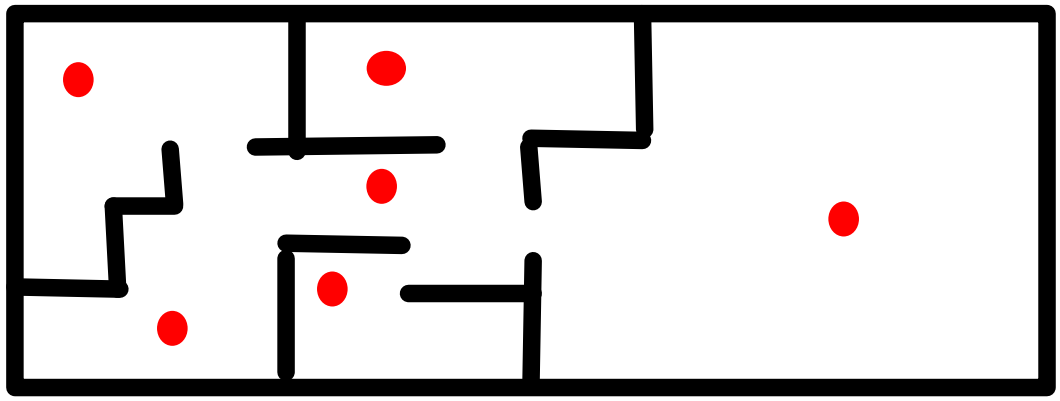
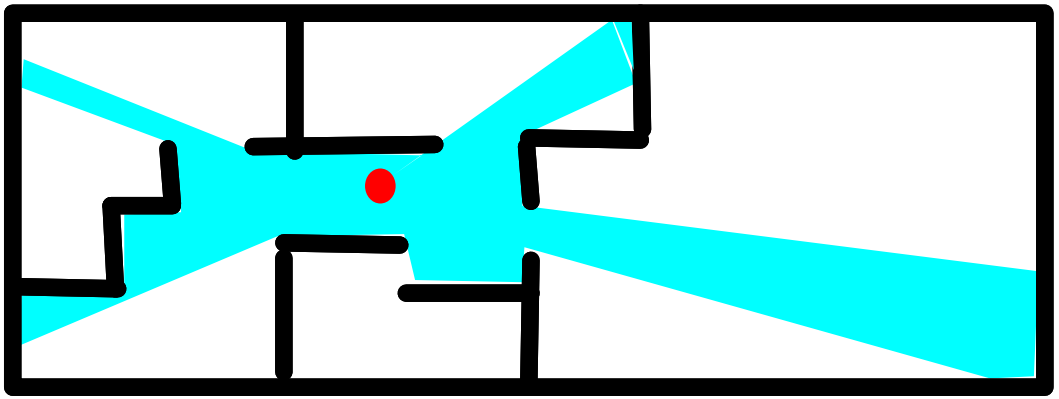
Polygon Triangulation

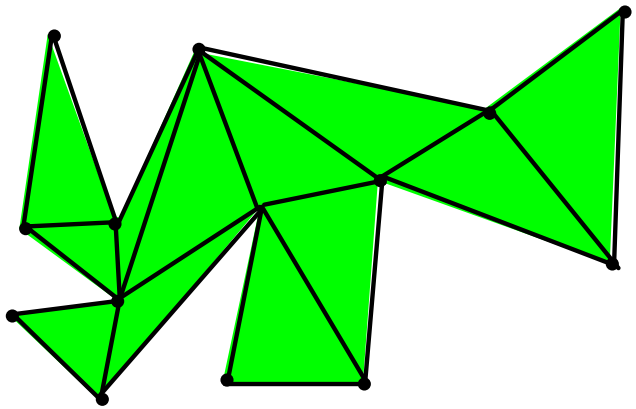
"Art Gallery Problem"



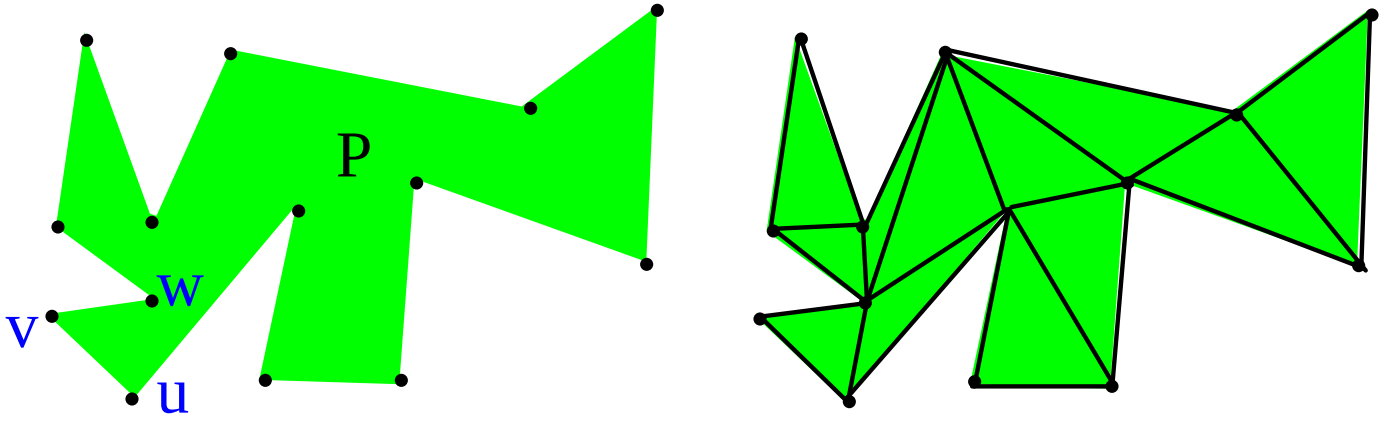
Sichtbarkeitspolygon

robot.scr





Triangulation simpler Polygone



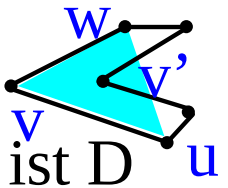
Satz: Triangulation existiert und besitzt n-2 Dreiecke

Bew.: Induktion über n> 3. Suche Diagonale D in P.

