-monoton ist?

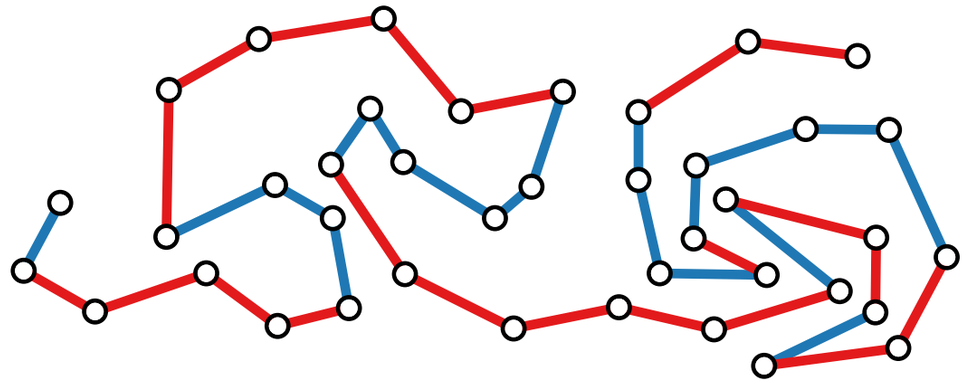
- Unterteile C in x-monotone Linienzüge



Allgemeine Linienzüge

Was machen wir, wenn C **nicht** x-monoton ist?

- Unterteile C in x-monotone Linienzüge

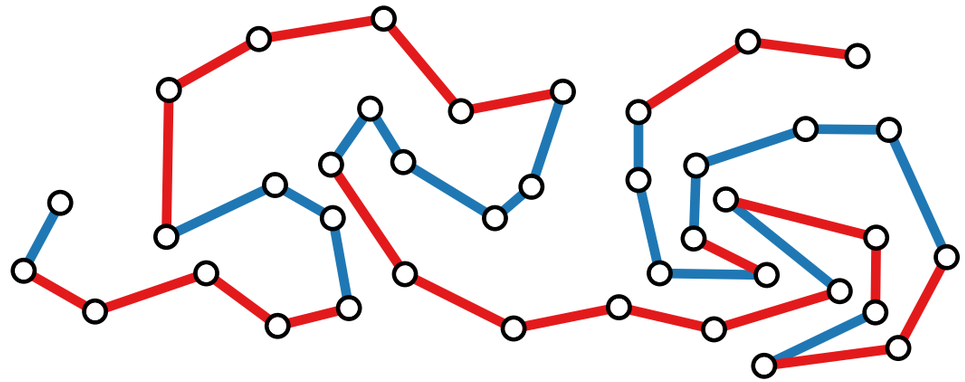


Allgemeine Linienzüge

Was machen wir, wenn C **nicht** x-monoton ist?

- Unterteile C in x-monotone Linienzüge

Problem?

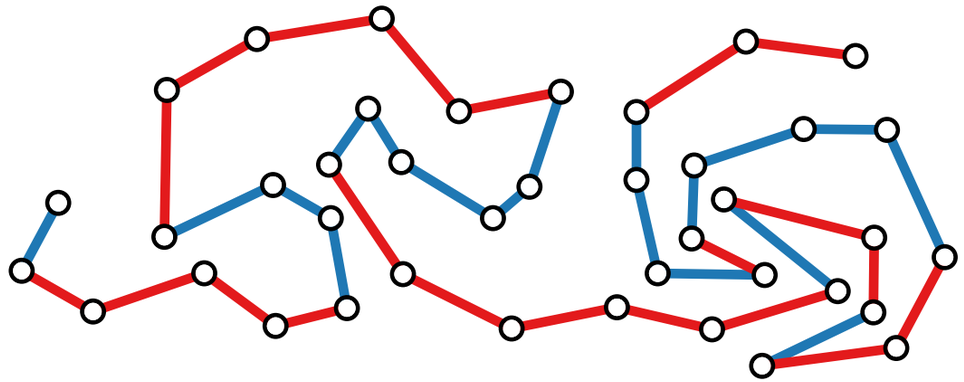


Allgemeine Linienzüge

Was machen wir, wenn C **nicht** x-monoton ist?

- Unterteile C in x-monotone Linienzüge

Problem? Könnte sehr viele Teile geben, wenn C viel links-rechts geht.



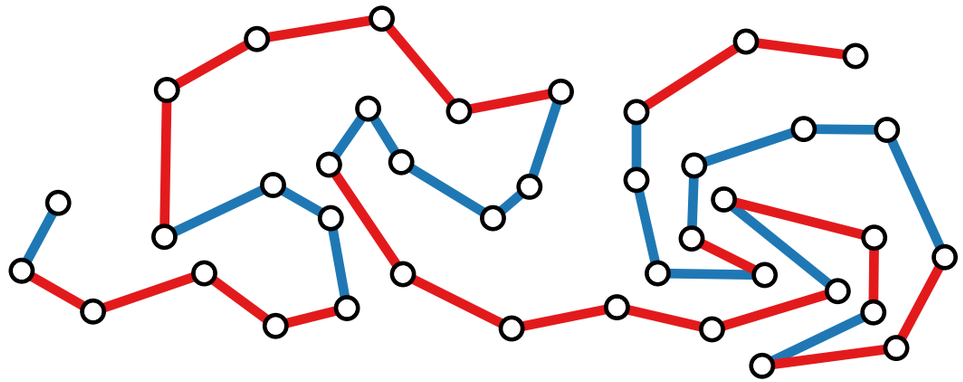
Allgemeine Linienzüge

Was machen wir, wenn C **nicht** x-monoton ist?

- Unterteile C in x-monotone Linienzüge

Problem? Könnte sehr viele Teile geben, wenn C viel links-rechts geht.

Verbesserung?



Allgemeine Linienzüge

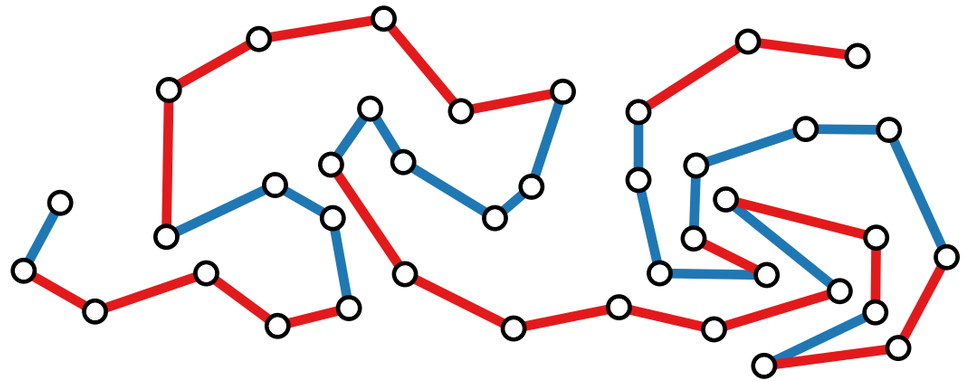
Was machen wir, wenn C **nicht** x-monoton ist?

