Liniensegmentschnitt

Motivation, Überlagerung von Karten,
Problemformulierung

Ein einfaches Problem und dessen Lösung mit Hilfe des Sweep-Line Prinzips

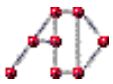
Output-sensitiver Liniensegmentschnittalgorithmus

Doppelt verkettete Kantenliste

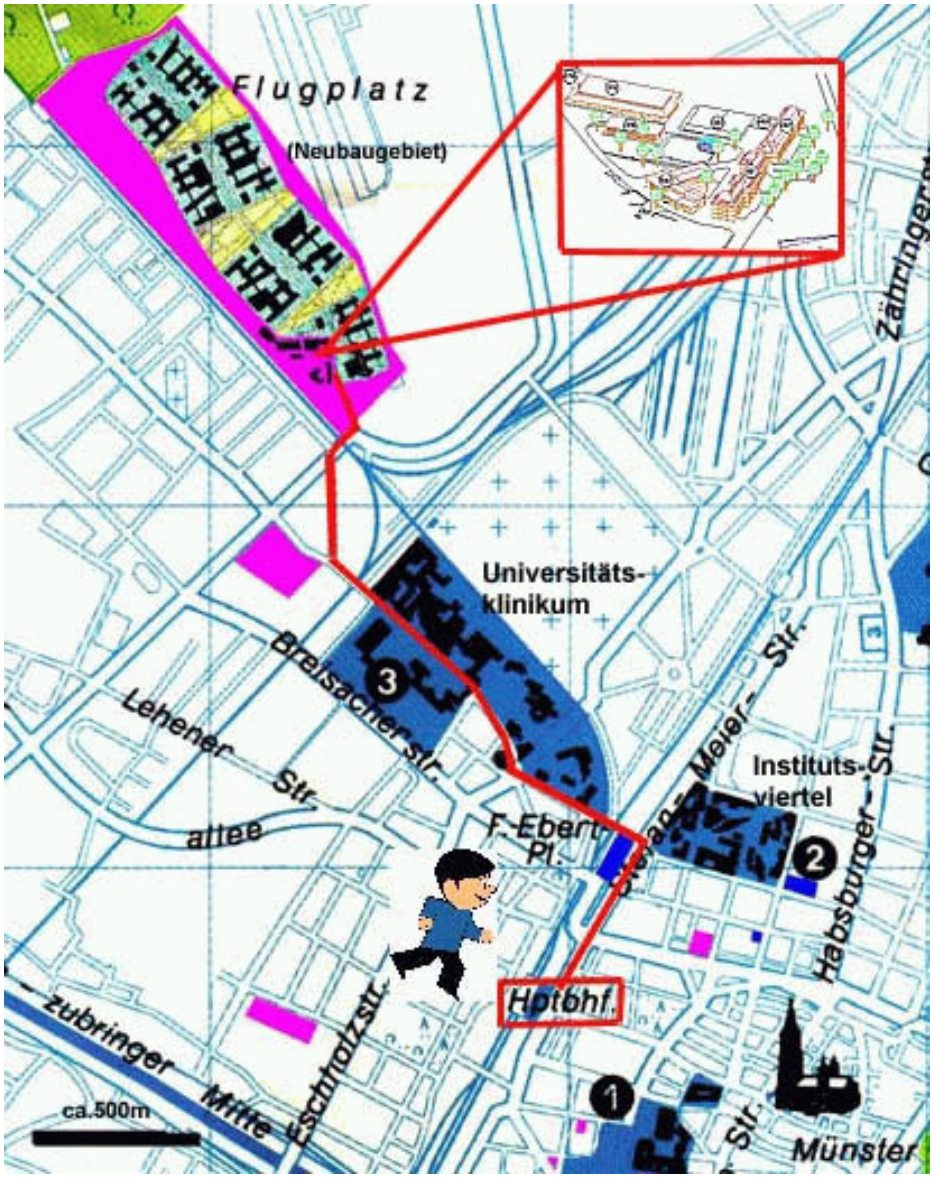
Überlagerung von 2 ebenen Graphen

Boolsche Operatoren für einfache Polygone

(LEDA)