Sei v linkester Knoten von Pund u,w Nachbarn von v

Falls $\overline{uw}$ in P => D gefunden

sonst existieren Knoten innerhalb von $\overline{uvw}$

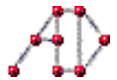


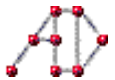
Sei v' am weitesten von $\overline{uw}$ entfernt => $\overline{vv'}$ ist D

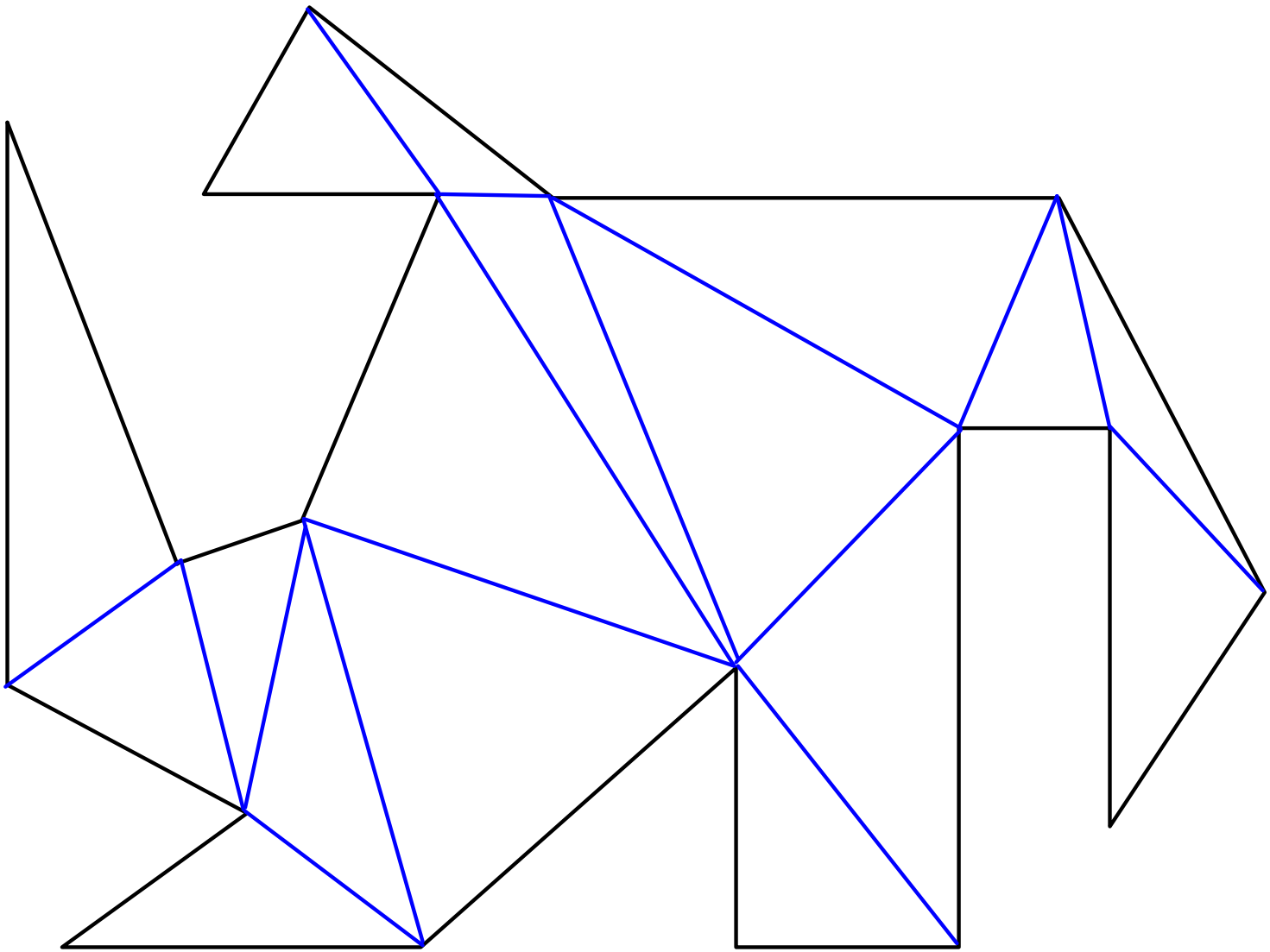
=> Teilpolygone P1 und P2 mit Triangulation von
m1-2 bzw. m2-2 Dreiecken. Es gilt m1+m2 = n+2

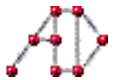
=> (m1-2)+(m2-2) = n-2 Dreiecke in P

□





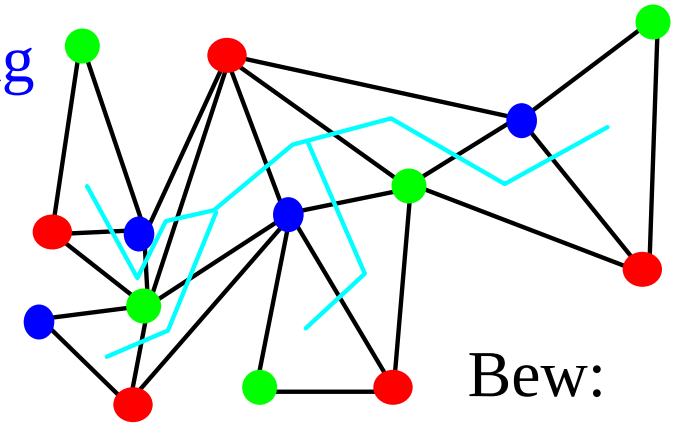




Obere und Untere Schranke

Wissen: (n-2) Kameras reichen. Wie geht's besser?

Nehme 3-Färbung
und plaziere
Kameras auf
eine Farbe



Satz:

3-Färbung existiert immer.

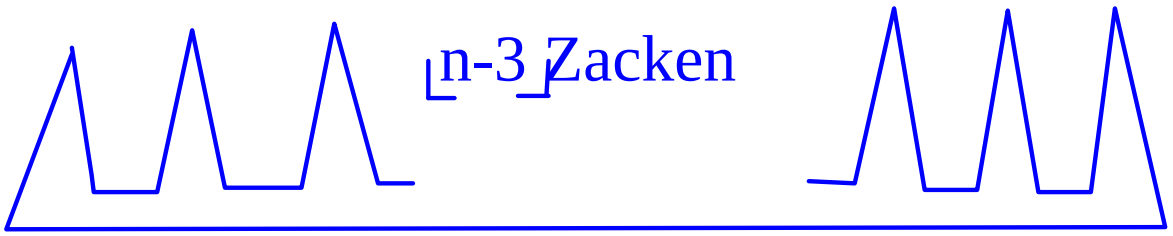
=> $\lfloor n/3 \rfloor$ Kameras reichen aus