- Unterteile C in x-monotone Linienzüge

Problem? Könnte sehr viele Teile geben, wenn C viel links-rechts geht.

Verbesserung?

- Unterteile C in θ-monotone Linienzüge (θ beliebige Richtung)



Allgemeine Linienzüge

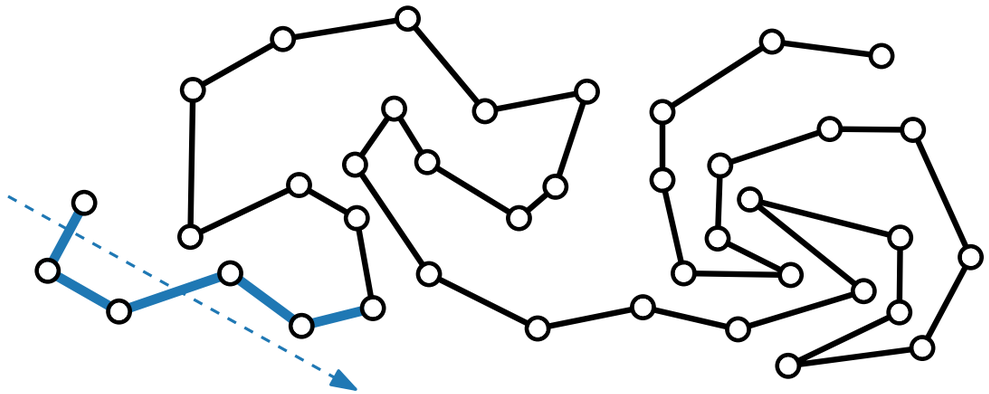
Was machen wir, wenn C **nicht** x-monoton ist?

- Unterteile C in x-monotone Linienzüge

Problem? Könnte sehr viele Teile geben, wenn C viel links-rechts geht.

Verbesserung?

- Unterteile C in θ-monotone Linienzüge (θ beliebige Richtung)



Allgemeine Linienzüge

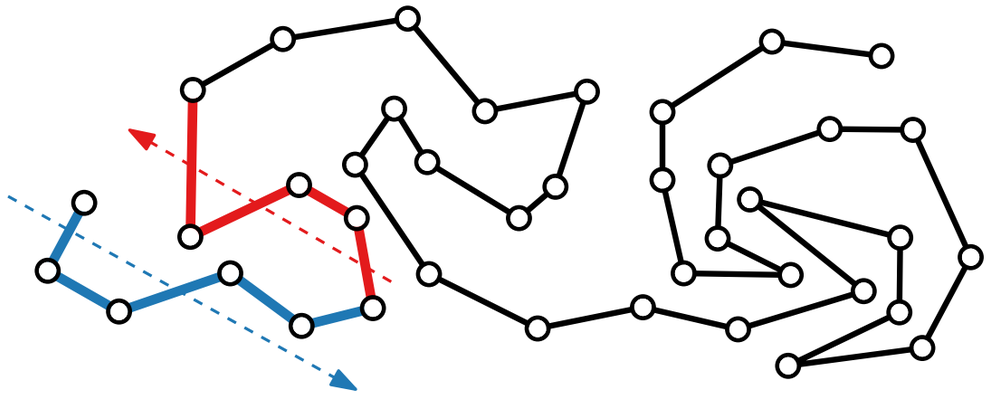
Was machen wir, wenn C **nicht** x-monoton ist?

- Unterteile C in x-monotone Linienzüge

Problem? Könnte sehr viele Teile geben, wenn C viel links-rechts geht.

Verbesserung?

- Unterteile C in θ-monotone Linienzüge (θ beliebige Richtung)



Allgemeine Linienzüge

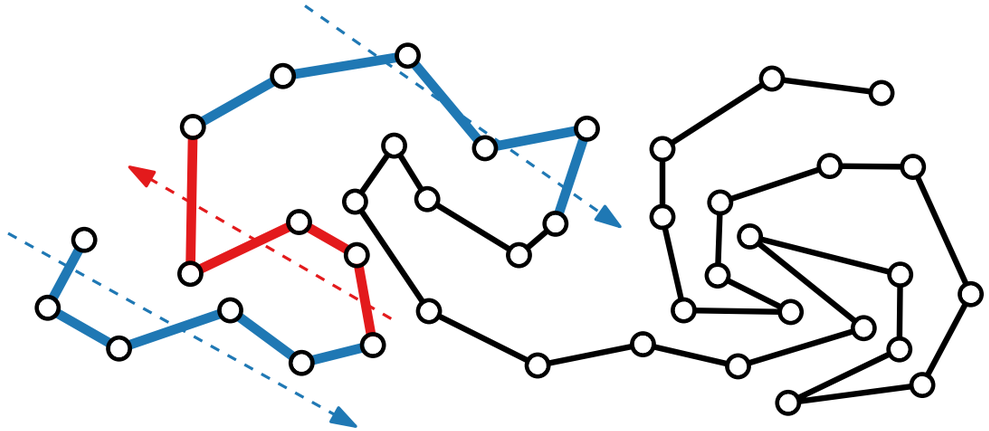
Was machen wir, wenn C **nicht** x-monoton ist?

- Unterteile C in x-monotone Linienzüge

Problem? Könnte sehr viele Teile geben, wenn C viel links-rechts geht.

Verbesserung?