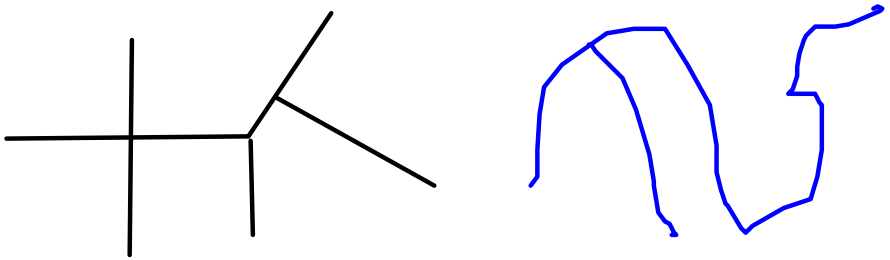
Karten



unentbehrlich, aber auch frustrierend

Segmentschnittproblem

Kartenüberlappung in Geographischen IS



Straßenkarte

Fluß

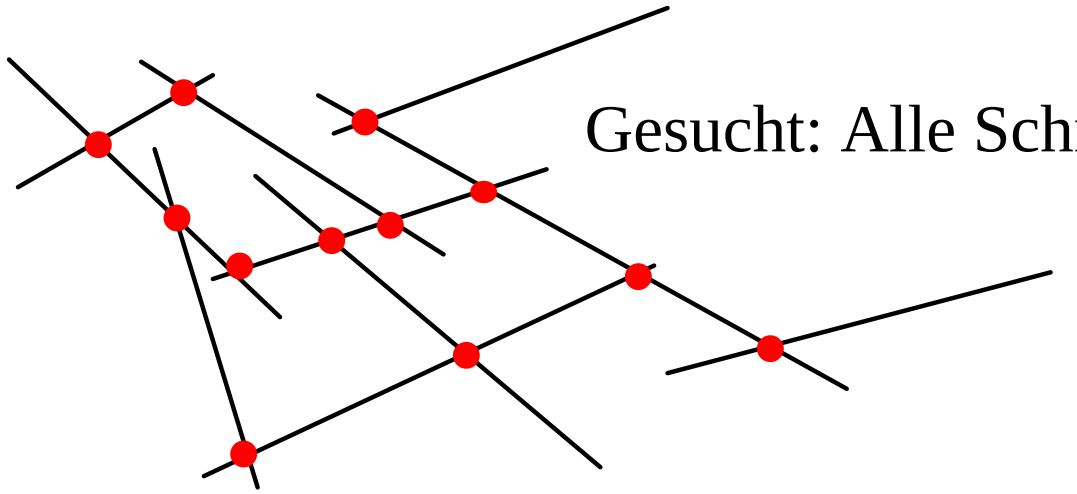
Kombinierte Karte

1. Schnittpunkte liefern wichtige Informationen.
2. Duch Diskretisierungen kann ein Fluß als abschnittsweise linear angesehen werden.

"Naives" Verfahren

Problemstellung:

S={s1,...,sn} Segmente, si = {(x1,y1),(x2,y2)}



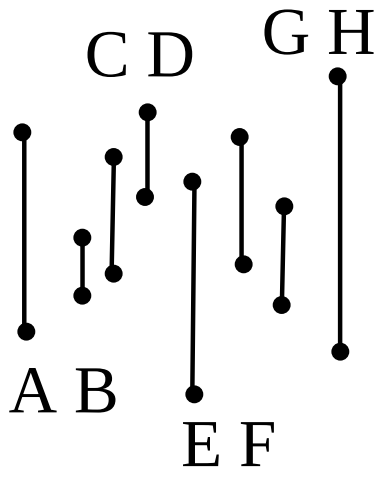
Gesucht: Alle Schnittpunkte.

Der Schnitt zweier Linien läßt sich in O(1) berechnen.

Trivialer Ansatz liefert O(n^2) Vergleiche.

Ein einfacheres Problem

Sichtbarkeit

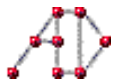


Geg: Menge vertikaler Segmente

Ges: Alle Paare gegenseitig sichtbarer Segmente

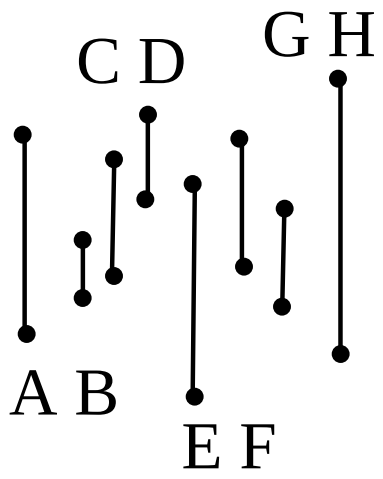
Auch hier: Naives Verfahren in O(n^2)

Beobachtung: 2 Segmente s und s' gegenseitig sichtbar
<=> ex. y mit s und s' unmittelbar benachbart



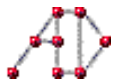
Algorithmus

Sei Q Menge der Anfangs- und Endpunkte
und T Menge der jeweils aktiven
Liniensegmente

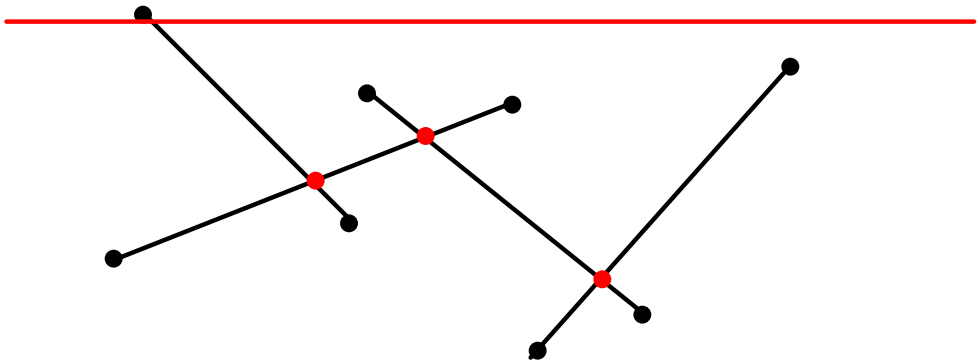


.H	H
.D	D!H
.A	A!DH
.F	AD!F!H
.C	A!C!DFH
.E	ACD!E!FH
D.	AC!EFH
.G	ACEF!G!H
.B	A!B!CEFGH
F.	ABCE!GH
C.	AB!EGH

```
while Q <> {}
    p = Q.Minentfernen
    if (p Startpunkt)  T = T∪{s}
        Gebe Nachbarn (s,s') und (s,s") von s aus
     else
        T = T -{s}
         Gebe Nachbarn (s',s") von s aus
```

Sweep-Line Prinzip



Imaginäre Linie bewegt sich in y Richtung

Jeder Punkt ist ein Ereignis.

Eingabe: Eine Menge (iso-orientierter) Objekte
Ausgabe: {Problemabhängig}

Q: objekt- und problemabh. Folge von Haltepunkten
"Event Queue"

T: angeordnete Menge der jeweils aktiven Objekte
"Status Struktur"

while Q<> {}
wähle nächsten Haltepunkt aus Q und entferne ihn
update (T)
gib problemabhängige Antwort aus

Datenstrukturen

Ereignis-Queue:

Allgemein: Einfügen, Suchen und Löschen

O(log m) Zeit bei m Elementen in Queue.