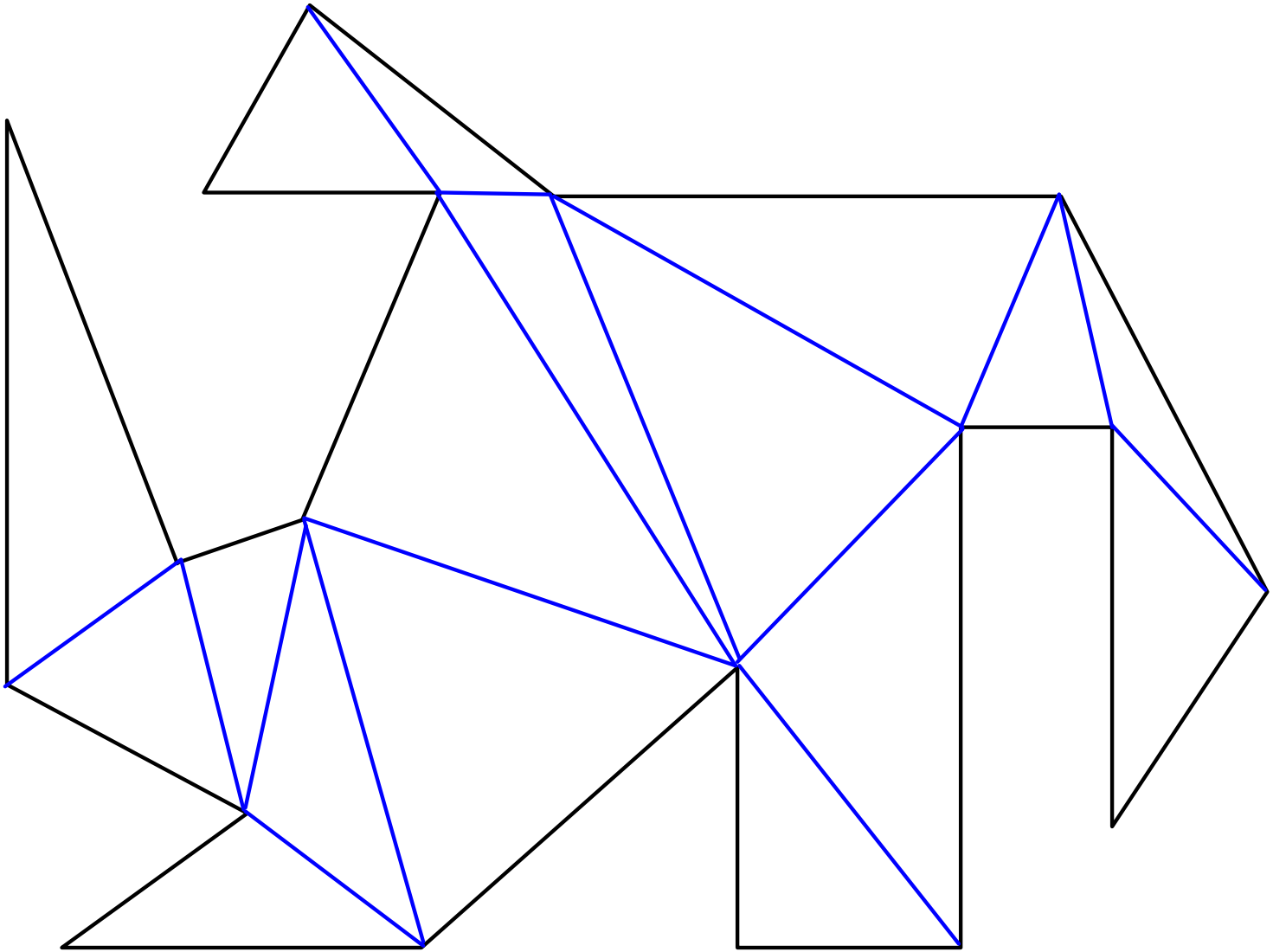
Bew:

Dualer Graph
ist ein Baum, der
in DFS konstruiert
werden kann

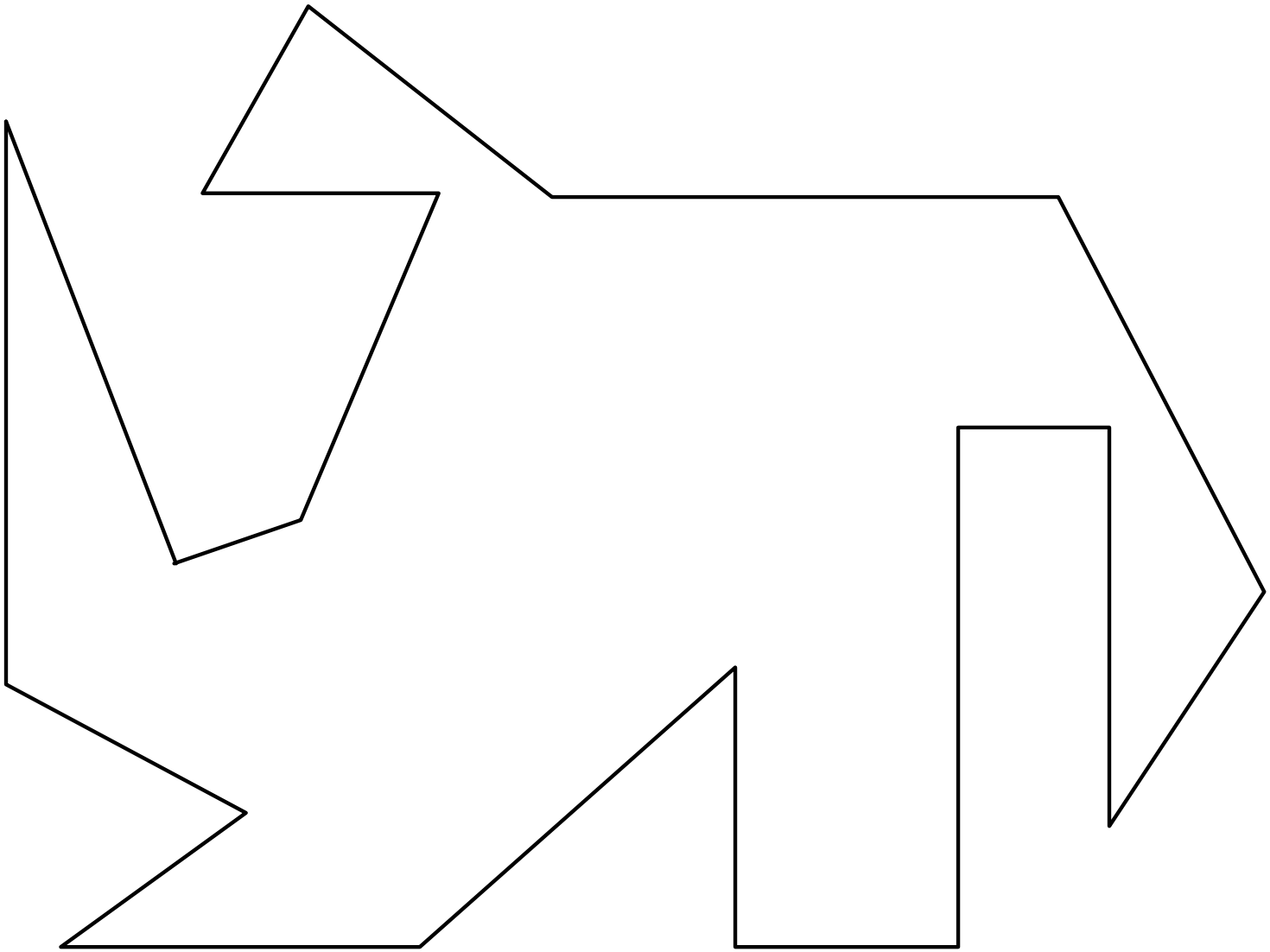
Satz: n/3 Kameras notwendig!