- Unterteile C in θ-monotone Linienzüge (θ beliebige Richtung)



Allgemeine Linienzüge

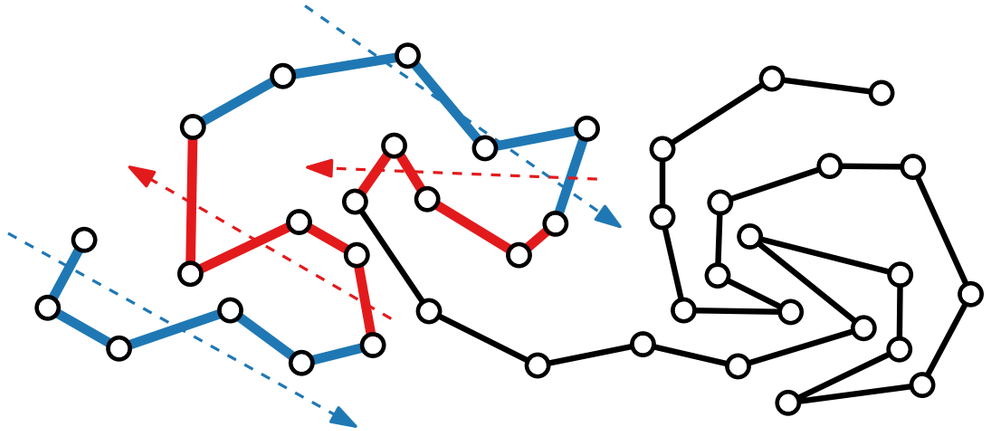
Was machen wir, wenn C **nicht** x-monoton ist?

- Unterteile C in x-monotone Linienzüge

Problem? Könnte sehr viele Teile geben, wenn C viel links-rechts geht.

Verbesserung?

- Unterteile C in θ-monotone Linienzüge (θ beliebige Richtung)



Allgemeine Linienzüge

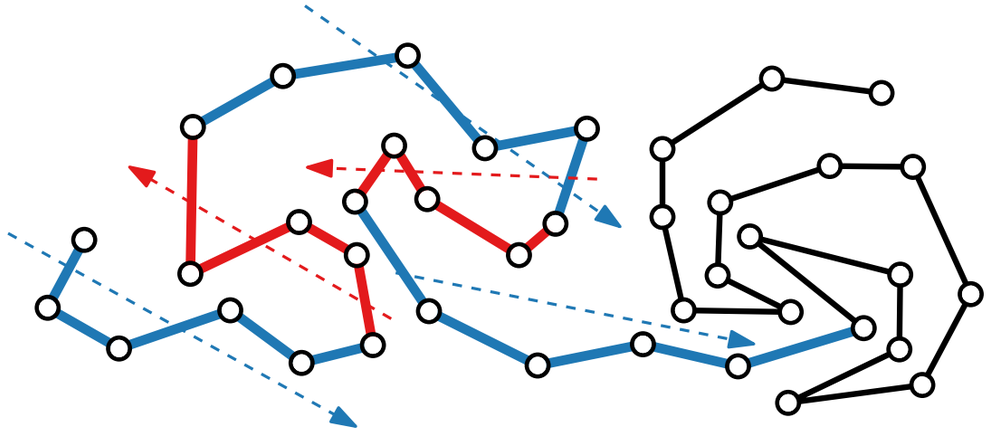
Was machen wir, wenn C **nicht** x-monoton ist?

- Unterteile C in x-monotone Linienzüge

Problem? Könnte sehr viele Teile geben, wenn C viel links-rechts geht.

Verbesserung?

- Unterteile C in θ-monotone Linienzüge (θ beliebige Richtung)



Allgemeine Linienzüge

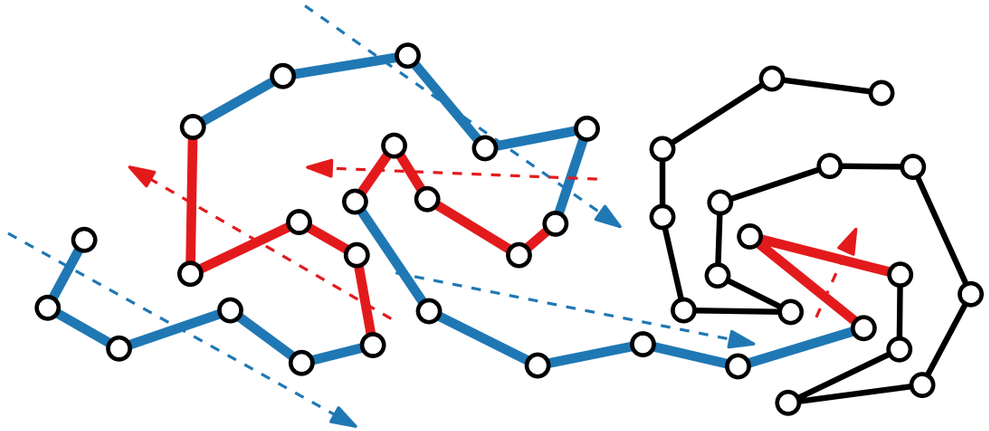
Was machen wir, wenn C **nicht** x-monoton ist?

- Unterteile C in x-monotone Linienzüge

Problem? Könnte sehr viele Teile geben, wenn C viel links-rechts geht.

Verbesserung?

- Unterteile C in θ-monotone Linienzüge (θ beliebige Richtung)



Allgemeine Linienzüge

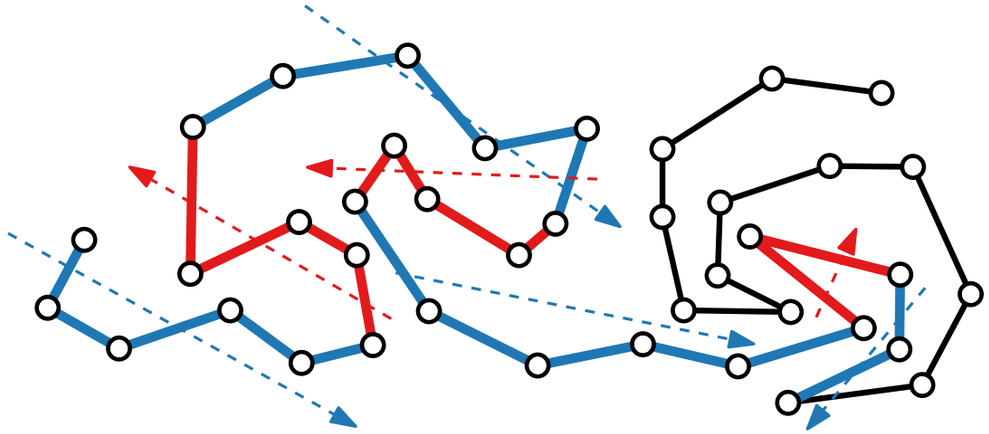
Was machen wir, wenn C **nicht** x-monoton ist?

- Unterteile C in x-monotone Linienzüge

Problem? Könnte sehr viele Teile geben, wenn C viel links-rechts geht.

Verbesserung?