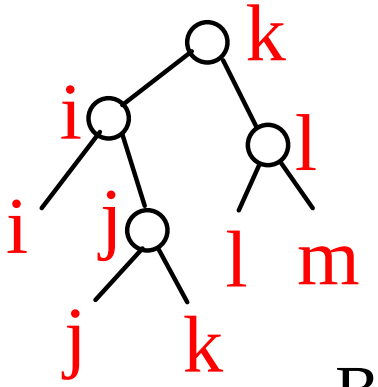
Balancierter Suchbaummit Ordnung

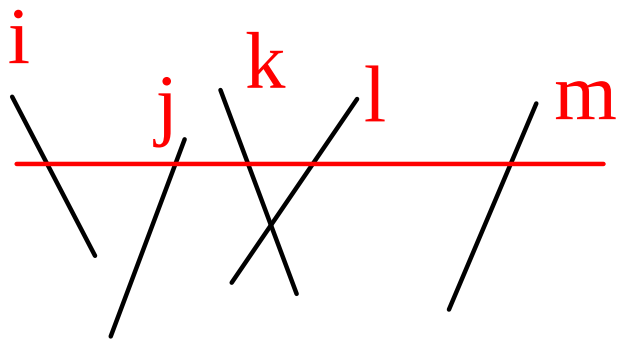
p<q <=> py<qy oder (py=qy und px<qx)

Zumeist reicht: Einfügen und Minentfernen

Vorrangwarteschlange (z.B. Heap, O(log m))

Status Struktur





Balancierter Suchbaum.

Knotenbeschriftung dient zum Routen.

