# Aufteilung eines einfachen, konvexen Polygons

Die Eingangsdaten zur Berechnung der Flächenaufteilung sind ein konvexes Polygon CP sowie eine (nicht leere) Liste von Standorte S(CP), die auf dem Rand von CP liegen. Für jeden der n Standorte Si ist der geforderte Flächenanteil ci mit 0 < ci < 1 gegeben, sodass gilt. Das konvexe Polygon CP soll in n Polygone zerlegt werden, sodass jeder Standort auf dem Rand eines Polygons mit Fläche ci \* Area(CP) liegt. Vorab wird für jeden der n Standorte zunächst die geforderte Fläche mit AreaRequired(Si) = ci \* Area(CP) berechnet.

## Grundidee

Bei der nachfolgend beschriebenen Lösung des Problems wird das konvexe Eingangs-Polygon CP mithilfe von Liniensegmenten schrittweise zerlegt. Jedes Liniensegment L ist hierbei vom Startpunkt Ls zum Endpunkt Le orientiert. Die jeder Teilung entstehen zwei Polygone, welche entsprechend ihrer Lage zum Liniensegment die Bezeichnungen für das *rechts* und für das *links* des Liniensegments liegenden Polygons erhalten. Die exakte Positionierung der Liniensegmente wird in Kapitel xxx erläutert. An dieser Stelle soll genügen, dass die Liniensegmente so positioniert werden, dass die Fläche von der geforderten Fläche der auf dem Rand von liegenden Standorte entspricht ( analog). Die Zerlegung wird für jedes (verbleibende) Polygon oder so oft wiederholt, bis je Polygon nur noch ein Standort auf dessen Rand liegt[[1]](#footnote-1).

|  |  |  |
| --- | --- | --- |
|  |  |  |

Abbildung 2: Schematische Zerlegung eines konvexen Polygons CP in vier konvexe Polygone CP1 ... CP4 mit c1 ... c4

Aus CP entstehende, konvexe Polygone, werden mit CPj notiert. Mit den genannten Überlegungen lässt sich ein *divide-and-conquer* - Algorithmus zur Flächenaufteilung eines konvexen Polygons - basierend auf n Standorten - nun folgendermaßen skizzieren:

1 // Input: Convex polygon CP, described with arrays W, V, S

2 if Length(S(CP)) > 1 then

3 ConvexPolygons.add(CP)

4 end

5 while Length(ConvexPolygons) > 0 do

6 CP = ReadFirst(ConvexPolygons)

7 PrL, PlL = ConvexDivide(CP) // Divide into two convex polygons

8 if Length(S(PrL)) > 1 then ConvexPolygons.add(PrL) else Output(PrL) end

9 if Length(S(PlL)) > 1 then ConvexPolygons.add(PlL) else Output(PlL) end

Nach einer Teilung eines Polygons durch eine Linie L wird für die beiden Polygone und jeweils geprüft, ob mehr als 2 Standorte vorliegen. Falls nein, ist der Zielzustand für diesen Standort erreicht und es ist keine weitere Flächenaufteilung erforderlich. Falls ja, wird das Polygon wieder in die Warteschlange der zu teilenden Polygone eingereiht.

## Lösung des Problems

Aus vorangegangenem Kapitel bleibt noch offen, wie genau die Aufteilung eines konvexen Polygons CP in die Polygone und erfolgt, sodass in jedem Iterationsschritt AreaRequired(S1 ... Si) == Area() und AreaRequired(Si+1 ... Sn) == Area() gilt. Konkret ist zu klären, wie Anfangs- und Endpunkt der Schnittlinien positioniert werden. Dieses Kapitel geht daher auf die konkrete Umsetzung des Algorithmus für ConvexDivide()ein.

Eingangsdaten sind - wie zuvor - ein konvexes Polygon CP zusammen mit einer Liste von Standorten S(CP), die auf dem Rand von CP liegen. Durch die Bedingungen in den Zeilen 2-4, 8 und 9 wird sichergestellt, dass mindestens zwei Standorte auf dem Rand von CP liegen. Für jeden der Standorte Si ist der geforderte Flächenanteil ci mit 0 < ci < 1 gegeben.

Initialisierung vor jedem Aufruf von ConvexDivide(): Der Startpunkt Ls der Linie L wird mit den Koordinaten des ersten Punkts der Liste W initialisiert, wobei dieser nach Definition ein Polygonpunkt ist, sodass gilt w1 V. Der Endpunkt Le wird mit den Koordinaten des ersten Standorts in W initialisiert und mit wk notiert, wobei k der Index in W ist, bei welchem der erste Standort liegt. Da die Standorte nach ihrem Vorkommen auf dem Weg von v1 nach vn geordnet sind, ist sichergestellt, dass die Standorte S2 … Sn alle links der Linie L liegen. Bei einer Zerlegung mit einer so initialisierten Linie würde S() == S1 und S() == Si+1 … Sn gelten.