- Unterteile C in θ-monotone Linienzüge (θ beliebige Richtung)



Allgemeine Linienzüge

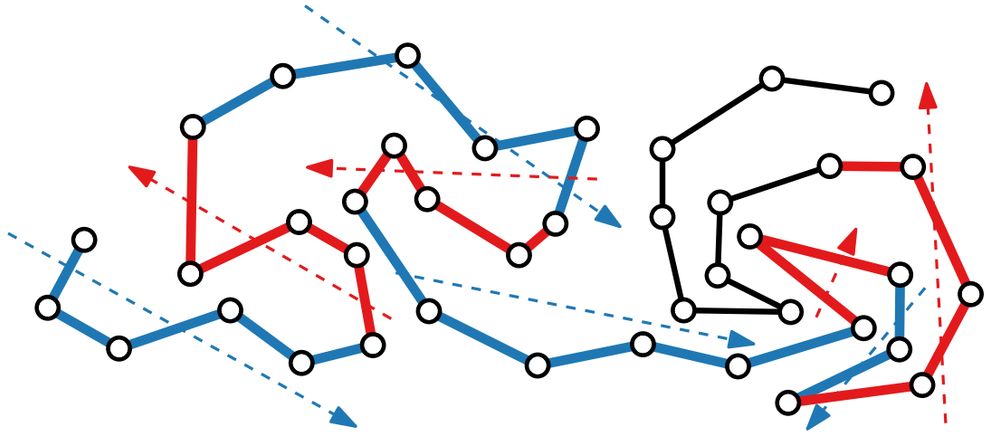
Was machen wir, wenn C **nicht** x-monoton ist?

- Unterteile C in x-monotone Linienzüge

Problem? Könnte sehr viele Teile geben, wenn C viel links-rechts geht.

Verbesserung?

- Unterteile C in θ-monotone Linienzüge (θ beliebige Richtung)



Allgemeine Linienzüge

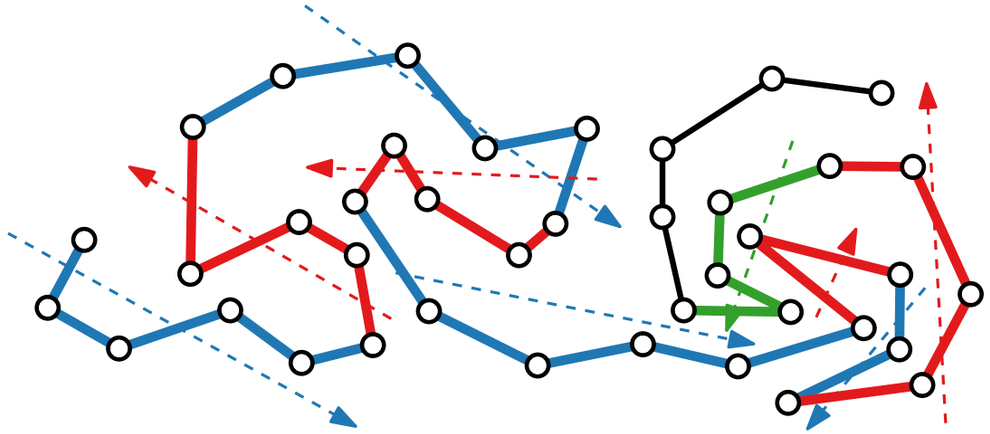
Was machen wir, wenn C **nicht** x-monoton ist?

- Unterteile C in x-monotone Linienzüge

Problem? Könnte sehr viele Teile geben, wenn C viel links-rechts geht.

Verbesserung?

- Unterteile C in θ-monotone Linienzüge (θ beliebige Richtung)



Allgemeine Linienzüge

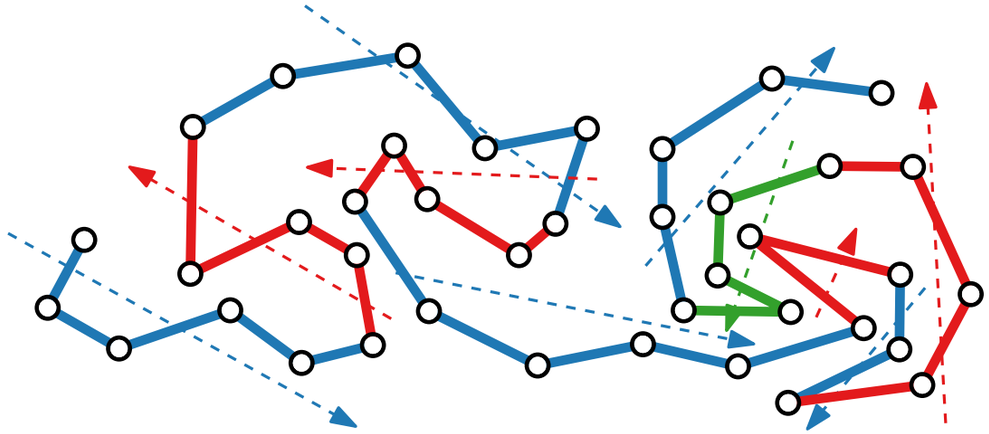
Was machen wir, wenn C **nicht** x-monoton ist?

- Unterteile C in x-monotone Linienzüge

Problem? Könnte sehr viele Teile geben, wenn C viel links-rechts geht.

Verbesserung?

- Unterteile C in θ-monotone Linienzüge (θ beliebige Richtung)



Allgemeine Linienzüge

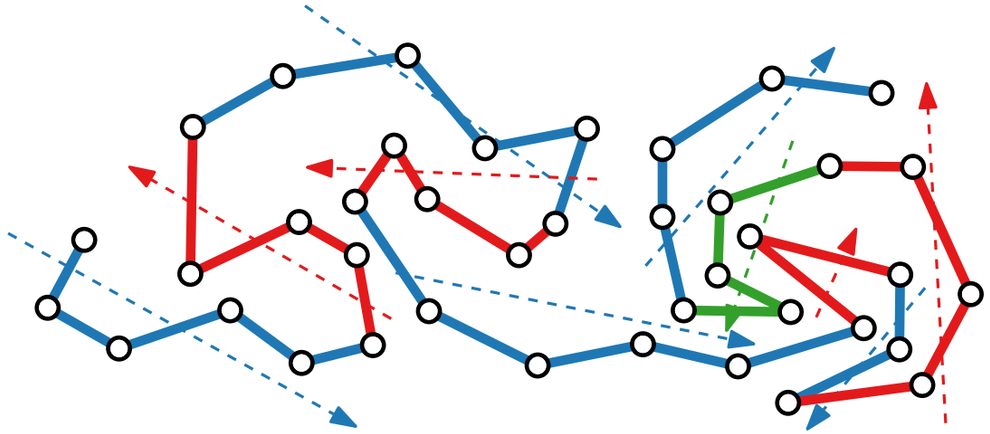
Was machen wir, wenn C **nicht** x-monoton ist?