Beispiel: Segmentschnitt

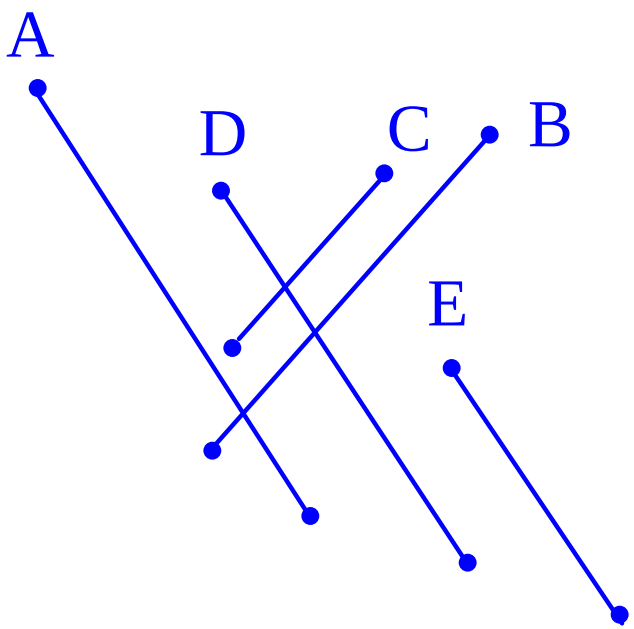
Q

.A
.B
.C
.D

C.
.E

B.
A.
D.
E.

T

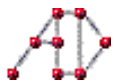


Anzahl Operationen auf Q <= 2n+k, k = # Schnitte

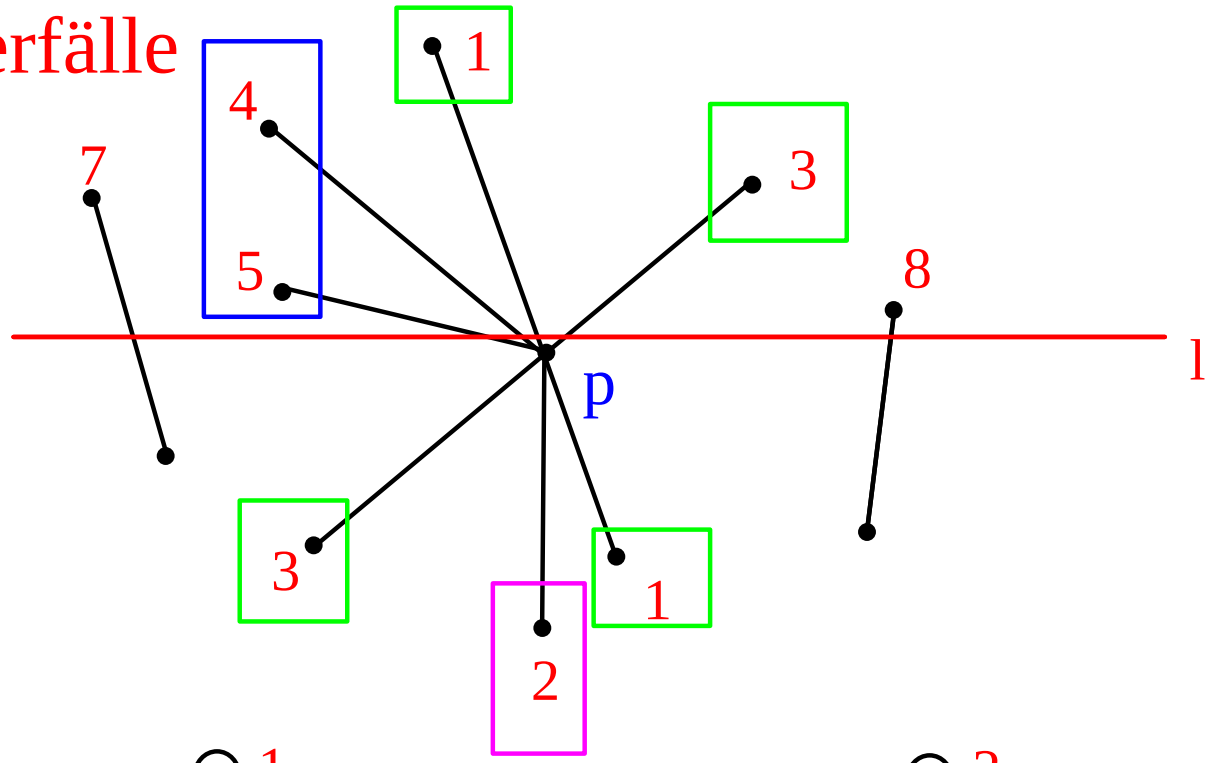
Anzahl Operationen auf T <= 2n+k

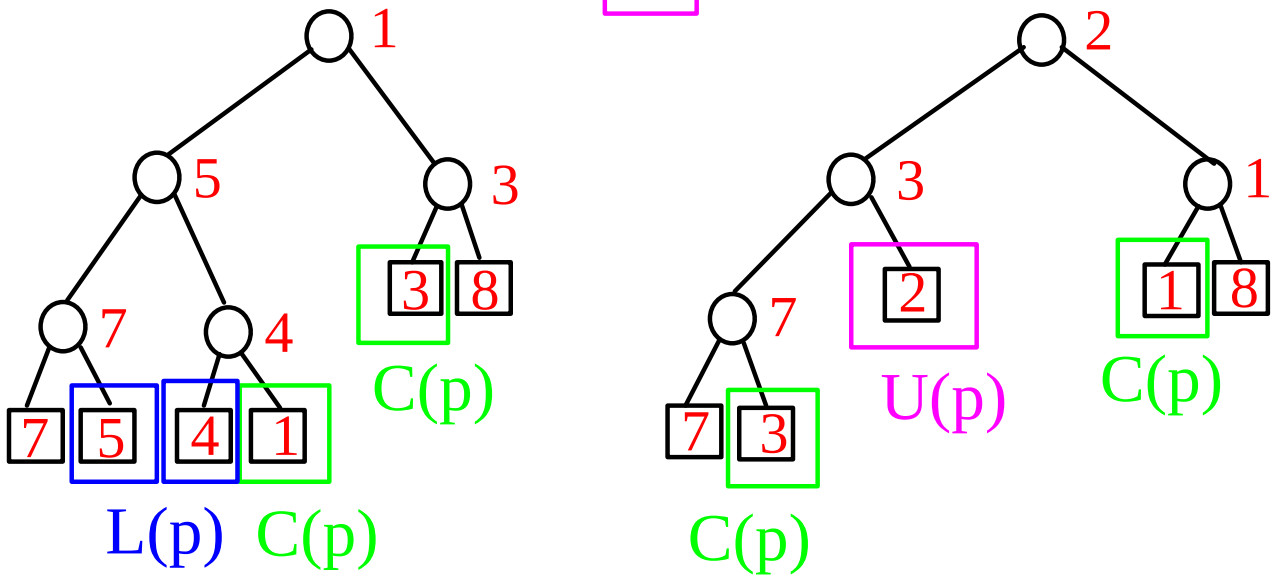
=> Zeitaufwand O((n+k) log n)

Output-Sensitiv



Sonderfälle





Behandlung der Ereignisse

VerarbeiteEvents(p)

if (S(p) enthält mehr als ein Segment)

Gebe p als Schnittpunkt aus

Lösche L(p)$\cup$ C(p) in T, Füge U(p)$\cup$ C(p) in T ein

if (U(p)$\cup$ C(p) = {})

sl und sr linker und rechter Nachbar in T

if (Schnitt(sl,sr) unterhalb Sweep) Q = Q$\cup$ Schnitt

else

s’ linkestes Segment vonU(p) $\cup$ C(p)

sl linker Nachbar in T

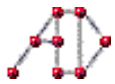
if (Schnitt(sl,s’) unterhalb Sweep) Q = Q$\cup$ Schnitt

s" rechtestes Segment vonU(p) $\cup$ C(p)

sr rechter Nachbar von s" in T

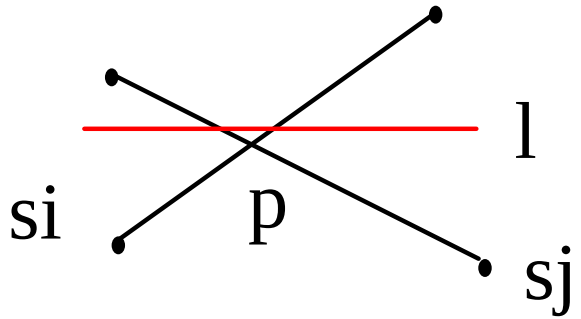
if (Schnitt(s",sr) unterhalb Sweep) Q = Q$\cup$ Schnitt

Segment



Ein einfaches "Nachbarschaftslemma"

Lemma: Seien si und sj zwei nicht-horizontale Segmente mit Schnittpunkt p. Dann gibt es ein Ereignis oberhalb von p, indem si und sj Nachbarn werden.



Beweis:

Sei l so nah an p, daß
si und sj nebeneinander liegen.
Da si und sj zu Anfang des Algorithmus nicht benachbart sind, gibt es ein Ereignis q, an dem si und sj Nachbarn werden (und auf Durchschnitt geprüft werden).

Korrektheit

Invariante: Schnittpunkte oberhalb der Sweep-Line korrekt

Induktion über die Anzahl von Elementen in Q

Fall 1: Schnittpunkt p ist Endpunkt eines Segmentes

in Q und U(p) => Segment mit p gespeichert (initial)

in Q und $L(p) \cup C(p)$ => Segment in T enthalten (derzeit)

=> Segmente und p gefunden und korrekt berechnet

Fall 2: p nicht Endpunkt. z.Z. p wird in Q eingefügt

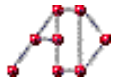
Seien si und sj zwei benachbarte Segmente

Lemma => $\exists$ Event oberhalb von p, so daß

si und sj Nachbarn werden

si und sj werden auf Durchschnitt geprüft und p eingefügt

=> p wurde in Q gefunden und gespeichert



Laufzeit

Einfügen der Endpunkte: O(n log n)

m(p) = |L(p)|+|U(p)|+|C(p)| und m = m(p1) + ... + m(pn)

