Triangulation eines Polygons in 2D

Sven Eckelmann

TU Chemnitz

27. Mai 2005

TECHNISCHE UNIVERSITÄT
CHEMNITZ

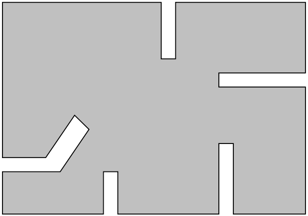
Inhaltsverzeichnis

Motivation

- Einfache Darstellung komplexer Geometrie
- Effiziente Verarbeitung durch Dreiecke
- Beispiele:
  - Zeichnen von Polygonen
  - Art-Gallery-Problem

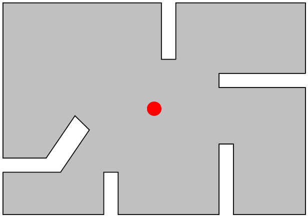
Motivation

- Einfache Darstellung komplexer Geometrie
- Effiziente Verarbeitung durch Dreiecke
- Beispiele:
  - Zeichnen von Polygonen
  - Art-Gallery-Problem



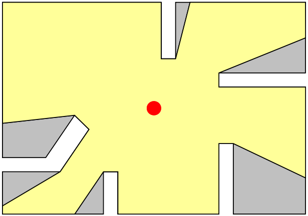
Motivation

- Einfache Darstellung komplexer Geometrie
- Effiziente Verarbeitung durch Dreiecke
- Beispiele:
  - Zeichnen von Polygonen
  - Art-Gallery-Problem



Motivation

- Einfache Darstellung komplexer Geometrie
- Effiziente Verarbeitung durch Dreiecke
- Beispiele:
  - Zeichnen von Polygonen
  - Art-Gallery-Problem



Definitionen

- Polygon: $P = (p_1, p_2, \ldots, p_n), p_i \in \mathbb{R}^m, 1 \leq i \leq n$
- Kante eines Polygons: $\overline{p_i p_{i+1}}(i = 1, \ldots, n-1)$ und $\overline{p_n p_1}$
- Diagonalen eines Polygons:
  - keine Kante des Polygons und
  - schneidet keine Kante und innerhalb des Polygons
- Triangulation eines Polygons:
  - Zerlegung eines einfachen Polygons P in Dreiecke ohne das Hinzufügen neuer Punkte

Wie viele Dreiecke braucht man?

- Gegeben: $n =$ Anzahl der Ecken
- Gesucht: $F =$ Anzahl der Dreiecke
- Kantenanzahl: $K = \frac{3*F+n}{2} + \frac{3*F+n}{2} - n = 3*F$
- Eulersche Polyedersatz: $n + F - 3*F = 2$
  - nur eine Seite des Polyeders interessant!
  - $n + 2*F - 3*F = 2 \Rightarrow n - F = 2 \Rightarrow F = n - 2$

Wie viele Dreiecke braucht man?

- Gegeben: $n =$ Anzahl der Ecken
- Gesucht: $F =$ Anzahl der Dreiecke
- Kantenanzahl: $K = \frac{3*F+n}{2} + \frac{3*F+n}{2} - n = 3*F$
- Eulersche Polyedersatz: $n + F - 3*F = 2$
  - nur eine Seite des Polyeders interessant!
  - $n + 2*F - 3*F = 2 \Rightarrow n - F = 2 \Rightarrow F = n - 2$

Wie viele Dreiecke braucht man?

- Gegeben: $n =$ Anzahl der Ecken
- Gesucht: $F =$ Anzahl der Dreiecke
- Kantenanzahl: $K = \frac{3*F+n}{2} + \frac{3*F+n}{2} - n = 3*F$
- Eulersche Polyedersatz: $n + F - 3*F = 2$
  - nur eine Seite des Polyeders interessant!
  - $n + 2*F - 3*F = 2 \Rightarrow n - F = 2 \Rightarrow F = n - 2$

Wie viele Dreiecke braucht man?

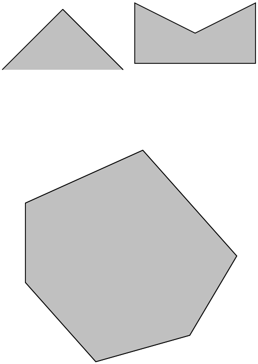
- Gegeben: $n =$ Anzahl der Ecken
- Gesucht: $F =$ Anzahl der Dreiecke
- Kantenanzahl: $K = \frac{3*F+n}{2} + \frac{3*F+n}{2} - n = 3*F$
- Eulersche Polyedersatz: $n + F - 3*F = 2$
  - nur eine Seite des Polyeders interessant!
  - $n + 2*F - 3*F = 2 \Rightarrow n - F = 2 \Rightarrow F = n - 2$

Wie viele Dreiecke braucht man?

- Gegeben: $n =$ Anzahl der Ecken
- Gesucht: $F =$ Anzahl der Dreiecke
- Kantenanzahl: $K = \frac{3*F+n}{2} + \frac{3*F+n}{2} - n = 3*F$
- Eulersche Polyedersatz: $n + F - 3*F = 2$
  - nur eine Seite des Polyeders interessant!
  - $n + 2*F - 3*F = 2 \Rightarrow n - F = 2 \Rightarrow F = n - 2$

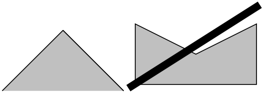
Konvexes Polygon

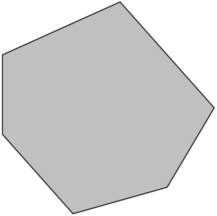
- Umgebung jeder Ecke konvex (Gegenteil: konkav)
- Linie schneidet Polygon maximal $2\times$
  - trivial
  - verbinden Punkt mit allen nicht benachbarten Punkten
  - $O(n)$



Konvexes Polygon

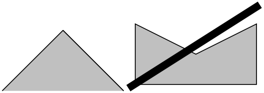
- Umgebung jeder Ecke konvex (Gegenteil: konkav)
- Linie schneidet Polygon maximal 2×
  - trivial
  - verbinden Punkt mit allen nicht benachbarten Punkten
  - $O(n)$