Je nach Fläche von und der in S1 benötigten Fläche AreaRequired(S1) werden nun drei Fälle unterschieden:

**Fall 1: Area() > AreaRequired(S1)**

Nach der Initialisierung der Linie L wird festgestellt, dass die Fläche von größer ist als die benötigte Fläche von S1. In diesem Fall erfolgt eine Verkleinerung von Area() unter Beibehaltung von S() == S1. Dies geschieht, indem Le so lange im Gegenuhrzeigersinn entlang des Polygons verschoben wird, bis Area() == AreaRequired(S1) gilt. Diese Vorgehensweise funktioniert, da

* auf dem Weg von w1 zu wk kein weiterer Standort liegt, d.h. die benötigte Fläche von nicht steigen kann
* Area() stetig kleiner wird und spätestens 0 wird, wenn Ls = S1 gilt.
* Le konstant bleibt, d.h. S1 immer Teil von bleibt

Wenn die Bedingung S() == S1 eintritt, erfolgt eine Polygonzerlegung. muss nicht weiter zerlegt werden, lediglich S1 auf dessen Standort liegt. muss weiter zerlegt werden, falls Length(S()) >= 2 gilt.

**Fall 2: Area() < AreaRequired(S1):**

Nach der Initialisierung der Linie L wird festgestellt, dass die Fläche von kleiner ist als die benötigte Fläche von S1. In diesem Fall erfolgt eine Vergrößerung von Area() mit dem Ziel, die Anforderung von S1 zu erfüllen. Hierbei wird der Punkt Le auf den nächsten in W vorkommenden Polygonpunkt oder Standort (wk+1) bewegt und die Anforderung erneut geprüft.

Bei jeder Prüfung der Anforderung wird zusätzlich der vorherige Punkt – sofern dieser ein Standort S1 ist – ausgewertet zur Anforderung von S() hinzugenommen. Bei Fall 2 kann die benötigte Fläche AreaRequired() demnach ansteigen, sodass ein „Weiterrücken“ von zwar zu einer größeren Fläche, nicht aber unbedingt zu einem günstigeren Verhältnis aus Area() / AreaRequired(S()) führt.

Durch „Weiterrücken“ von Le können nun zwei Fälle eintreten:

Fall 2.1: Le Sn && Area(PrL) > AreaRequired(S(PrL)). In diesem Fall wird der Endpunkt Le inkrementell im Uhrzeigersinn entlang des Polygons bewegt, bis Area(PrL) == AreaRequired(S(PrL)) gilt.

Fall 2.2: Le == Sn && Area(PrL) < AreaRequired(S(PrL)). In diesem Fall wird der Anfangspunkt Ls inkrementell im Uhrzeigersinn entlang des Polygons bewegt, bis Area(PrL) == AreaRequired(S(PrL)) gilt.

|  |  |  |
| --- | --- | --- |
|  |  |  |

Abbildung : Fall 1, 2.1 und 2.2

Mit den Fällen 1 und 2 lässt sich nun der Algorithmus von ConvexDivide() wie folgt konstruieren:

1 // Algorithm ConvexDivide()

2 // Input: Convex polygon CP, described with arrays W, V, S

3 Ls = w1, Le = wk // k = index for first Site in W

4 PrL, PlL = cut(W, L)

5 k = 1

6 while Area(PrL) < AreaRequired(S(PrL)) and Le != Sn do

7 if wk-1 != S1 and wk-1.type() == “Site” then

8 S(PrL) = S(PrL) + wk-1

9 end

10 k += 1 // increase k

11 Le = wk // move Le to next point in W

12 PrL, PlL = cut(W, L)

13 if Area(PrL) > AreaRequired(S(PrL)) and Le == S1 then

14 move Le CCW until Area(PrL) == AreaRequired(S(PrL))

15 else if Area(PrL) < AreaRequired(S(PrL))

16 if Le != Sn then

17 move Le CW until Area(PrL) == AreaRequired(S(PrL))

18 else if Le == Sn then

19 move Ls CW until Area(PrL) == AreaRequired(S(PrL))

20 end

21 end

22 PrL, PlL = cut(W, L)

23 return PrL, PlL

## Beispiel

|  |  |  |
| --- | --- | --- |
| Fall 2.1: P1-S1 / P1-P5 / CW | Fall x.x | Fall x.x |
| Fall x.x | Fall x.x | Fall x.x |
| Fall x.x | Fall x.x | Fall x.x |

1. Da aus einer Zerlegung eines konvexen Polygons mit einem Liniensegment immer zwei *konvexe* Polygone und insbesondere kein nicht-konvexes Polygon resultiert, ist dieser Ansatz möglich. [↑](#footnote-ref-1)