=> Laufzeit O(m log n)

Sicher m = O(n+k) mit k = Outputlänge
(wenn m(p) > 1 werden Segmente ausgegeben)

Ziel: m = O(n+I) mit I = #Schnittpunkte

m(p) <= Verzweigungsgrad des Knotens p

=> m <= 2 |E| mit E = #Kanten

|V| = 2n+I und in planaren Graphen |E| = O(|V|)

z.Z. |E| = O(|V|)

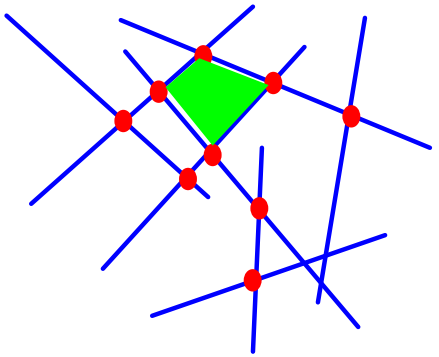
Euler: |V| - |E| + |F| >= 2

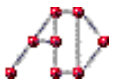
F=Flächen. Es gilt |F| <= 2|E|/3

=> 2 <=(2n+I)-|E|+2|E|/3

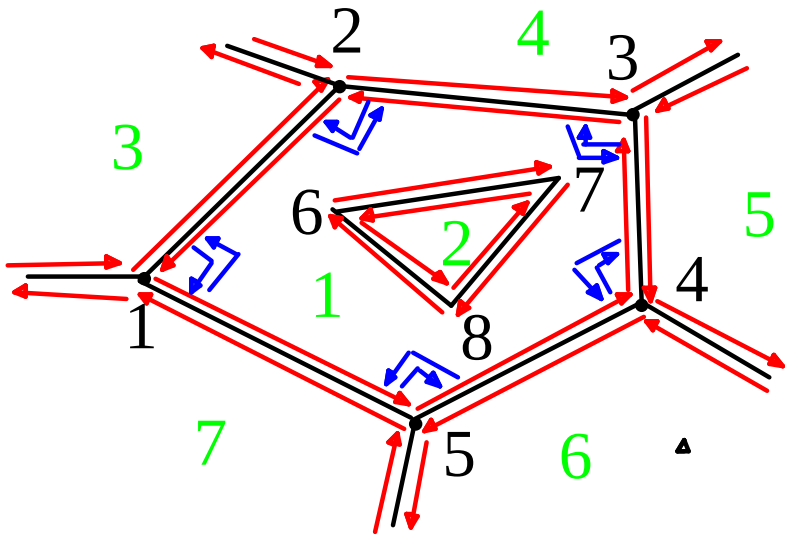
Damit |E| <= 6n+3I-6

und m <= 12n +6I-12





Doppelt verkettete Kantenliste



z.B.

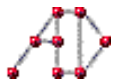
Knoten 1 = {(1,2) | 12}

Fläche 1 = { 15 | [67] }

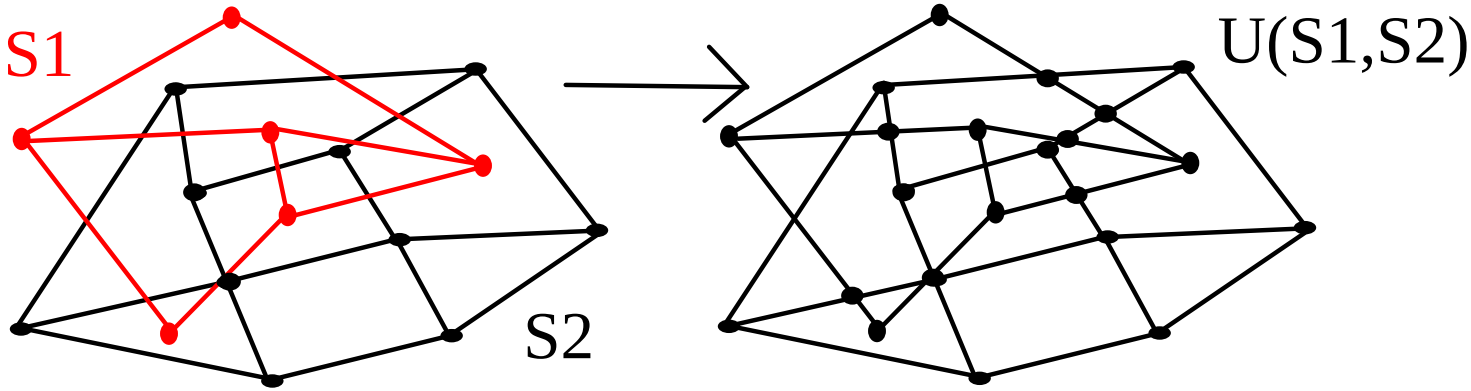
Kante 54 = { 4 | 45 | 1 | 43 | 15 }

3 Records:

```
vertex {
    Coordinates
    Incident Edge
};
face {
    OuterComponent
    InnerComponents
};
halfedge {
    Origin
    Twin
    IncidentFace
    Next
    Prev
};
```

Überlagerung

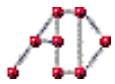


Plane Sweep (oben nach unten)
Datenstrukturen: Status Struktur T, EventQueue Q
Doppelt verkettete Kantenliste D

Kanten in Q und D sind kreuzweise miteinander verbunden.