Bew.: worst-case Beispiel

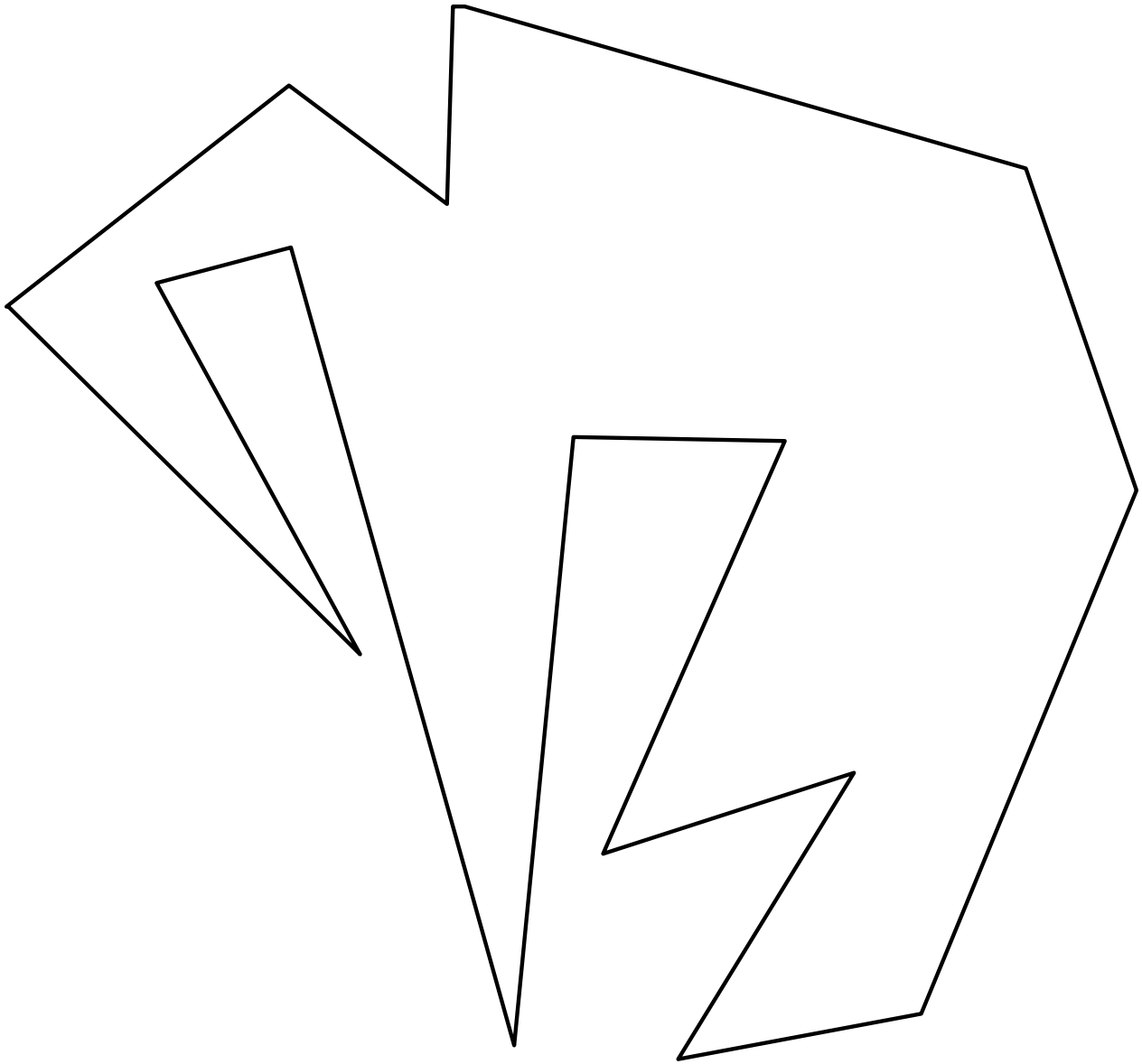




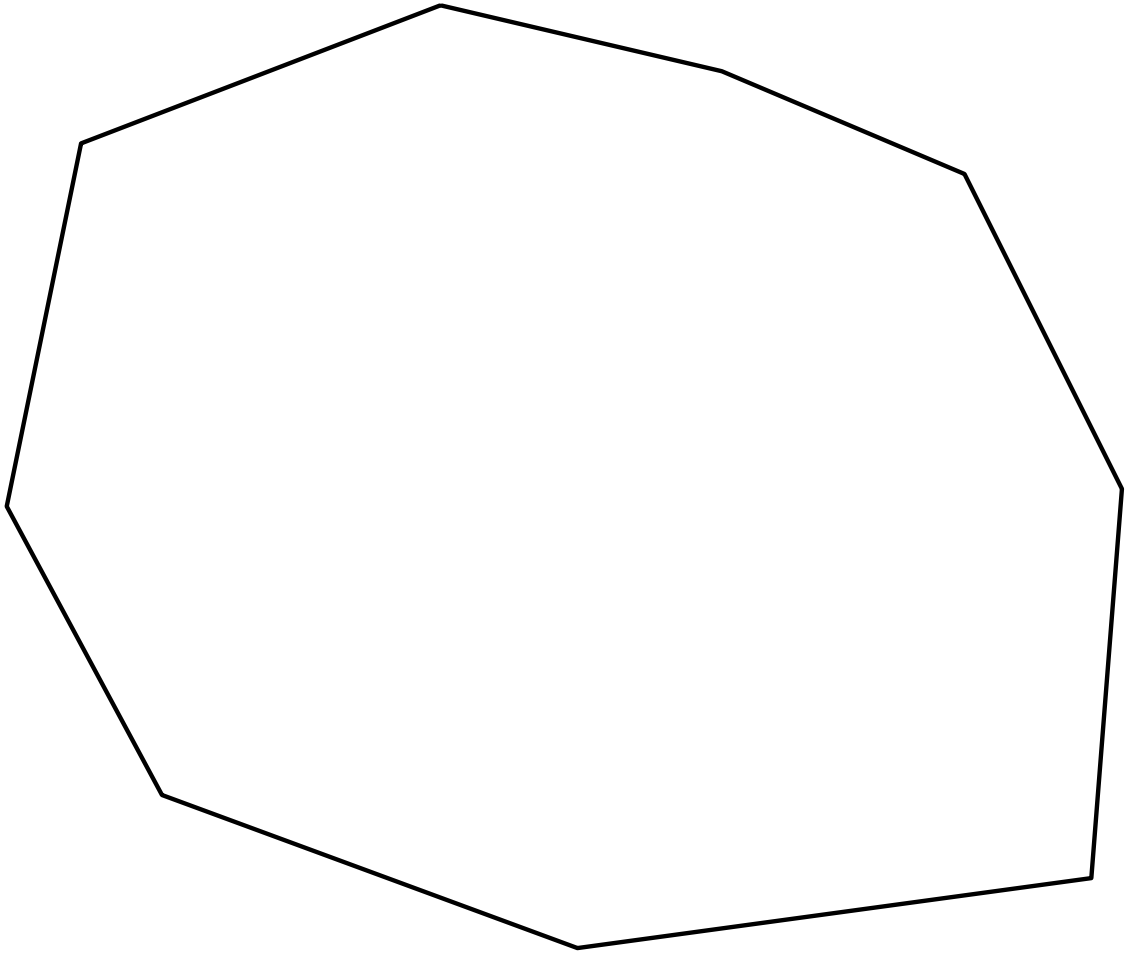
Triangulation (naiv)