- Unterteile C in x-monotone Linienzüge

Problem? Könnte sehr viele Teile geben, wenn C viel links-rechts geht.

Verbesserung?

- Unterteile C in θ-monotone Linienzüge (θ beliebige Richtung)
- Noch besser: Finde für jeden Knoten v_i den Knoten $v_{n(i)}$, bis zu dem der Algorithmus angewendet werden kann



Allgemeine Linienzüge

Was machen wir, wenn C **nicht** x-monoton ist?

- Unterteile C in x-monotone Linienzüge

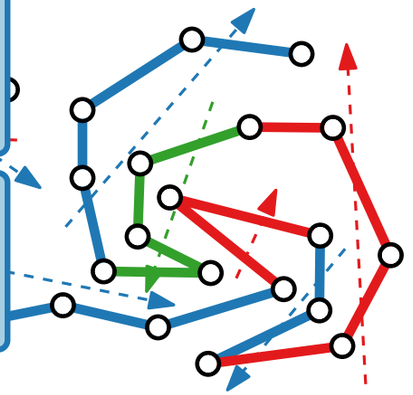
Problem? Könnte sehr viele Teile geben, wenn C viel links-rechts geht.

Verbesserung?

- Unterteile C in θ-monotone Linienzüge (θ beliebige Richtung)
- Noch besser: Finde für jeden Knoten v_i den Knoten $v_{n(i)}$, bis zu dem der Algorithmus angewendet werden kann
  - ➡ Dafür müssen Lemma 2 und 4 anwendbar sein.

Lemma 2. Jede begrenzte Facette f von S_i ist **θ-monoton** bzgl. v_i, d.h. $|r \cap f| \in \{0, 1\}$ für jeden Strahl r von v_i

Lemma 4. Jeder Strahl von v_i schneidet die Facetten von S_i in aufsteigender Reihenfolge



Allgemeine Linienzüge

Was machen wir, wenn C **nicht** x-monoton ist?

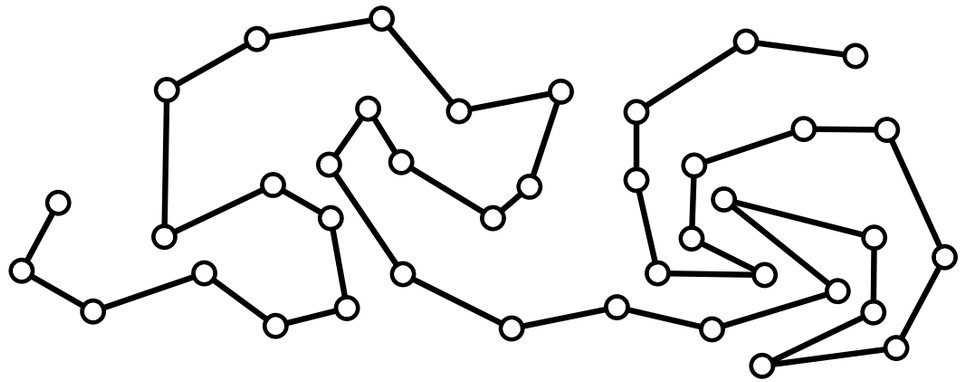
- Unterteile C in x-monotone Linienzüge

Problem? Könnte sehr viele Teile geben, wenn C viel links-rechts geht.

Verbesserung?

- Unterteile C in θ-monotone Linienzüge (θ beliebige Richtung)
- Noch besser: Finde für jeden Knoten v_i den Knoten $v_{n(i)}$, bis zu dem der Algorithmus angewendet werden kann
  - ➡ Dafür müssen Lemma 2 und 4 anwendbar sein.

Anforderungen.



Allgemeine Linienzüge

Was machen wir, wenn C **nicht** x-monoton ist?

- Unterteile C in x-monotone Linienzüge

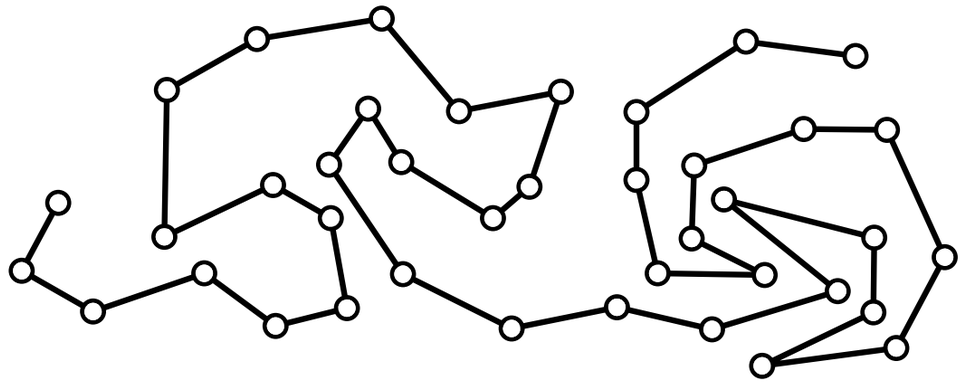
Problem? Könnte sehr viele Teile geben, wenn C viel links-rechts geht.

Verbesserung?

- Unterteile C in θ-monotone Linienzüge (θ beliebige Richtung)
- Noch besser: Finde für jeden Knoten v_i den Knoten $v_{n(i)}$, bis zu dem der Algorithmus angewendet werden kann
  - ➡ Dafür müssen Lemma 2 und 4 anwendbar sein.

Anforderungen.

- Linienzug bildet umkreist v_i nicht.



Allgemeine Linienzüge

Was machen wir, wenn C **nicht** x-monoton ist?

- Unterteile C in x-monotone Linienzüge

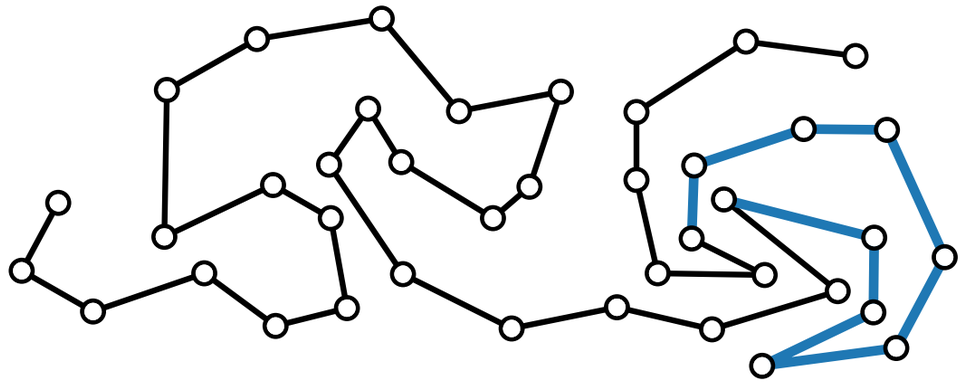
Problem? Könnte sehr viele Teile geben, wenn C viel links-rechts geht.