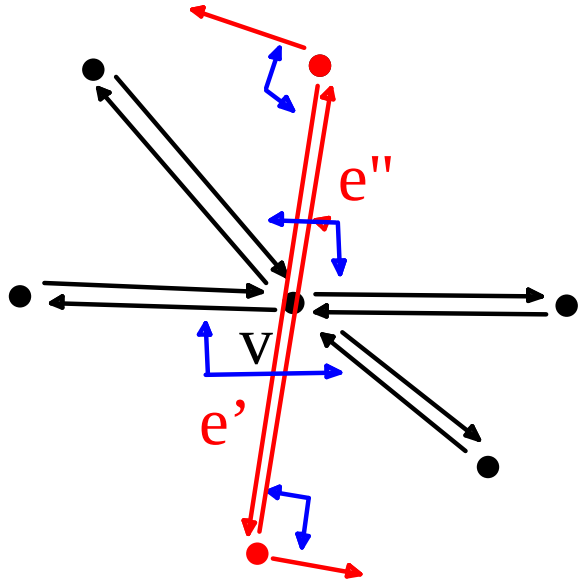
Wiederverwendung von Kanten, da Orientierung erhalten bleibt
Aktualisierung von T und Q wie Segmentschnitt

2 Phasen: Kanten und Ecken
Flächen



Kanten und Ecken

Beispiel: Kante einer Komponente schneidet Knoten der anderen



zwei halfedge Records e', e" mit v als origingenerieren

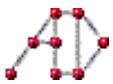
setze twin-Pointer und verdoppele Kanten

next und prev an den Eckpunkten setzen und Nachbarn von e aktualisieren

next und prev an v setzen, zyklische Ordnung

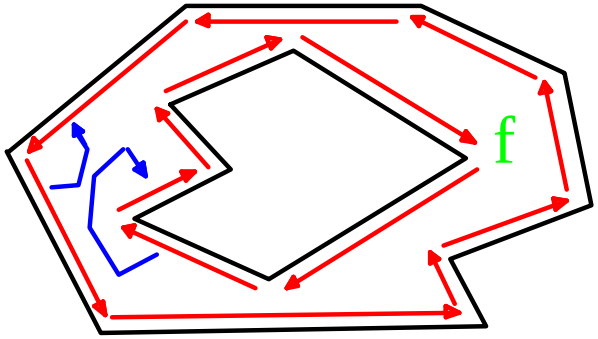
=> Zeit O(1+deg(v)) an einem Knoten

=> Zeit O(n log n+k log n) insgesamt, k Kompl. U(S1,S2)



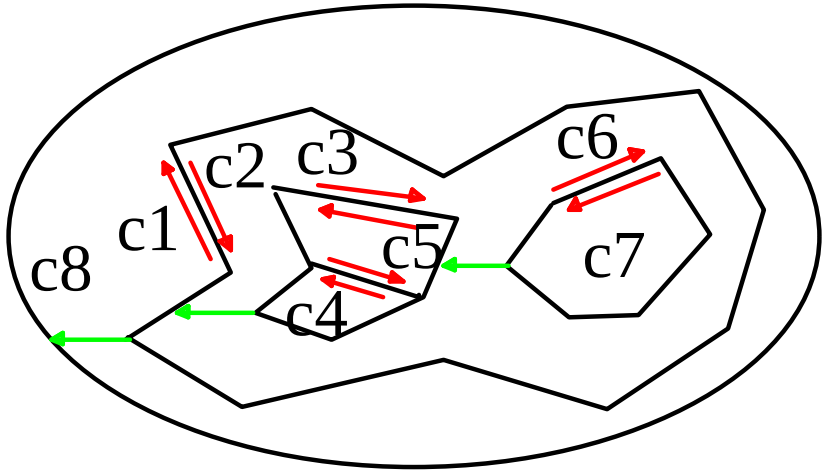
Flächen

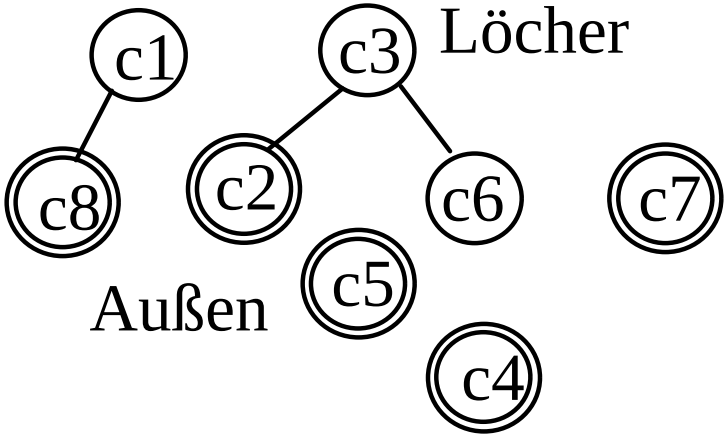
Unterscheidung innen und außen durch 180^o Argument



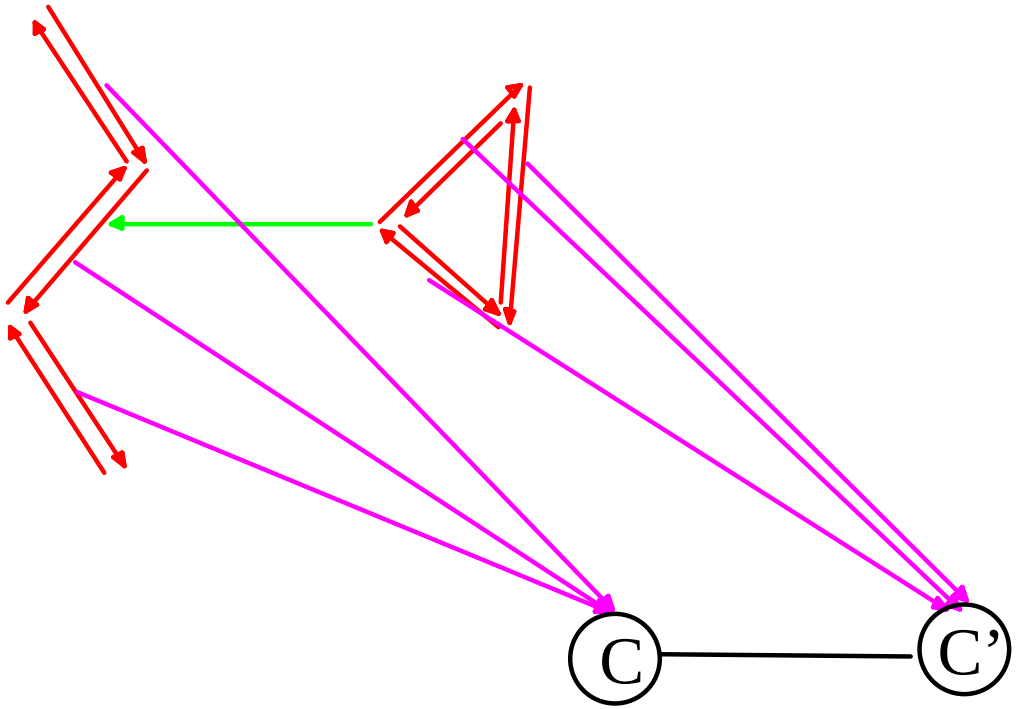
Gilt nur für linkesten Knoten!