Triangulation (naiv)

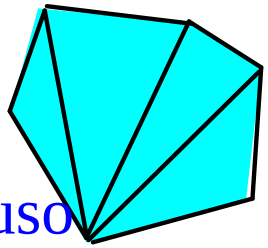


Triangulation konvexer Polygone

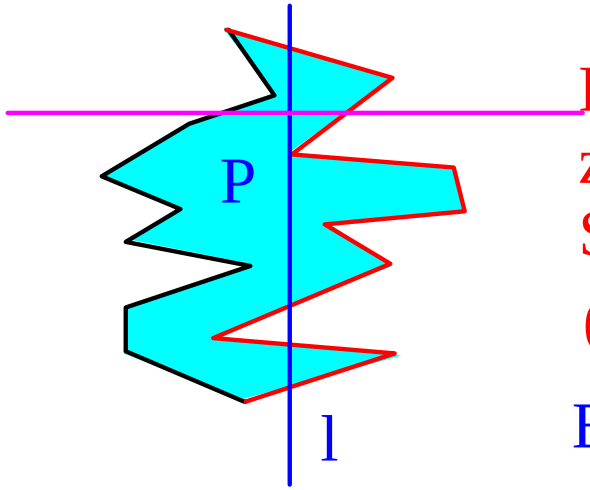


l-Monotonie

Konvexe Polygone sind leicht zu triangulieren



Leider: Aufteilung in konvexe Einheiten genauso schwer wie Triangulation.



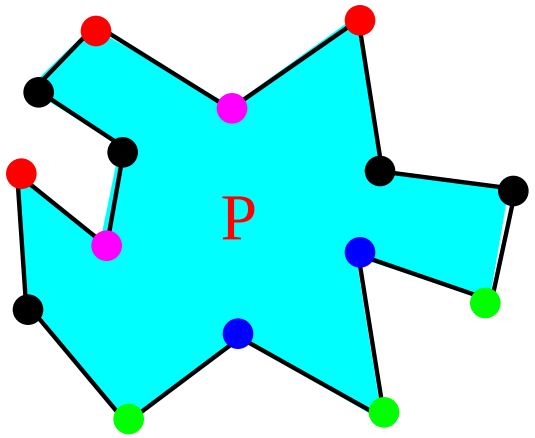
Ein Polygon P ist l-monoton zur Geraden l, falls alle orthogonalen Schnitte von P mit l verbunden sind.

(y-monoton, wenn l = y-Achse)

Beobachtung: P y-monoton => linke und rechte Begrenzung von top-Knoten immer abwärts

2 Schritte zur Triangulation:
- Unterteile P in monotone Teile P1,...,Pk
- Trianguliere P1,...,Pk

Split- und Mergeknoten



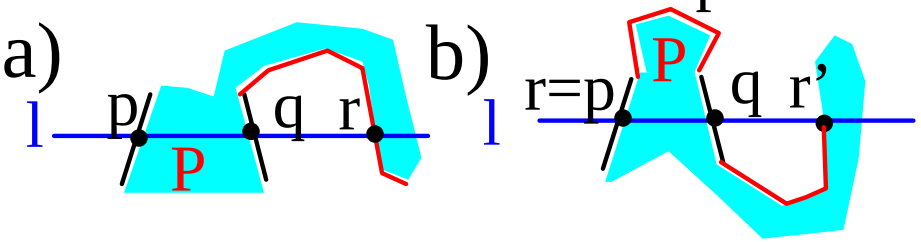
- start: Nachbarn unten, < 180
- end: Nachbarn oben, < 180
- merge: Nachbarn oben, > 180
- split: Nachbarn unten, > 180
- normal: sonst