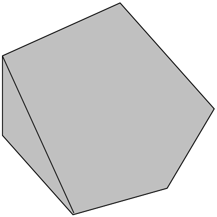




Konvexes Polygon

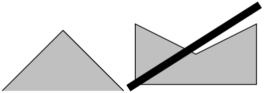
- Umgebung jeder Ecke konvex (Gegenteil: konkav)
- Linie schneidet Polygon maximal 2×
  - trivial
  - verbinden Punkt mit allen nicht benachbarten Punkten
  - $O(n)$

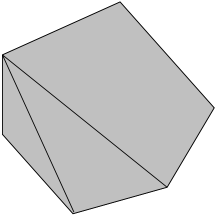




Konvexes Polygon

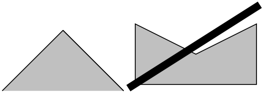
- Umgebung jeder Ecke konvex (Gegenteil: konkav)
- Linie schneidet Polygon maximal 2×
  - trivial
  - verbinden Punkt mit allen nicht benachbarten Punkten
  - $O(n)$

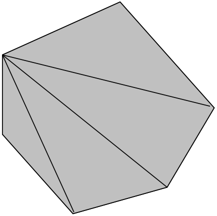




Konvexes Polygon

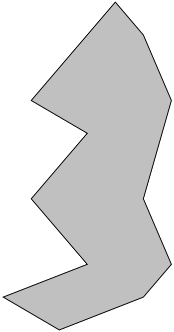
- Umgebung jeder Ecke konvex (Gegenteil: konkav)
- Linie schneidet Polygon maximal 2×
  - trivial
  - verbinden Punkt mit allen nicht benachbarten Punkten
  - $O(n)$





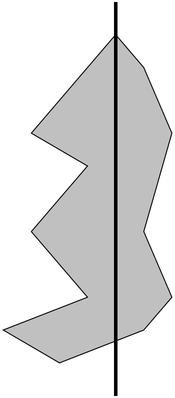
Monotones Polygon

- monoton zu einer Geraden, wenn alle Senkrechten der Geraden Polygon max. 2× schneiden
- 2 Seiten sind monoton
- y-monoton: monoton bezüglich der y-Achse



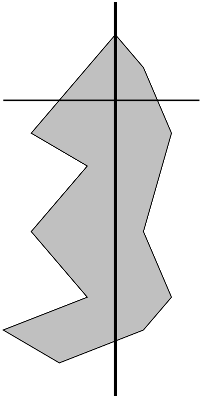
Monotones Polygon

- monoton zu einer Geraden, wenn alle Senkrechten der Geraden Polygon max. 2× schneiden
- 2 Seiten sind monoton
- y-monoton: monoton bezüglich der y-Achse



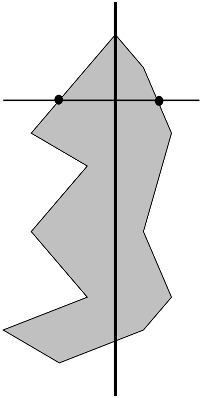
Monotones Polygon

- monoton zu einer Geraden, wenn alle Senkrechten der Geraden Polygon max. 2× schneiden
- 2 Seiten sind monoton
- y-monoton: monoton bezüglich der y-Achse



Monotones Polygon

- monoton zu einer Geraden, wenn alle Senkrechten der Geraden Polygon max. $2\times$ schneiden
- 2 Seiten sind monoton
- y-monoton: monoton bezüglich der y-Achse

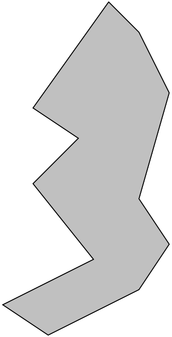


Triangulation

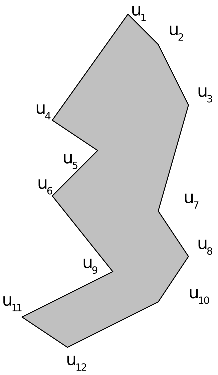
- Sweepline Algorithmus
    - in y-Richtung bei y-monotonen Polygon
    - Sortieren Punkte (Merge der Seiten) nach y (2. Kriterium x) $\rightarrow u_1, \ldots, u_n$
- Stack S, der nicht bearbeiteten Punkte erstellen
    - Initialisierung mit u_1 und u_2

```
Für i=3 bis n
  Wenn ui auf anderer Seite als S.top
    Diagonale von ui zu Punkten in S bis auf Letzten
    Entferne alle Punkte aus S
    Lege ui-1 und ui auf S
  Sonst
    Solange S.top-1 nicht für ui von S.top verdeckt wird
      Erstelle Diagonale von ui nach S.top-1 und entferne S.top
    Packe letzten Punkt und ui in S

Füge Diagonalen von un zu Punkten in S bis auf
  Ersten und Letzten ein
```

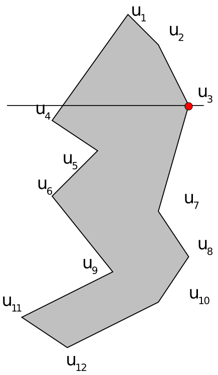
- Laufzeit: $O(n)$
  - pro Punkt O(1 + herausgenommene Punkte)
  - kein Punkt mehr als 2x zum Stack hinzugefügt



- Laufzeit: $O(n)$
  - pro Punkt O(1 + herausgenommene Punkte)
  - kein Punkt mehr als 2x zum Stack hinzugefügt

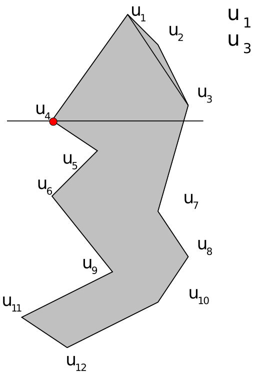


- Laufzeit: $O(n)$
  - pro Punkt O(1 + herausgenommene Punkte)
  - kein Punkt mehr als 2x zum Stack hinzugefügt