Unterscheidung gleiche Fläche



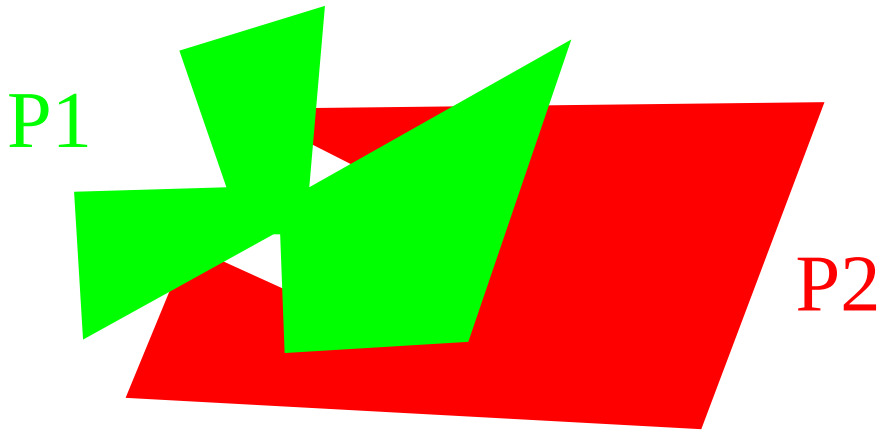


Konstruktion von G



Satz Verbundene Komponenten bilden eine Fläche
G kann in O(n+k) konstruiert werden

Boole'sche Operationen für Polygone



$P1 \cap P2$ Neue Flächen in Überlappung

$P1 \cup P2$ Alle Flächen in Überlappung

$P1 - P2$ Alte Flächen - neu generierte Flächen

gpoly

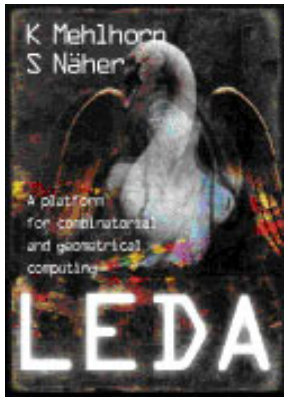
=>Sei n = |P1| + |P2|

Alle 3 Operationen können in O(n log n+k log n) berechnet werden. k ist die Outputgröße

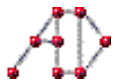
LEDA library of efficient data structures and algorithms

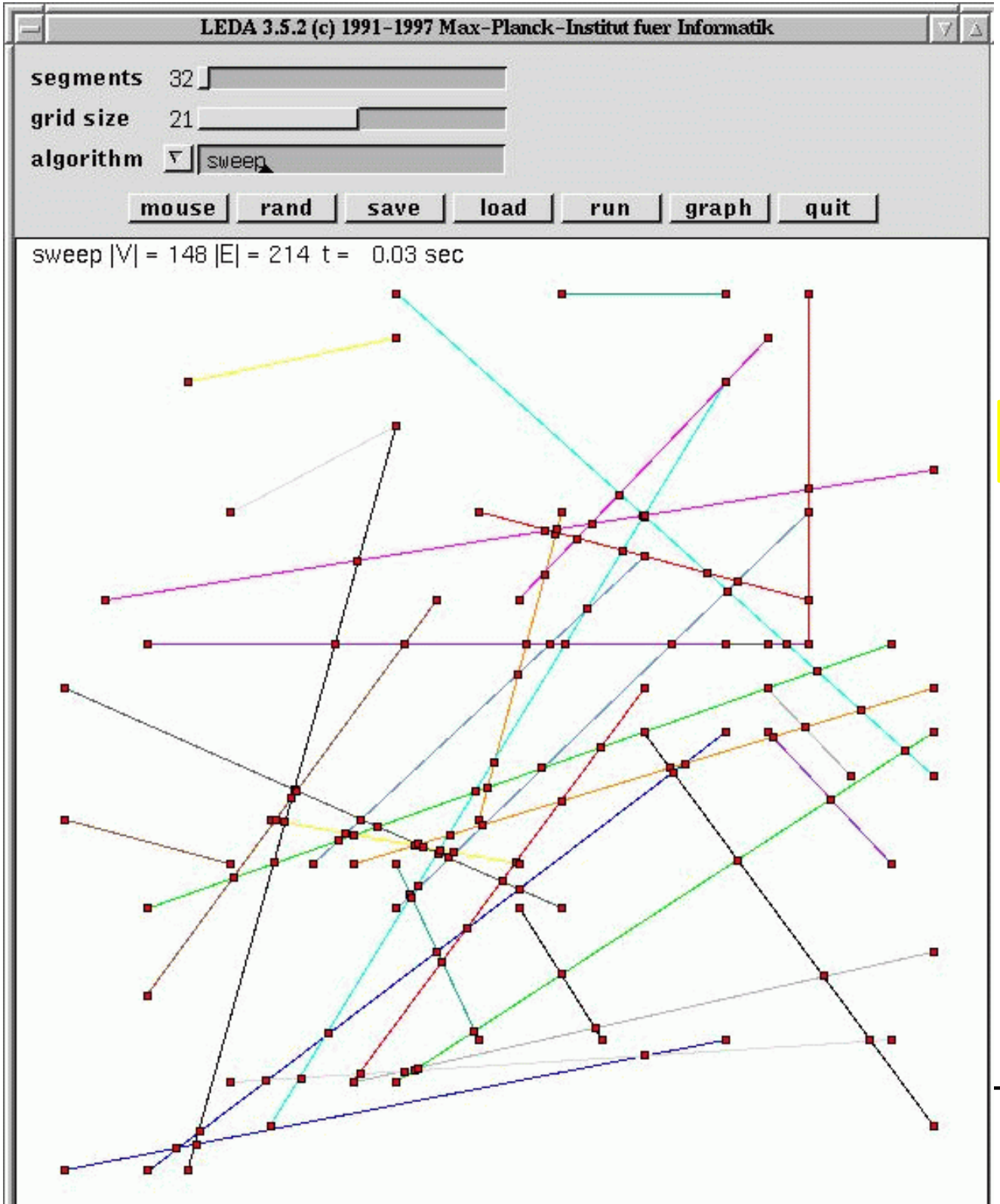
http://www.mpi-sb.mpg.de/~LEDA/leda.html