Satz: P y-monoton, falls es keine split und merge Knoten hat

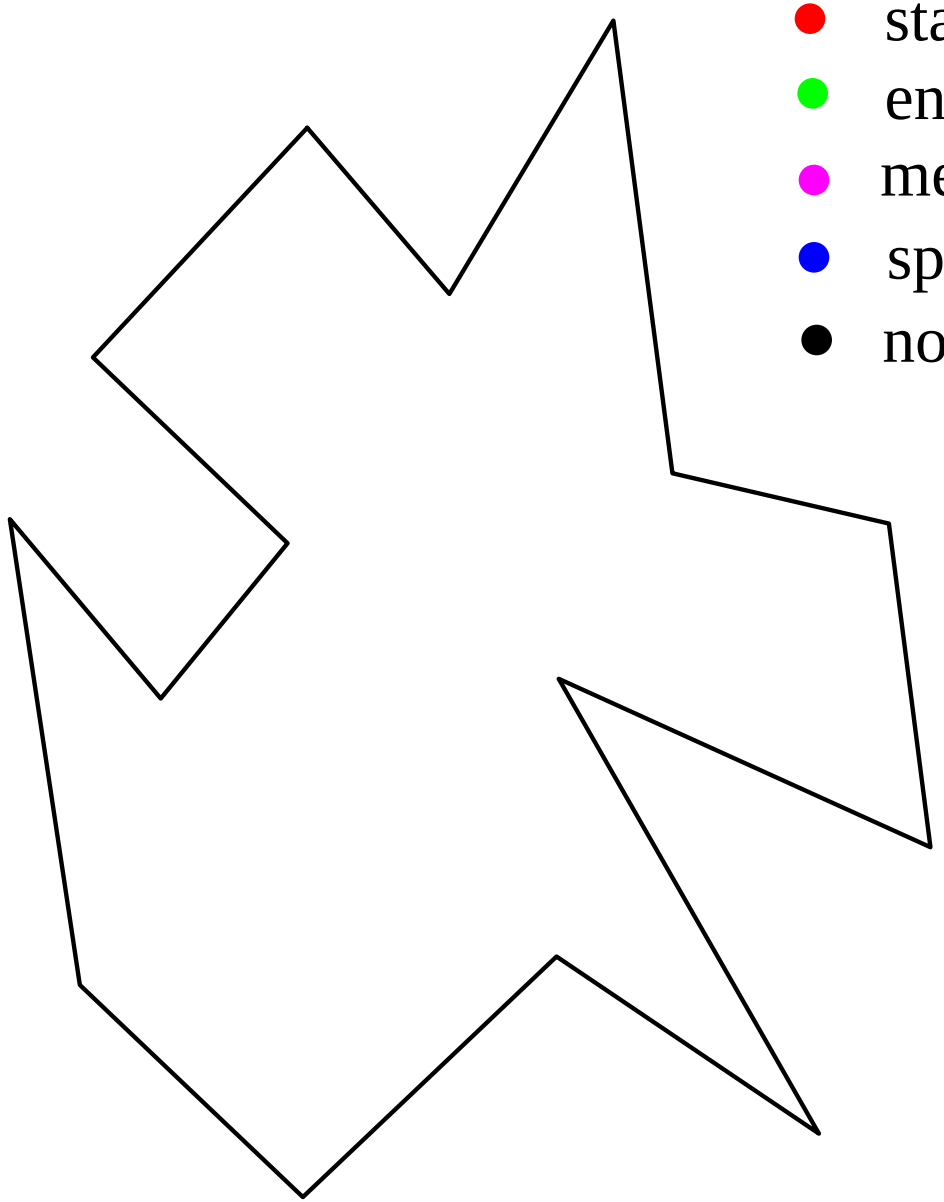
Bew: Ann: P nicht y-monoton => ex. l mit > 1 Komponente

2 Fälle mit Linie links:

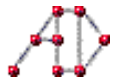
Starte bei q bis r erreicht,
(gehe nach oben)



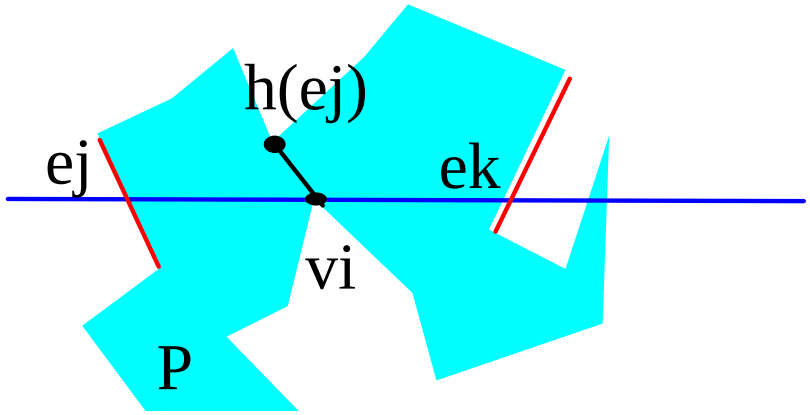
a) r <> p => es existiert ein split Knoten zw. q und r
b) r = p => es existiert ein r' (gehe nach unten) und damit ein merge Knoten □



- start: Nachbarn unten, < 180
- end: Nachbarn oben, < 180
- merge: Nachbarn oben, > 180
- split: Nachbarn unten, > 180
- normal: sonst



Entfernen von split und merge Knoten



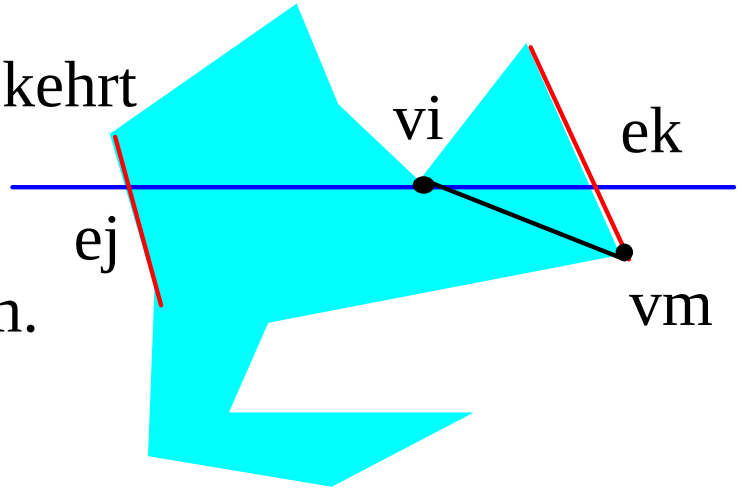
h(ej) ist niedrigster Knoten oberhalb der sweep-line, so daß Horiz. $\overline{ej\ h(ej)}$ ganz in P liegt

=> Entfernung von splits

merge Knoten sind splits umgekehrt

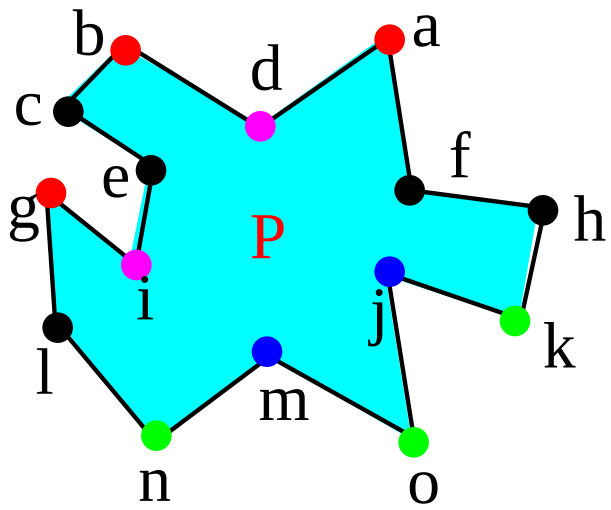
Wenn im sweep also der Helfer h(ej) wechselt, finden wir den assoziierten Knoten vm.