Verbesserung?

- Unterteile C in θ-monotone Linienzüge (θ beliebige Richtung)
- Noch besser: Finde für jeden Knoten v_i den Knoten $v_{n(i)}$, bis zu dem der Algorithmus angewendet werden kann
  - ➡ Dafür müssen Lemma 2 und 4 anwendbar sein.

Anforderungen.

- Linienzug bildet umkreist v_i nicht.



Allgemeine Linienzüge

Was machen wir, wenn C **nicht** x-monoton ist?

- Unterteile C in x-monotone Linienzüge

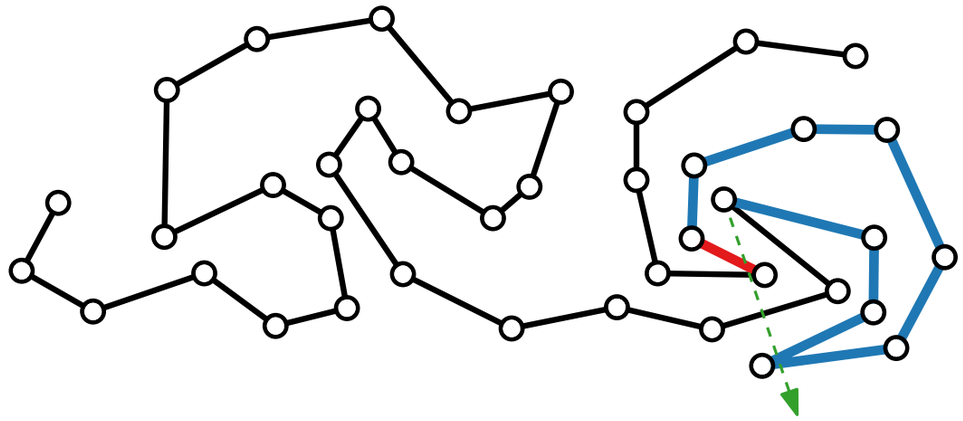
Problem? Könnte sehr viele Teile geben, wenn C viel links-rechts geht.

Verbesserung?

- Unterteile C in θ-monotone Linienzüge (θ beliebige Richtung)
- Noch besser: Finde für jeden Knoten v_i den Knoten $v_{n(i)}$, bis zu dem der Algorithmus angewendet werden kann
  - ➡ Dafür müssen Lemma 2 und 4 anwendbar sein.

Anforderungen.

- Linienzug bildet umkreist v_i nicht.



Allgemeine Linienzüge

Was machen wir, wenn C **nicht** x-monoton ist?

- Unterteile C in x-monotone Linienzüge

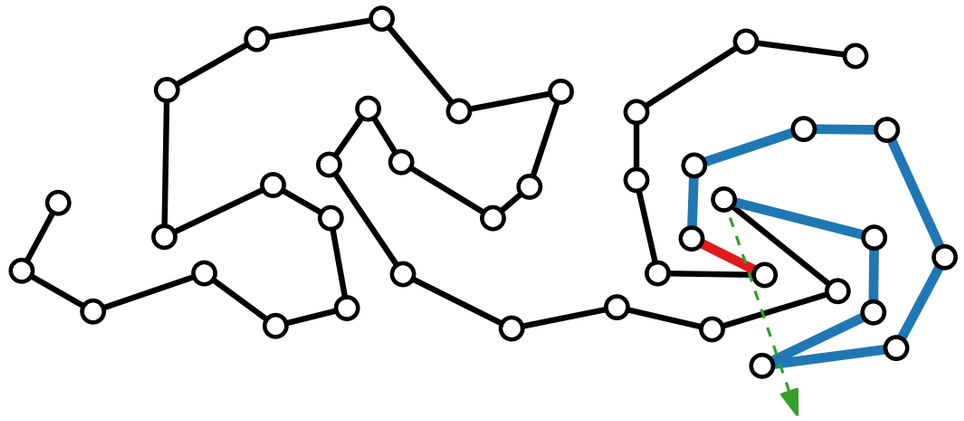
Problem? Könnte sehr viele Teile geben, wenn C viel links-rechts geht.

Verbesserung?

- Unterteile C in θ-monotone Linienzüge (θ beliebige Richtung)
- Noch besser: Finde für jeden Knoten v_i den Knoten $v_{n(i)}$, bis zu dem der Algorithmus angewendet werden kann
  - ➡ Dafür müssen Lemma 2 und 4 anwendbar sein.

Anforderungen.

- Linienzug bildet umkreist v_i nicht.
- Es gibt keine **Rückwärtstangente**:
  Eine Kante (v_j, v_{j+1}), die näher auf einem Strahl von v_i ist, als (v_{j-1}, v_j).



Allgemeine Linienzüge

Was machen wir, wenn C **nicht** x-monoton ist?

- Unterteile C in x-monotone Linienzüge

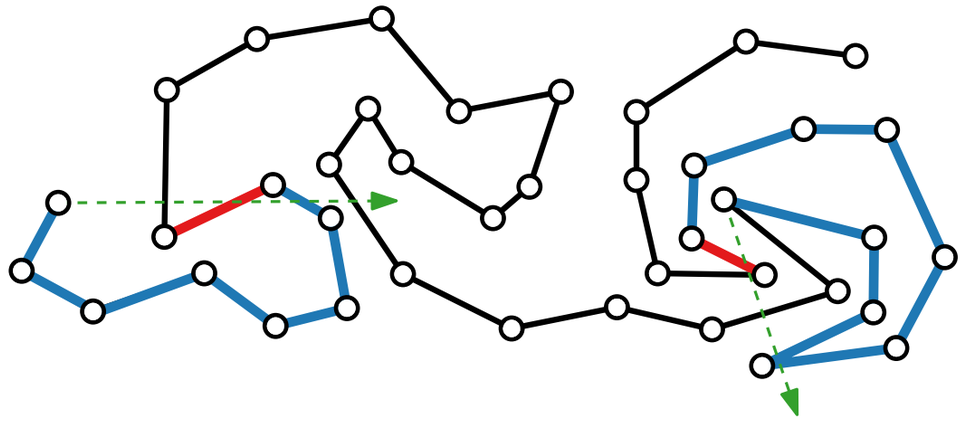
Problem? Könnte sehr viele Teile geben, wenn C viel links-rechts geht.