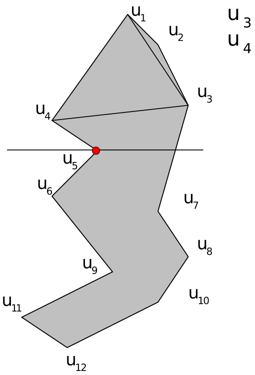


u 1
u 2

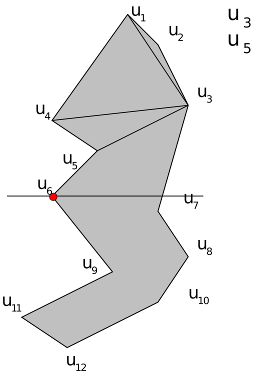
- Laufzeit: $O(n)$
  - pro Punkt O(1 + herausgenommene Punkte)
  - kein Punkt mehr als 2x zum Stack hinzugefügt



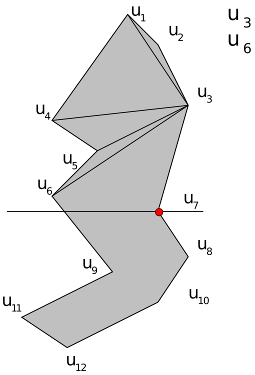
- Laufzeit: $O(n)$
  - pro Punkt O(1 + herausgenommene Punkte)
  - kein Punkt mehr als 2x zum Stack hinzugefügt



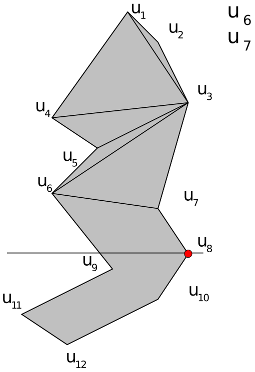
- Laufzeit: $O(n)$
    - pro Punkt O(1 + herausgenommene Punkte)
    - kein Punkt mehr als 2x zum Stack hinzugefügt



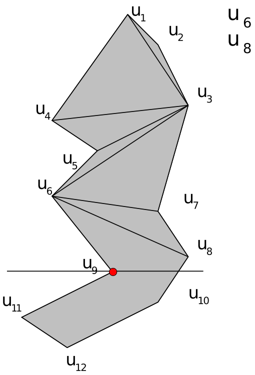
- Laufzeit: $O(n)$
    - pro Punkt O(1 + herausgenommene Punkte)
    - kein Punkt mehr als 2x zum Stack hinzugefügt



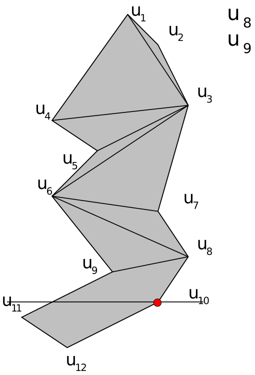
- Laufzeit: $O(n)$
  - pro Punkt O(1 + herausgenommene Punkte)
  - kein Punkt mehr als 2x zum Stack hinzugefügt



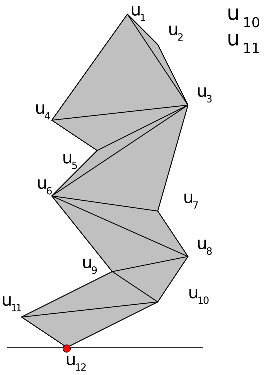
- Laufzeit: $O(n)$
  - pro Punkt O(1 + herausgenommene Punkte)
  - kein Punkt mehr als 2x zum Stack hinzugefügt



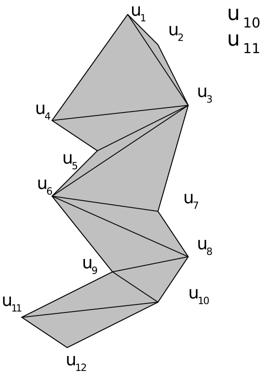
- Laufzeit: $O(n)$
    - pro Punkt O(1 + herausgenommene Punkte)
    - kein Punkt mehr als 2x zum Stack hinzugefügt



- Laufzeit: $O(n)$
  - pro Punkt O(1 + herausgenommene Punkte)
  - kein Punkt mehr als 2x zum Stack hinzugefügt

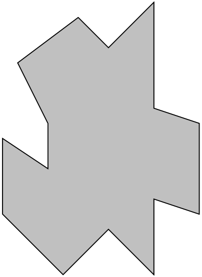


- Laufzeit: $O(n)$
    - pro Punkt O(1 + herausgenommene Punkte)
    - kein Punkt mehr als 2x zum Stack hinzugefügt



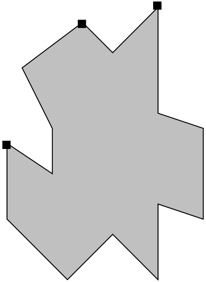
Einfache Polygone

- Polygon dessen Kanten sich nur in den Ecken schneiden
- 5 Arten von Punkten:
  - Startpunkt 
  - Endpunkt 
  - Verbindungspunkt 
  - Teilungspunkt 
  - normalen Punkt 



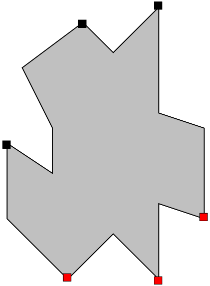
Einfache Polygone

- Polygon dessen Kanten sich nur in den Ecken schneiden
- 5 Arten von Punkten:
    - Startpunkt 
    - Endpunkt 
    - Verbindungspunkt 
    - Teilungspunkt 
    - normalen Punkt 



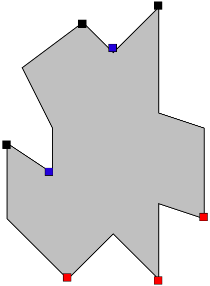
Einfache Polygone

- Polygon dessen Kanten sich nur in den Ecken schneiden
- 5 Arten von Punkten:
  - Startpunkt 
  - Endpunkt 
  - Verbindungspunkt 
  - Teilungspunkt 
  - normalen Punkt 