Falls Helfer nicht wechselt, nehme Endpunkt von ej



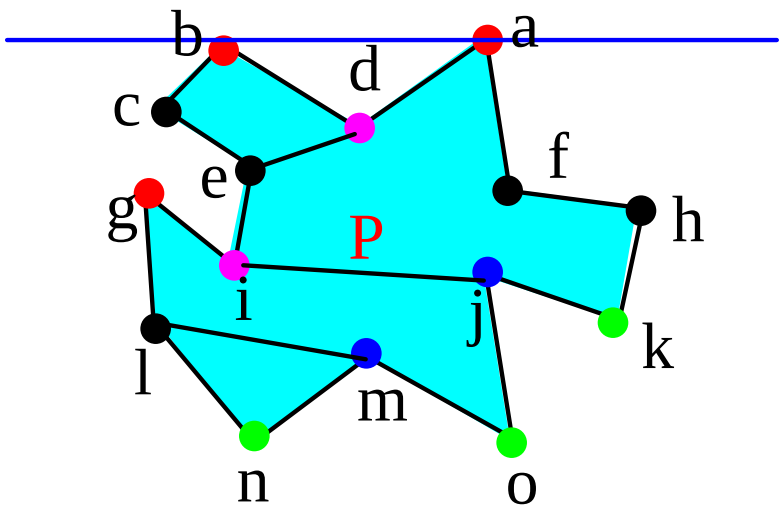
=> Entfernung aller merges

Ein Beispiel



v		T	helper
a		ad	ad = a
b		ad,bc	bc = b
c		ad,ce	ce = c
d		ce	ce = d
e	!	ei	ei = e
f		ei	
g		ei,gl	gl = g
h		ei,gl	
i		gl	gl = i
j	!	gl,jo	jo = gl = j
k		gl	
l		ln	ln = l
m	!	ln,mo	ln= mo= m
n		mo	
o			

Ein Beispiel



v		T	helper
a		ad	ad = a
b		ad,bc	bc = b
c		ad,ce	ce = c
d		ce	ce = d
e	!	ei	ei = e
f		ei	
g		ei,gl	gl = g
h		ei,gl	
i		gl	gl = i
j	!	gl,jo	jo = gl = j
k		gl	
l		ln	ln = l
m	!	ln,mo	ln= mo= m
n		mo	
o			

Prozedur MakeMonoton

Input: Polygon P, doppelt verkettete Kantenliste D

Output: Aufteilung in monotone Teilpolygone

Konstruiere EventQueue Q und initialisiere Status Struktur T

Solange Q <> {}

Entferne vi mit höchster Priorität

HandleStart(vi): $T = T \cup \{ei\}$, $h(ei) = vi$

HandleEnd(vi): if ($h(e_{i-1})$ merge) $D = D \cup diag(vi, h(e_{i-1}))$

$T = T - \{ e_{i-1} \}$

HandleSplit(vi): $ej = T.search()$, $D = D \cup diag(vi, h(ej))$

$h(ej) = h(ei) = vi$, $T = T \cup \{ei\}$

HandleMerge(vi): HandleEnd(vi) $ej = T.search()$, $h(ej) = vi$

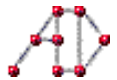
if (h(ej) merge) $D = D \cup diag(vi, ej)$

HandleRegular(vi): if (vi links in P) HandleEnd(vi)

$h(ei) = vi$, $T = T \cup \{ei\}$

else $ej = T.search()$ $h(ej) = vi$

if (h(ej) merge) $D = D \cup diag(vi, ej)$



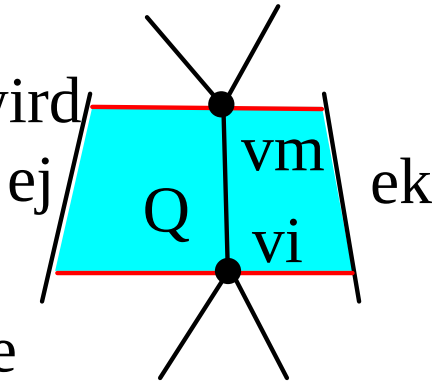
Korrektheit (HandleSplit)

Betrachte $\overline{vm\ vi}$, wenn vi besucht wird

=> h(ej) = vm

z.Z. 1) $\overline{vm\ vi}$ schneidet weder P

2) noch eine andere Diagonale

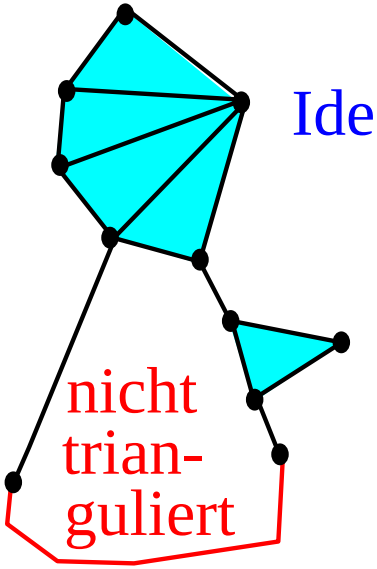


1) Es gibt keine Knoten von P in Q, sonst h(ej) <> vm
Ann. ex. Kante von P, die $\overline{vm\ vi}$ schneidet
=> Schnitt $\overline{vi\ ej}$ oder $\overline{vi\ ek}$
Widerspruch: ej links von vi und vm

2) Da keine Knoten von P in Q und beide Endpunkte oberhalb von vi => keinen Schnit einer anderen Diagonale mit $\overline{vm\ vi}$

Analoge Argumentation in den anderen Ereignisbehandlungen

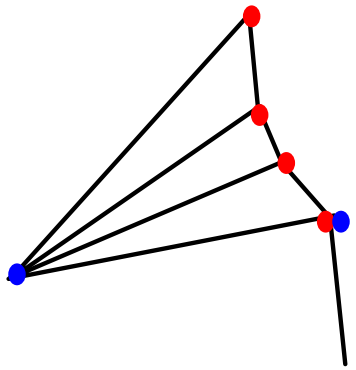
Triangulation y-monotoner Polygone



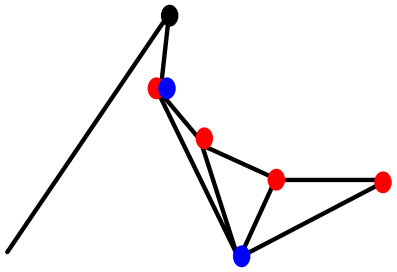
Idee: Fächer solange bauen bis Konvexheit verletzt
Alternierung von linker + rechter Seite