Verbesserung?

- Unterteile C in θ-monotone Linienzüge (θ beliebige Richtung)
- Noch besser: Finde für jeden Knoten v_i den Knoten $v_{n(i)}$, bis zu dem der Algorithmus angewendet werden kann
  - ➡ Dafür müssen Lemma 2 und 4 anwendbar sein.

Anforderungen.

- Linienzug bildet umkreist v_i nicht.
- Es gibt keine **Rückwärtstangente**:
  Eine Kante (v_j, v_{j+1}), die näher auf einem Strahl von v_i ist, als (v_{j-1}, v_j).



Allgemeine Linienzüge

Was machen wir, wenn C **nicht** x-monoton ist?

- Unterteile C in x-monotone Linienzüge

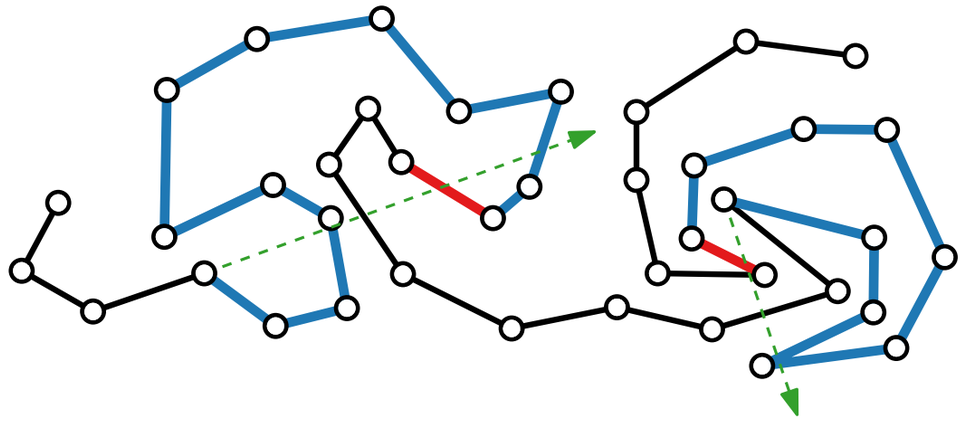
Problem? Könnte sehr viele Teile geben, wenn C viel links-rechts geht.

Verbesserung?

- Unterteile C in θ-monotone Linienzüge (θ beliebige Richtung)
- Noch besser: Finde für jeden Knoten v_i den Knoten $v_{n(i)}$, bis zu dem der Algorithmus angewendet werden kann
  - ➡ Dafür müssen Lemma 2 und 4 anwendbar sein.

Anforderungen.

- Linienzug bildet umkreist v_i nicht.
- Es gibt keine **Rückwärtstangente**:
  Eine Kante (v_j, v_{j+1}), die näher auf einem Strahl von v_i ist, als (v_{j-1}, v_j).



Allgemeine Linienzüge

Was machen wir, wenn C **nicht** x-monoton ist?

- Unterteile C in x-monotone Linienzüge

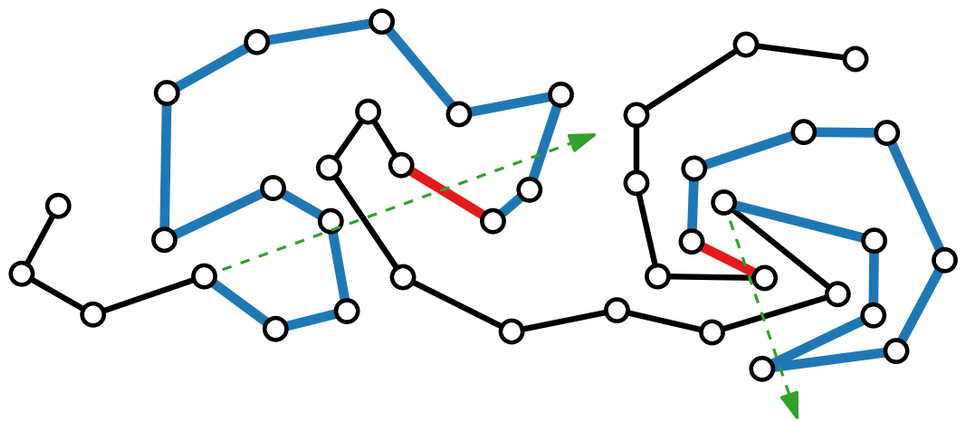
Problem? Könnte sehr viele Teile geben, wenn C viel links-rechts geht.

Verbesserung?

- Unterteile C in θ-monotone Linienzüge (θ beliebige Richtung)
- Noch besser: Finde für jeden Knoten v_i den Knoten $v_{n(i)}$, bis zu dem der Algorithmus angewendet werden kann
  - ➡ Dafür müssen Lemma 2 und 4 anwendbar sein.

Anforderungen.

- Linienzug bildet umkreist v_i nicht.
- Es gibt keine **Rückwärtstangente**:
  Eine Kante (v_j, v_{j+1}), die näher auf einem Strahl von v_i ist, als (v_{j-1}, v_j).



Laufzeit?

Allgemeine Linienzüge

Was machen wir, wenn C **nicht** x-monoton ist?

- Unterteile C in x-monotone Linienzüge

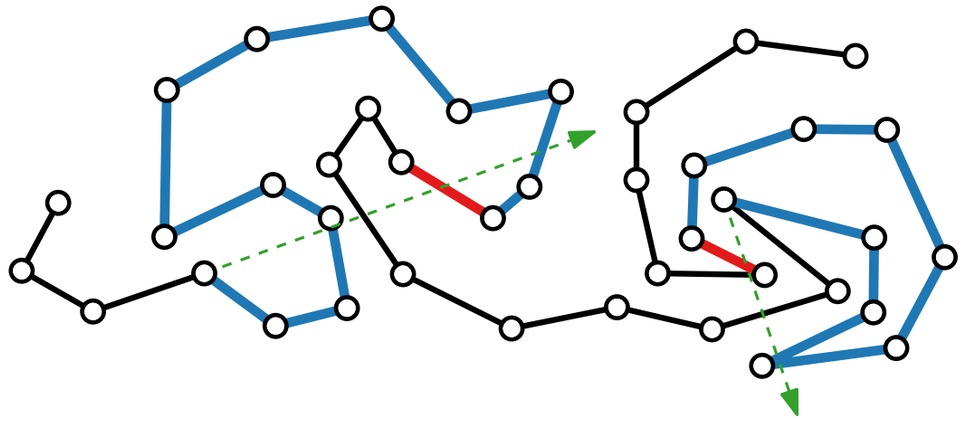
Problem? Könnte sehr viele Teile geben, wenn C viel links-rechts geht.