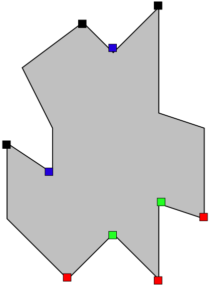
Einfache Polygone

- Polygon dessen Kanten sich nur in den Ecken schneiden
- 5 Arten von Punkten:
  - Startpunkt 
  - Endpunkt 
  - Verbindungspunkt 
  - Teilungspunkt 
  - normalen Punkt 



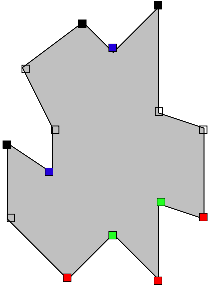
Einfache Polygone

- Polygon dessen Kanten sich nur in den Ecken schneiden
- 5 Arten von Punkten:
    - Startpunkt
    - Endpunkt
    - Verbindungspunkt
    - Teilungspunkt
    - normalen Punkt



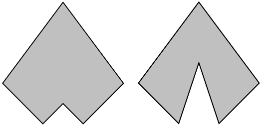
Einfache Polygone

- Polygon dessen Kanten sich nur in den Ecken schneiden
- 5 Arten von Punkten:
  - Startpunkt 
  - Endpunkt 
  - Verbindungspunkt 
  - Teilungspunkt 
  - normalen Punkt 



Triangulation

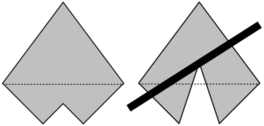
- Abschneiden konvexer Ecken
- Ecke darf keine weiteren Punkte enthalten: Ohr



- Test:
  - ist konvexe Ecke?
  - schneidet sie keine Kanten?

Triangulation

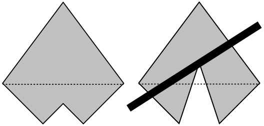
- Abschneiden konvexer Ecken
- Ecke darf keine weiteren Punkte enthalten: Ohr



- Test:
  - ist konvexe Ecke?
  - schneidet sie keine Kanten?

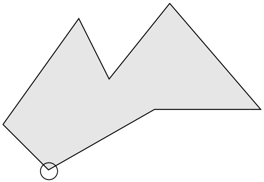
Triangulation

- Abschneiden konvexer Ecken
- Ecke darf keine weiteren Punkte enthalten: Ohr

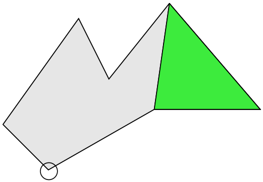


- Test:
  - ist konvexe Ecke?
  - schneidet sie keine Kanten?

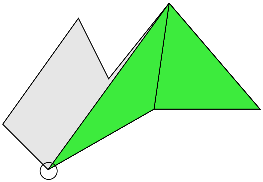
```
Wenn n > 3
  Für jede potentielle Diagonale (i, i+2)
    Wenn Diagonale(i, i+2) ist
      Entferne Ohr bei pi
      Wiederhole mit restlichem Polygon
```



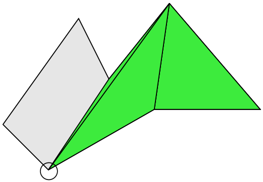
```
Wenn n > 3
  Für jede potentielle Diagonale (i, i+2)
    Wenn Diagonale(i, i+2) ist
      Entferne Ohr bei pi
      Wiederhole mit restlichem Polygon
```



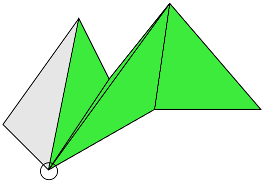
```
Wenn n > 3
  Für jede potentielle Diagonale (i, i+2)
    Wenn Diagonale(i, i+2) ist
      Entferne Ohr bei pi
      Wiederhole mit restlichem Polygon
```



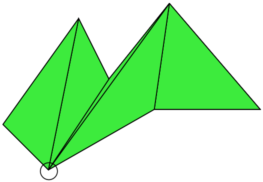
```
Wenn n > 3
  Für jede potentielle Diagonale (i, i+2)
    Wenn Diagonale(i, i+2) ist
      Entferne Ohr bei pi
      Wiederhole mit restlichem Polygon
```



```
Wenn n > 3
  Für jede potentielle Diagonale (i, i+2)
    Wenn Diagonale(i, i+2) ist
      Entferne Ohr bei pi
      Wiederhole mit restlichem Polygon
```



```
Wenn n > 3
  Für jede potentielle Diagonale (i, i+2)
    Wenn Diagonale(i, i+2) ist
      Entferne Ohr bei pi
      Wiederhole mit restlichem Polygon
```



Eigenschaften

- rekursive Funktion
- bei jedem Aufruf suche nach erstem Ohr $O(n^2)$
- insgesamt $O(n^3)$

Besserer Algorithmus: Algorithmus von Kong - $O(n^2)$

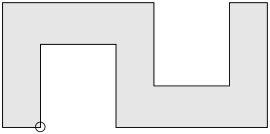
Eigenschaften

- rekursive Funktion
- bei jedem Aufruf suche nach erstem Ohr $O(n^2)$
- insgesamt $O(n^3)$

Besserer Algorithmus: Algorithmus von Kong - $O(n^2)$

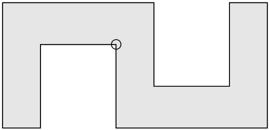
Algorithmus von Kong

```
Starte an Position 3
Solange n > 3
  Wenn Dreieck Ohr(i-2, i-1, i) ist
    Entferne Ohr
    Setze Position auf i-2
  Sonst gehe eine Position weiter
```



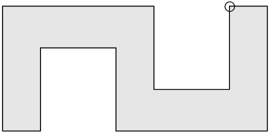
Algorithmus von Kong

```
Starte an Position 3
Solange n > 3
  Wenn Dreieck Ohr(i-2, i-1, i) ist
    Entferne Ohr
    Setze Position auf i-2
  Sonst gehe eine Position weiter
```