Implementierung: Scan-line
Nutze Stack als Datenstruktur

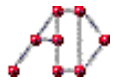
Fall 1: Seitenwechsel



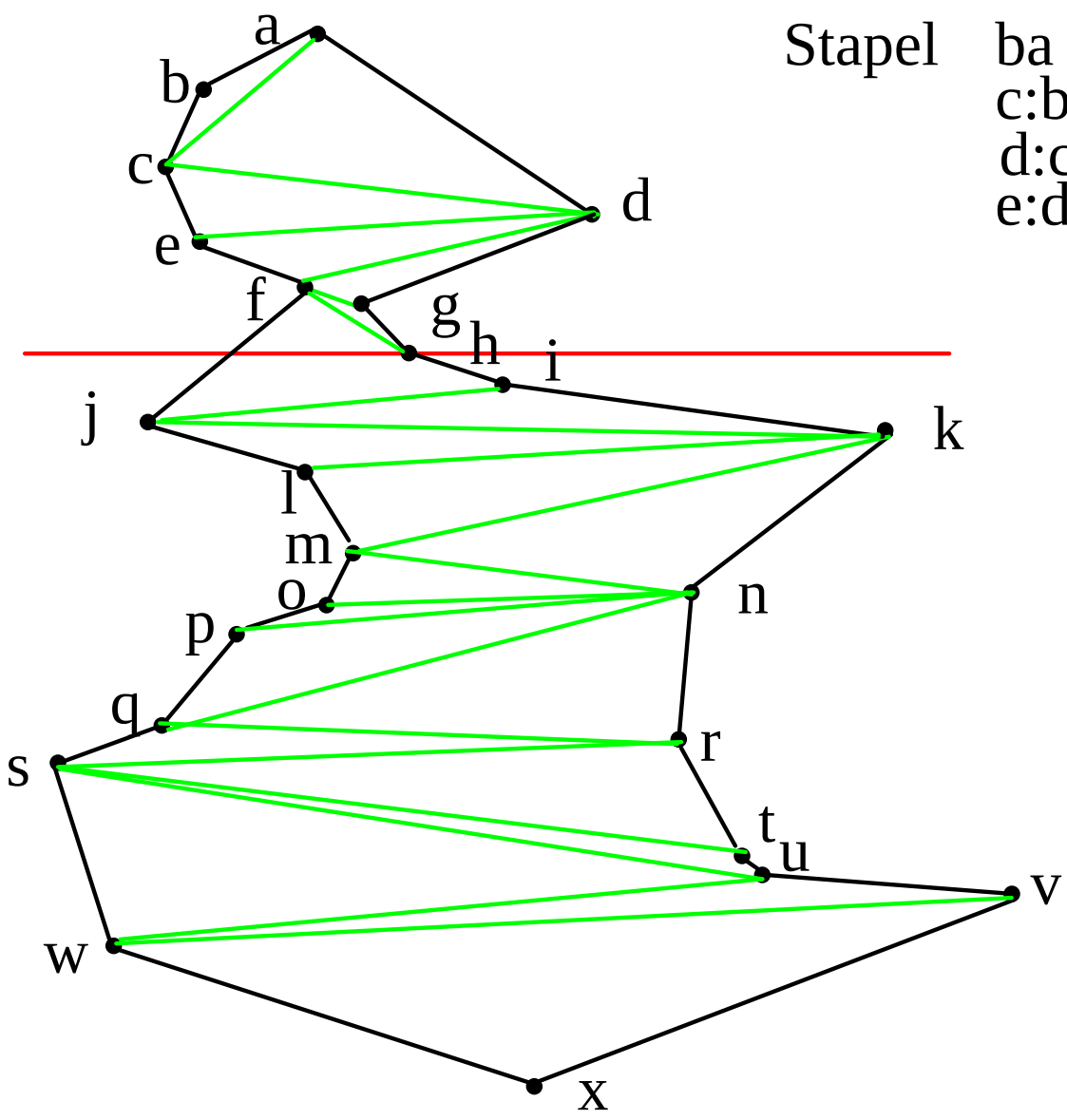
Fall 2: Gleiche Seite



popped
pushed



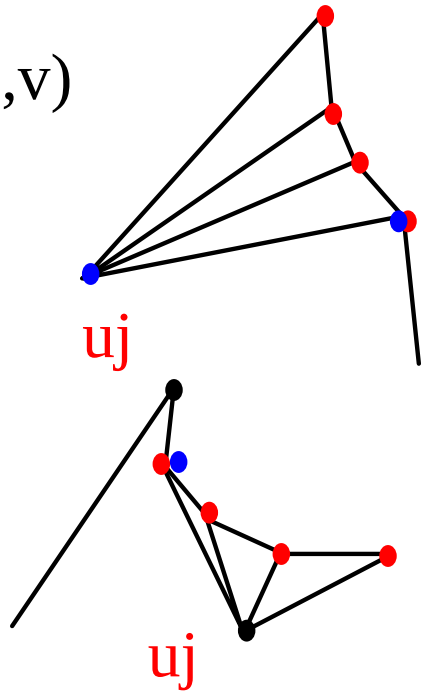
Beispiel



Stapel ba
c:ba->ca
d:ca->dc
e:dc->ed

Implementation

```
S.push(u1), S.push(u2)
for j = 1,..,n-1
        if (side(uj)<>side(S.top))
          while (S<>{}) v=S.pop, diag(uj,v)
          S.push(uj-1)
          S.push(uj)
        else
           while (diag(S.top,uj) in P)
              diag(S.top,uj)
              S.pop
           S.push(last)
           S.push(uj)
```

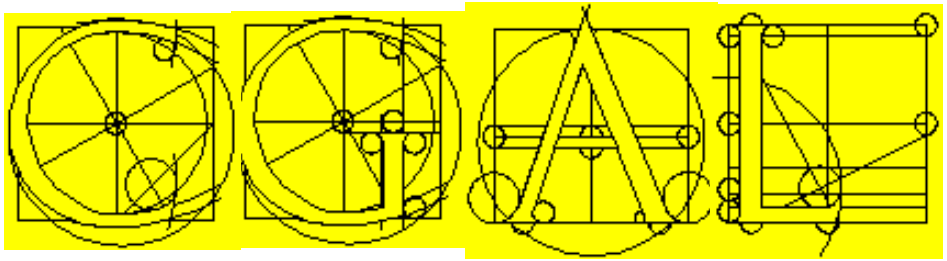


Satz: Zeit O(n).

Bew: #pops < #pushes

CGAL

computational geometry algorithms library



Kernel

http://www.cs.uu.nl/CGAL

2D/3D point, vector, direction, segment, ray, line, dD point
triangle, bounding box, iso-rectangle, circle, plane, tetrahedron
predicates, affine transformations, intersection and distance calculation

Basic Library

halfedge data structure, topological map, planar map, polyhedron
Boolean operations on polygons, planar map overlay, triangulation,
Delaunay triangulation 2D/3D convex hull, and 2D extreme points
smallest enclosing circle/sphere and ellipse, maximum inscribed k-gon,
and other optimisations range tree, segment tree, kD tree