http://www.mpi-sb.mpg.de/~mehlhorn/LEDAbook.html



Installiert unter
/usr/local/leda/v3.6.1

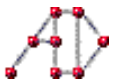


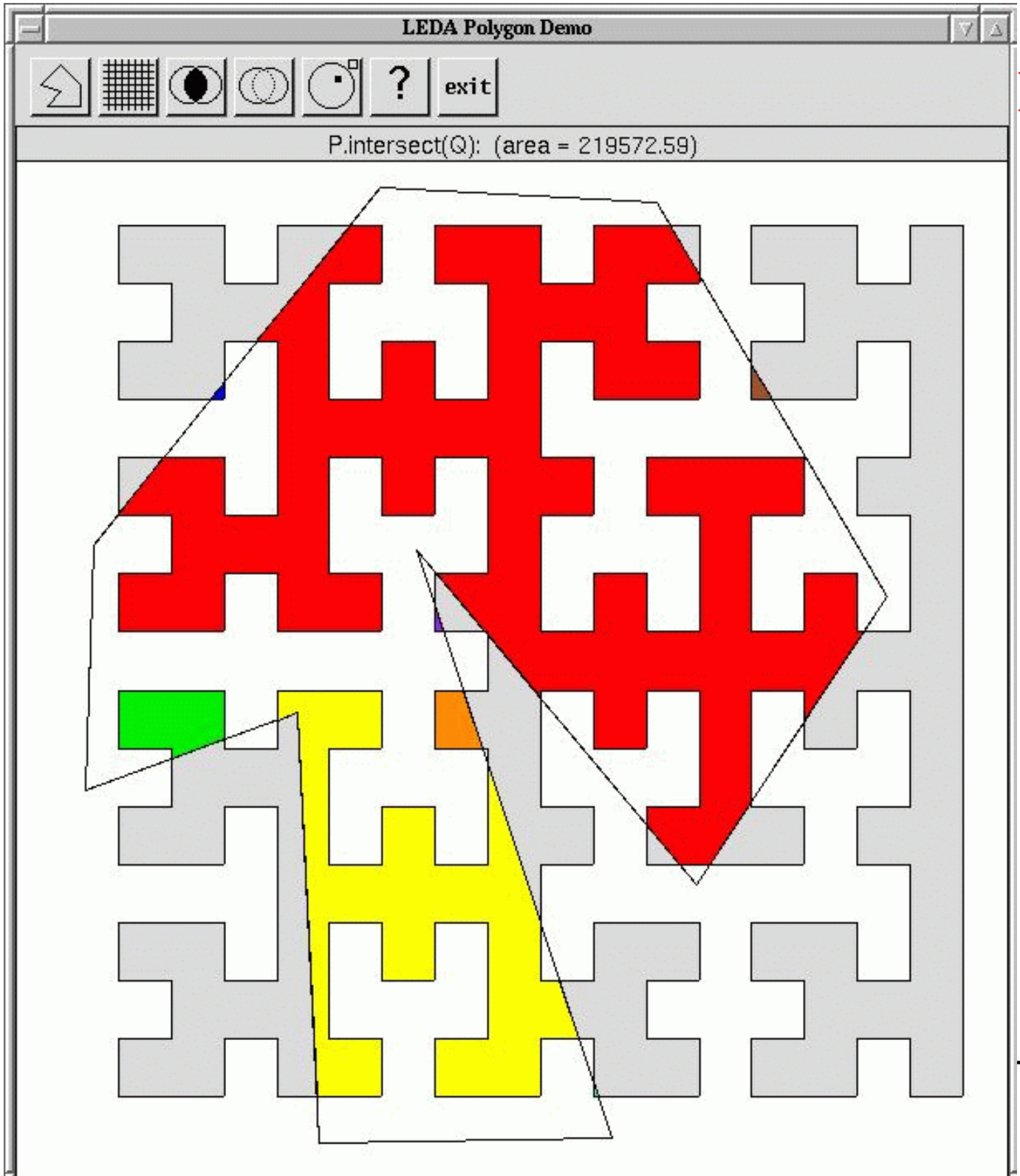


Beispiel

Segment-schnitt

segment2





Beispiel

Polygon-schnitt

Beispiel

3D konvexe Hülle