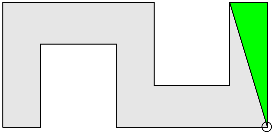
Algorithmus von Kong

```
Starte an Position 3
Solange n > 3
  Wenn Dreieck Ohr(i-2, i-1, i) ist
    Entferne Ohr
    Setze Position auf i-2
  Sonst gehe eine Position weiter
```



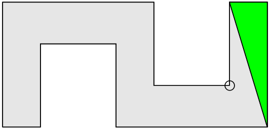
Algorithmus von Kong

```
Starte an Position 3
Solange n > 3
  Wenn Dreieck Ohr(i-2, i-1, i) ist
    Entferne Ohr
    Setze Position auf i-2
  Sonst gehe eine Position weiter
```



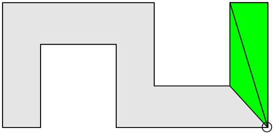
Algorithmus von Kong

```
Starte an Position 3
Solange n > 3
  Wenn Dreieck Ohr(i-2, i-1, i) ist
    Entferne Ohr
    Setze Position auf i-2
  Sonst gehe eine Position weiter
```



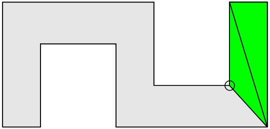
Algorithmus von Kong

```
Starte an Position 3
Solange n > 3
  Wenn Dreieck Ohr(i-2, i-1, i) ist
    Entferne Ohr
    Setze Position auf i-2
  Sonst gehe eine Position weiter
```



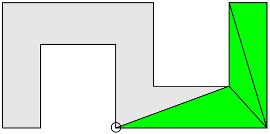
Algorithmus von Kong

```
Starte an Position 3
Solange n > 3
  Wenn Dreieck Ohr(i-2, i-1, i) ist
    Entferne Ohr
    Setze Position auf i-2
  Sonst gehe eine Position weiter
```



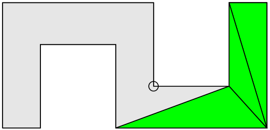
Algorithmus von Kong

```
Starte an Position 3
Solange n > 3
  Wenn Dreieck Ohr(i-2, i-1, i) ist
    Entferne Ohr
    Setze Position auf i-2
  Sonst gehe eine Position weiter
```



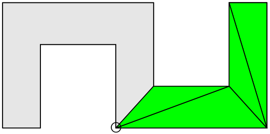
Algorithmus von Kong

```
Starte an Position 3
Solange n > 3
  Wenn Dreieck Ohr(i-2, i-1, i) ist
    Entferne Ohr
    Setze Position auf i-2
  Sonst gehe eine Position weiter
```



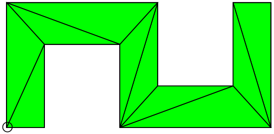
Algorithmus von Kong

```
Starte an Position 3
Solange n > 3
  Wenn Dreieck Ohr(i-2, i-1, i) ist
    Entferne Ohr
    Setze Position auf i-2
  Sonst gehe eine Position weiter
```



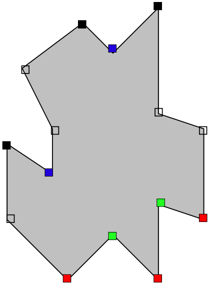
Algorithmus von Kong

```
Starte an Position 3
Solange n > 3
  Wenn Dreieck Ohr(i-2, i-1, i) ist
    Entferne Ohr
    Setze Position auf i-2
  Sonst gehe eine Position weiter
```



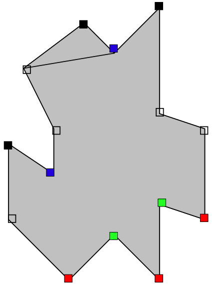
Partitionierung von Polygonen

- Algorithmen zur Triangulation zu langsam (Kong $O(n^2)$)
- Unterteilung der Polygone in einfachere Polygone (konvexe und monotone Polygone)



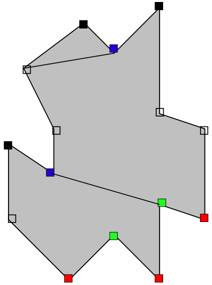
Partitionierung von Polygonen

- Algorithmen zur Triangulation zu langsam (Kong $O(n^2)$)
- Unterteilung der Polygone in einfachere Polygone (konvexe und monotone Polygone)



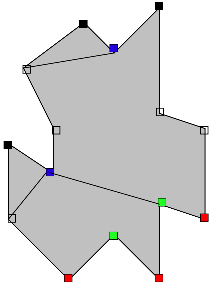
Partitionierung von Polygonen

- Algorithmen zur Triangulation zu langsam (Kong $O(n^2)$)
- Unterteilung der Polygone in einfachere Polygone (konvexe und monotone Polygone)



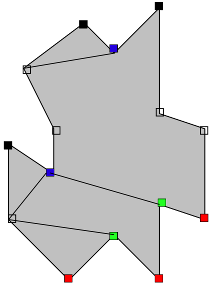
Partitionierung von Polygonen

- Algorithmen zur Triangulation zu langsam (Kong $O(n^2)$)
- Unterteilung der Polygone in einfachere Polygone (konvexe und monotone Polygone)



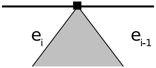
Partitionierung von Polygonen

- Algorithmen zur Triangulation zu langsam (Kong $O(n^2)$)
- Unterteilung der Polygone in einfachere Polygone (konvexe und monotone Polygone)

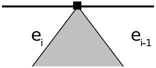


- Verbinde Teilungspunkt mit geeigneten Punkt unterhalb
- Verbinde Verbindungspunkte mit geeigneten Punkt oberhalb
- Diagonalen dürfen sich nicht überschneiden
- Verwenden Sweepline Algorithmus
  - Sortieren Punkte (auch über Prioritätsliste möglich) und Erstellen leeren binären Baum T zum Speichern der Kanten
  - Rufen für Punkte entsprechende Funktion auf