Verbesserung?

- Unterteile C in θ-monotone Linienzüge (θ beliebige Richtung)
- Noch besser: Finde für jeden Knoten v_i den Knoten $v_{n(i)}$, bis zu dem der Algorithmus angewendet werden kann
  - ➡ Dafür müssen Lemma 2 und 4 anwendbar sein.

Anforderungen.

- Linienzug bildet umkreist v_i nicht.
- Es gibt keine **Rückwärtstangente**:
  Eine Kante (v_j, v_{j+1}), die näher auf einem Strahl von v_i ist, als (v_{j-1}, v_j).



Laufzeit? Unterteilung in $\mathcal{O}(n)$ Zeit.

Allgemeine Linienzüge

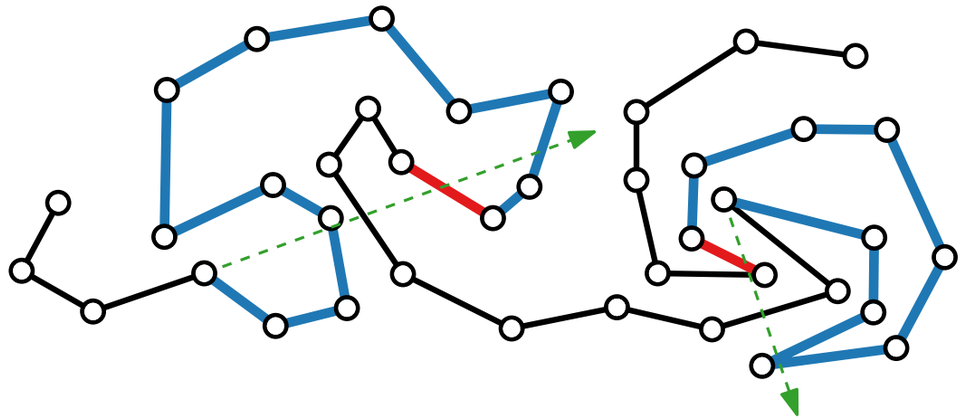
Was machen wir, wenn C **nicht** x-monoton ist?

Satz. Für Kantenzug C mit n Knoten, Menge P von m Punkten und Fehlerschranke ε kann die kleinste konsistente Vereinfachung C' mit Fehler $\leq \varepsilon$ in Zeit $\mathcal{O}(n(n+m)\log n)$ berechnet werden.

- Unterteile C in θ-monotone Linienzüge (θ beliebige Richtung)
- Noch besser: Finde für jeden Knoten v_i den Knoten $v_{n(i)}$, bis zu dem der Algorithmus angewendet werden kann
  - ➡ Dafür müssen Lemma 2 und 4 anwendbar sein.

Anforderungen.

- Linienzug bildet umkreist v_i nicht.
- Es gibt keine **Rückwärtstangente**:
  Eine Kante (v_j, v_{j+1}), die näher auf einem Strahl von v_i ist, als (v_{j-1}, v_j).



Laufzeit? Unterteilung in $\mathcal{O}(n)$ Zeit.

Zusammenfassung Linienvereinfachung

- typische Qualitätskriterien: Abstandsmaß, Anzahl Segmente

Zusammenfassung Linienvereinfachung

- typische Qualitätskriterien: Abstandsmaß, Anzahl Segmente
- einfache Greedy-Algorithmen

Zusammenfassung Linienvereinfachung

- typische Qualitätskriterien: Abstandsmaß, Anzahl Segmente
- einfache Greedy-Algorithmen
- Douglas-Peucker Algorithmus (und Beschleunigung)

Zusammenfassung Linienvereinfachung

- typische Qualitätskriterien: Abstandsmaß, Anzahl Segmente
- einfache Greedy-Algorithmen
- Douglas-Peucker Algorithmus (und Beschleunigung)
- Dreiecksbasiertes Verfahren von Visvalingam&Whyatt

Zusammenfassung Linienvereinfachung

- typische Qualitätskriterien: Abstandsmaß, Anzahl Segmente
- einfache Greedy-Algorithmen
- Douglas-Peucker Algorithmus (und Beschleunigung)
- Dreiecksbasiertes Verfahren von Visvalingam&Whyatt
- exakte Optimierung als kürzeste-Wege-Problem in Graphen

Zusammenfassung Linienvereinfachung

- typische Qualitätskriterien: Abstandsmaß, Anzahl Segmente
- einfache Greedy-Algorithmen
- Douglas-Peucker Algorithmus (und Beschleunigung)
- Dreiecksbasiertes Verfahren von Visvalingam&Whyatt
- exakte Optimierung als kürzeste-Wege-Problem in Graphen
- topologisch konsistente Vereinfachung von de Berg et al.

Zusammenfassung Linienvereinfachung

- typische Qualitätskriterien: Abstandsmaß, Anzahl Segmente
- einfache Greedy-Algorithmen
- Douglas-Peucker Algorithmus (und Beschleunigung)
- Dreiecksbasiertes Verfahren von Visvalingam&Whyatt
- exakte Optimierung als kürzeste-Wege-Problem in Graphen
- topologisch konsistente Vereinfachung von de Berg et al.
- de Berg et al. (1998) stellen noch weitere Anpassungen für einige praktisch relevante Fälle vor (z.B. nicht x-monoton)

Zusammenfassung Linienvereinfachung

- typische Qualitätskriterien: Abstandsmaß, Anzahl Segmente
- einfache Greedy-Algorithmen
- Douglas-Peucker Algorithmus (und Beschleunigung)
- Dreiecksbasiertes Verfahren von Visvalingam&Whyatt
- exakte Optimierung als kürzeste-Wege-Problem in Graphen
- topologisch konsistente Vereinfachung von de Berg et al.
- de Berg et al. (1998) stellen noch weitere Anpassungen für einige praktisch relevante Fälle vor (z.B. nicht x-monoton)
- für Linienvereinfachung unter Hinzunahme zusätzlicher Knoten ist die Segmentminimierung NP-schwer [Guibas et al. '93]