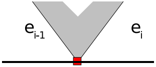
```
StartPunkt(pi)
Füge ei in T ein
Setze helper(ei) auf pi

EndPunkt(pi)
Wenn helper(ei-1) ein Verbindungsp.
Erstelle Diagonale pi-helper(ei-1)
Lösche ei-1 aus T

Trennungspunkt(pi)
ej=Suche T nach Kante direkt links pi
Erstelle Diagonale pi-helper(ej)
Füge ei in T
Setze helper(ei) auf pi
```




```
StartPunkt(pi)
Füge ei in T ein
Setze helper(ei) auf pi

EndPunkt(pi)
Wenn helper(ei-1) ein Verbindungsp.
Erstelle Diagonale pi-helper(ei-1)
Lösche ei-1 aus T

Trennungspunkt(pi)
ej=Suche T nach Kante direkt links pi
Erstelle Diagonale pi-helper(ej)
Füge ei in T
Setze helper(ei) auf pi
```

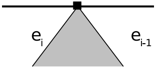



```
StartPunkt(pi)
Füge ei in T ein
Setze helper(ei) auf pi
```

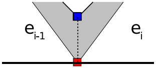



```
EndPunkt(pi)
Wenn helper(ei-1) ein Verbindungsp.
Erstelle Diagonale pi-helper(ei-1)
Lösche ei-1 aus T
```

```
Trennungspunkt(pi)
ej=Suche T nach Kante direkt links pi
Erstelle Diagonale pi-helper(ej)
Füge ei in T
Setze helper(ei) auf pi
```



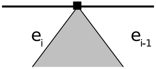

```
StartPunkt(pi)
Füge ei in T ein
Setze helper(ei) auf pi
```



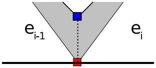

```
EndPunkt(pi)
Wenn helper(ei-1) ein Verbindungsp.
Erstelle Diagonale pi-helper(ei-1)
Lösche ei-1 aus T
```

```
Trennungspunkt(pi)
ej=Suche T nach Kante direkt links pi
Erstelle Diagonale pi-helper(ej)
Füge ei in T
Setze helper(ei) auf pi
```

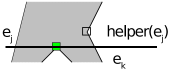
```
StartPunkt(pi)
Füge ei in T ein
Setze helper(ei) auf pi
```



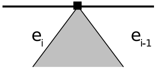

```
EndPunkt(pi)
Wenn helper(ei-1) ein Verbindungsp.
Erstelle Diagonale pi-helper(ei-1)
Lösche ei-1 aus T
```



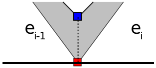

```
Trennungspunkt(pi)
ej=Suche T nach Kante direkt links pi
Erstelle Diagonale pi-helper(ej)
Füge ei in T
Setze helper(ei) auf pi
```



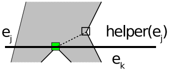
```
StartPunkt(pi)
Füge ei in T ein
Setze helper(ei) auf pi
```



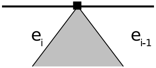

```
EndPunkt(pi)
Wenn helper(ei-1) ein Verbindungsp.
Erstelle Diagonale pi-helper(ei-1)
Lösche ei-1 aus T
```



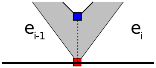

```
Trennungspunkt(pi)
ej=Suche T nach Kante direkt links pi
Erstelle Diagonale pi-helper(ej)
Füge ei in T
Setze helper(ei) auf pi
```



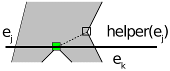
```
StartPunkt(pi)
Füge ei in T ein
Setze helper(ei) auf pi
```

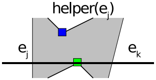



```
EndPunkt(pi)
Wenn helper(ei-1) ein Verbindungsp.
Erstelle Diagonale pi-helper(ei-1)
Lösche ei-1 aus T
```

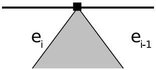



```
Trennungspunkt(pi)
ej=Suche T nach Kante direkt links pi
Erstelle Diagonale pi-helper(ej)
Füge ei in T
Setze helper(ei) auf pi
```

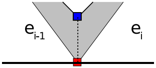




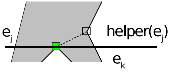
```
StartPunkt(pi)
Füge ei in T ein
Setze helper(ei) auf pi
```

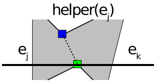



```
EndPunkt(pi)
Wenn helper(ei-1) ein Verbindungsp.
Erstelle Diagonale pi-helper(ei-1)
Lösche ei-1 aus T
```



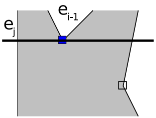

```
Trennungspunkt(pi)
ej=Suche T nach Kante direkt links pi
Erstelle Diagonale pi-helper(ej)
Füge ei in T
Setze helper(ei) auf pi
```

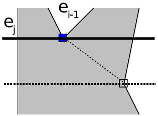




```
Verbindungspunkt(pi)
Wenn helper(ei) ist Verbindungsp.
Erstelle Diagonale pi-helper(ei-1)
Lösche ei-1 aus T
ej=Suche T nach Kanten direkt links pi
Wenn helper(ej) Verbindungsp.
Erstelle Diagonale pi-helper(ej)
```

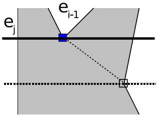
```
Verbindungspunkt(pi)
Wenn helper(ei) ist Verbindungsp.
Erstelle Diagonale pi-helper(ei-1)
Lösche ei-1 aus T
ej=Suche T nach Kanten direkt links pi
Wenn helper(ej) Verbindungsp.
Erstelle Diagonale pi-helper(ej)
```

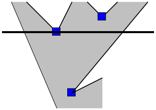




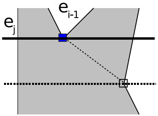

```
Verbindungspunkt(pi)
Wenn helper(ei) ist Verbindungsp.
Erstelle Diagonale pi-helper(ei-1)
Lösche ei-1 aus T
ej=Suche T nach Kanten direkt links pi
Wenn helper(ej) Verbindungsp.
Erstelle Diagonale pi-helper(ej)
```

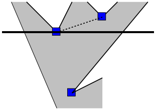
```
Verbindungspunkt(pi)
Wenn helper(ei) ist Verbindungsp.
Erstelle Diagonale pi-helper(ei-1)
Lösche ei-1 aus T
ej=Suche T nach Kanten direkt links pi
Wenn helper(ej) Verbindungsp.
Erstelle Diagonale pi-helper(ej)
```



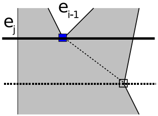


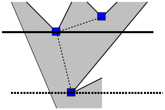
```
Verbindungspunkt(pi)
Wenn helper(ei) ist Verbindungsp.
Erstelle Diagonale pi-helper(ei-1)
Lösche ei-1 aus T
ej=Suche T nach Kanten direkt links pi
Wenn helper(ej) Verbindungsp.
Erstelle Diagonale pi-helper(ej)
```





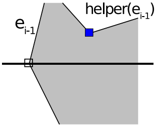
```
Verbindungspunkt(pi)
Wenn helper(ei) ist Verbindungsp.
Erstelle Diagonale pi-helper(ei-1)
Lösche ei-1 aus T
ej=Suche T nach Kanten direkt links pi
Wenn helper(ej) Verbindungsp.
Erstelle Diagonale pi-helper(ej)
```

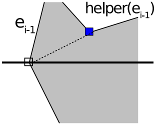




```
Normaler Punkt(pi)
Wenn Inneres von p rechts von pi und
helper(ei-1) ist Verbindungspunkt
Erstelle Diagonale pi-helper(ei-1)
Lösche ei-1 aus T
Füge ei in T ein; setze helper(ei)=pi
sonst ej=Suche T nach Kante direkt links pi
wenn helper(ej) ist Verbindungspunkt
Erstelle Diagonale pi-helper(ej)
```

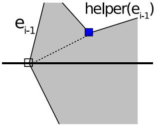
```
Normaler Punkt(pi)
Wenn Inneres von p rechts von pi und
helper(ei-1) ist Verbindungspunkt
Erstelle Diagonale pi-helper(ei-1)
Lösche ei-1 aus T
Füge ei in T ein; setze helper(ei)=pi
sonst ej=Suche T nach Kante direkt links pi
wenn helper(ej) ist Verbindungspunkt
Erstelle Diagonale pi-helper(ej)
```

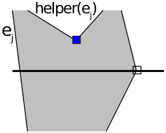




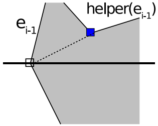

```
Normaler Punkt(pi)
Wenn Inneres von p rechts von pi und
helper(ei-1) ist Verbindungspunkt
Erstelle Diagonale pi-helper(ei-1)
Lösche ei-1 aus T
Füge ei in T ein; setze helper(ei)=pi
sonst ej=Suche T nach Kante direkt links pi
wenn helper(ej) ist Verbindungspunkt
Erstelle Diagonale pi-helper(ej)
```

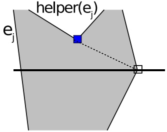
```
Normaler Punkt(pi)
Wenn Inneres von p rechts von pi und
helper(ei-1) ist Verbindungspunkt
Erstelle Diagonale pi-helper(ei-1)
Lösche ei-1 aus T
Füge ei in T ein; setze helper(ei)=pi
sonst ej=Suche T nach Kante direkt links pi
wenn helper(ej) ist Verbindungspunkt
Erstelle Diagonale pi-helper(ej)
```





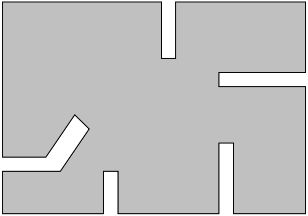
```
Normaler Punkt(pi)
Wenn Inneres von p rechts von pi und
helper(ei-1) ist Verbindungspunkt
Erstelle Diagonale pi-helper(ei-1)
Lösche ei-1 aus T
Füge ei in T ein; setze helper(ei)=pi
sonst ej=Suche T nach Kante direkt links pi
wenn helper(ej) ist Verbindungspunkt
Erstelle Diagonale pi-helper(ej)
```





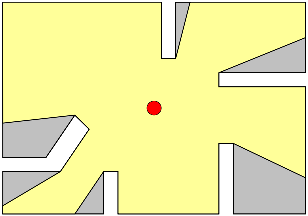
Art Gallery Theorem

- Raum einer Kunstgalerie der als Polygon wiedergegeben werden kann
- Wie viele Wächter/Kameras benötigt man?
- Wächter:
    - fixiert
    - 360° Sicht



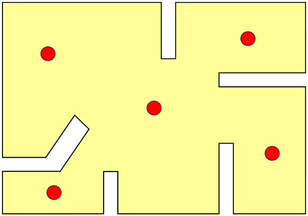
Art Gallery Theorem

- Raum einer Kunstgalerie der als Polygon wiedergegeben werden kann
- Wie viele Wächter/Kameras benötigt man?
- Wächter:
  - fixiert
  - 360° Sicht

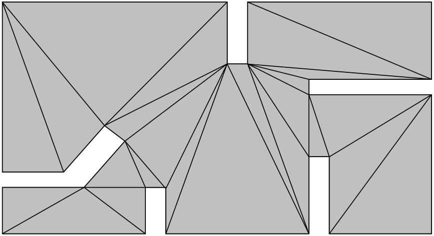


Art Gallery Theorem

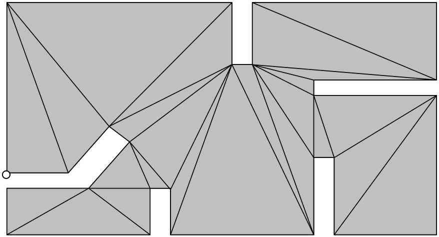
- Raum einer Kunstgalerie der als Polygon wiedergegeben werden kann
- Wie viele Wächter/Kameras benötigt man?
- Wächter:
  - fixiert
  - 360° Sicht



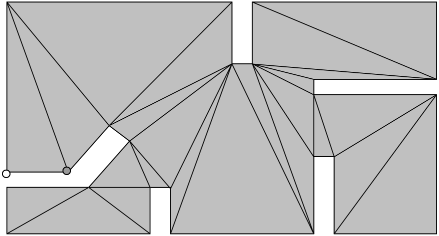
- Jeder Wächter kann mindestens ein Dreieck bewachen:
  - Einfaches Polygon – n-2 Wächter
  - nur eine obere Grenze
  - Verringerung durch Platzierung an geeigneten Ecken
- Wächter kann mindestens ein konvexes Polygon bewachen



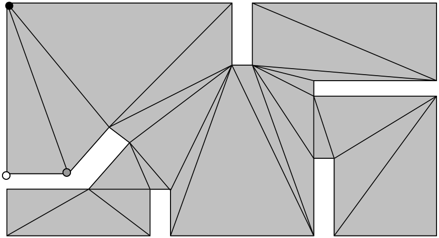
- Jeder Wächter kann mindestens ein Dreieck bewachen:
  - Einfaches Polygon – n-2 Wächter
  - nur eine obere Grenze
  - Verringerung durch Platzierung an geeigneten Ecken
- Wächter kann mindestens ein konvexes Polygon bewachen



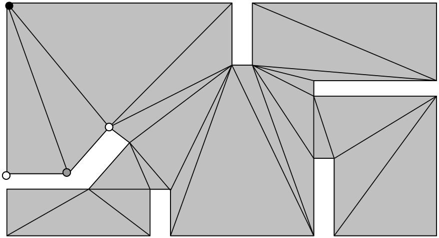
- Jeder Wächter kann mindestens ein Dreieck bewachen:
  - Einfaches Polygon – n-2 Wächter
  - nur eine obere Grenze
  - Verringerung durch Platzierung an geeigneten Ecken
- Wächter kann mindestens ein konvexes Polygon bewachen



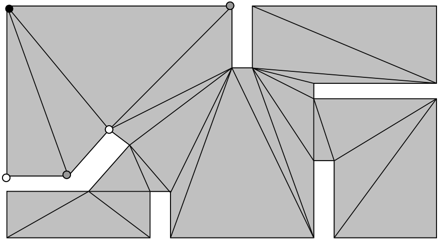
- Jeder Wächter kann mindestens ein Dreieck bewachen:
  - Einfaches Polygon – n-2 Wächter
  - nur eine obere Grenze
  - Verringerung durch Platzierung an geeigneten Ecken
- Wächter kann mindestens ein konvexes Polygon bewachen



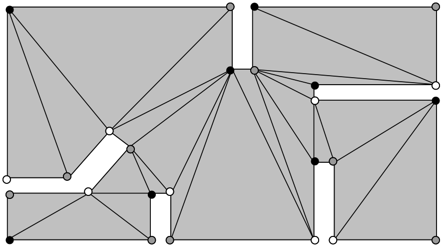
- Jeder Wächter kann mindestens ein Dreieck bewachen:
  - Einfaches Polygon – n-2 Wächter
  - nur eine obere Grenze
  - Verringerung durch Platzierung an geeigneten Ecken
- Wächter kann mindestens ein konvexes Polygon bewachen



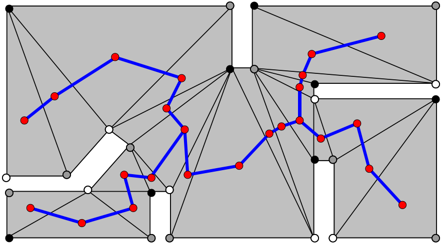
- Jeder Wächter kann mindestens ein Dreieck bewachen:
  - Einfaches Polygon – n-2 Wächter
  - nur eine obere Grenze
  - Verringerung durch Platzierung an geeigneten Ecken
- Wächter kann mindestens ein konvexes Polygon bewachen



- Jeder Wächter kann mindestens ein Dreieck bewachen:
  - Einfaches Polygon – n-2 Wächter
  - nur eine obere Grenze
  - Verringerung durch Platzierung an geeigneten Ecken
- Wächter kann mindestens ein konvexes Polygon bewachen



- Jeder Wächter kann mindestens ein Dreieck bewachen:
  - Einfaches Polygon – n-2 Wächter
  - nur eine obere Grenze
  - Verringerung durch Platzierung an geeigneten Ecken
- Wächter kann mindestens ein konvexes Polygon bewachen

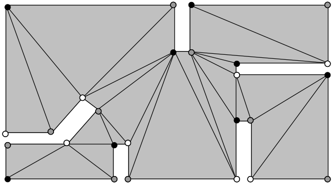


Finden der Ecken

- Färben der Eckpunkte
- Jede Kante hat 2 verschieden gefärbte Eckpunkte
  - 3 Farben notwendig (schwarz, weiß, grau)

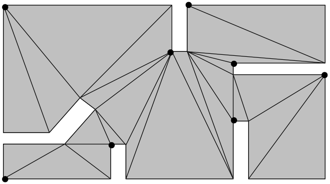
Auswahl der Ecken

- Auswahl einer Farbe
  - jedes Dreieck wird bewacht
- Auswahl der Farbe mit den wenigsten Punkten
  - Farbe mit den meisten Schnittpunkten
  - jedes Dreieck wird bewacht
  - im Beispiel:
    - 8 Schwarze
    - 8 Weiße
    - 10 Graue



Auswahl der Ecken

- Auswahl einer Farbe
  - jedes Dreieck wird bewacht
- Auswahl der Farbe mit den wenigsten Punkten
  - Farbe mit den meisten Schnittpunkten
  - jedes Dreieck wird bewacht
  - im Beispiel:
    - 8 Schwarze
    - 8 Weiße
    - 10 Graue



Quellenangabe

- Joseph O'Rourke: Computational Geometry in C, Cambridge University Press, 1994
- http://wwwmath.uni-muenster.de/u/jacobm/Lehre/AlGeo/
- http://www-gs.informatik.tu-cottbus.de/
- http://www.cs.berkeley.edu/~jrs/mesh/
- http://www.mema.ucl.ac.be/~wu/FSA2716-2002/project.html
- http://de.wikipedia.org/

Fragen?