# Verallgemeinerung: Aufteilung eines nicht einfachen, nicht konvexen Polygons (SJ)

Es wurde gezeigt, dass einfache, konvexe Polygon in polynomieller Zeit rekursiv in n 1-Standort-Polygone aufgeteilt werden können. Als Verallgemeinerung dient nun die Betrachtung von nicht einfachen, nicht konvexen Polygonen. Zuerst soll die Grundidee des Algorithmus beschrieben werden, um einen Überblick über den vorgestellten Algorithmus zu erlangen. Anschließend wird der Algorithmus zur Aufteilung eines nicht einfachen, nicht konvexen Polygons vorgestellt. Dazu werden zuerst kurz die Schritte der Vorverarbeitung vorgestellt und anschließend die Aufteilung des Polygons. Im Anschluss dient ein Beispiel zur Veranschaulichung des vorgestellten Algorithmus und zum Schluss des Kapitels wird der Sonderfall geschildert, dass Standorte im inneren des Polygons liegen und die Komplexität der Vorgehensweise aufgezeigt.

## Grundidee

In Kapitel 7 wurde bereits erläutert, wie ein einfaches, konvexes Polygon aufgeteilt werden kann. Dieses Vorgehen kann auch bei der Aufteilung komplexerer Polygone verwendet werden, muss jedoch in einigen Punkten erweitert werden.

Als Voraussetzung wird angenommen, dass ein nicht einfaches, nicht konvexes Polygon P bereits in konvexe Teilpolygone CP1, CP2, … CPp zerlegt wurde. Im ersten Schritt werden die Teilpolygone, ähnlich einer Tiefensuche, neu geordnet, um eine feste Bearbeitungsfolge für das weitere Vorgehen zu erhalten. Anschließend werden die Teilpolygone rekursiv aufgeteilt, wie es bereits in Kapitel 7 gezeigt wurde. Allerdings können nun Sonderfälle auftreten, die bei der Zerteilung eines einfachen, konvexen Polygons nicht vorkommen können. Ein solcher Fall besteht daraus, dass CPi weniger Fläche ausfüllt, als durch *AreaRequired(S(CPi))* gefordert ist. In diesem Fall ist CPi Flächen-unvollständig und muss Flächen von anderen Teilpolygonen übernehmen. Andererseits kann es sein, dass einzelne Teilpolygone keinen Standort enthalten oder weniger Fläche ausfüllen, als durch *AreaRequired(S(CPi))* gefordert ist. In diesem Fall ist CPi Standort-unvollständig und andere Teilpolygone müssen Fläche von CPi übernehmen.

Die Neuordnung wird innerhalb der Prozedur *OrderPieces* umgesetzt und die Aufteilung inklusive der Sonderfallbehandlung wird durch die beiden Methoden *NonconvexDivide* und *DetachAndAssign* umgesetzt, die sich gegenseitig rekursiv aufrufen, bis ein n-Standort Polygon in n 1-Standort Polygone aufgeteilt wurde.

## Aufteilung in konvexe Teilpolygone

Als Voraussetzung für die gleichmäßige Aufteilung eines nicht einfachen, nicht konvexen Polygons wird angenommen, dass das Polygon bereits in konvexe Teilpolygone aufgeteilt wurde. In verschiedenen Werken [6, 15, 17, 19,21,31] werde Möglichkeiten einer solchen Aufteilung vorgestellt. Ein Vorgehen wäre zum Beispiel eine Triangulation eines Polygons zu erzeugen. In diesem Fall würden jedoch eine hohe Anzahl von Teilpolygonen entstehen. Um Teilpolygone zusammenzufassen, können nacheinander alle Kanten der Triangulation entfernt werden, solange das dadurch entstehende Teilpolygon weiterhin konvex ist.

Hieraus wird ersichtlich, dass es verschiedene Möglichkeiten gibt ein Polygon in konvexe Teilpolygone aufzuteilen. Zum Schluss dieser Arbeit wird besprochen, welche Auswirkungen diese Vorverarbeitung auf den Verlauf des vorgestellten Algorithmus haben kann.

## Ordnung der Teilpolygone

Es kann nun davon ausgegangen werden, dass das Polygon P bereits in konvexe Teilpolygone CP1, CP2, … CPp zerlegt wurde. Die Indizes der Teilpolygone können willkürlich geordnet sein haben und treffen keine Aussage über die tatsächliche Anordnung im Polygon P. Aus diesem Grund werden die Teilpolygone zuerst neu geordnet, um eine spätere Aufteilung zu vereinfachen. Dazu wird ein Verbindungsgraph G erzeugt und anhand dessen mittels einer Tiefensuche eine Ordnung erzeugt. Für jedes Teilpolygon CPi wird ein Knoten Ni in G eingefügt und für jeden Nachbarn CPk (i ungleich k) des Teilpolygons CPi wird eine Kante zum jeweils korrespondierenden Knoten Nk eingefügt.

Wir definieren Knoten Ni in g als Blatt, wenn Ni entweder nur einen Nachbar hat oder alle Nachbarn von Ni als markiert gelten.

Die Prozedur *OrderPieces* beschreibt nun die Neuordnung der Teilpolygonen. *OrderPieces* wird nun mit einem Knoten Ni von G initialisiert. Zuerst wird geprüft, ob Ni bereits markiert wurde. Ist dies der Fall, kann der Aufruf zurückkehren. Falls Ni noch nicht markiert wurde, wird geprüft, ob Ni ein Blatt ist. Falls Ni kein Blatt ist, dann wird der Knoten markiert und für alle Nachbarn Nk von Ni rekursiv *OrderPieces* aufgerufen. Nach dem Rücksprung der Aufrufe, wird CPi ausgegeben. Falls Ni ein Blatt ist, dann wird Ni markiert und CPi ausgegeben. Anschließend wird für alle Nachbarn Nk von Ni rekursiv *OrderPieces* aufgerufen.

Die neue Ordnung über die CPi ist nun die Reihenfolge, in der die Teilpolygone ausgegeben wurden. Diese neu entstandene Ordnung wird in den nächsten Abschnitten genutzt und die Teilpolygone anhand von PlL, NextNeighbor(CP) und PredPoly(CP), wie sie in Kapitel 6 beschrieben wurden, zu klassifizieren.

## Aufteilung eines nicht einfachen, nicht konvexen Polygons

Für die Aufteilung wird jedes Teilpolygon CP1, CP2, … CPp betrachtet. Konkret wird das Polygon PredPoly(CPi) so aufgeteilt, dass ein Teilstück einem Standort in CPi zugeordnet wird und der Rest dem Polygon PredPoly(CPi+k) angehangen wird. Diese Aufteilung wird durch die, sich gegenseitig rekursiv aufrufenden, Prozeduren *NonconvexDivide* und *DetachAndAssign* erreicht. Ersteres erzeugt ein Liniensegment, dass PredPoly(CPi) in zwei Teile aufteilt und letzteres ordnet die Teile entweder einem Standort zu oder teilt die zwei neuen Teilpolygone wiederum rekursiv auf.

Zuerst wird die Prozedur *NonconvexDivide* beschrieben, die die Teilpolygone in zwei Teile aufteilt. Listing XX beschreibt diese Prozedur.

Als Eingabe dient ein konvexes Teilpolygon und deren Informationen, also die Liste W(CPi) (= wk, k = 1, … m) mit allen geometrischen Punkten und Steiner-Punkten und die Liste S(CPi ) mit den Standorten des Teilpolygons und den jeweiligen benötigten Flächen. Anders als *ConvexDivide* aus Kapitel 7 ist die Reihenfolge der Knoten in W(CPi) relevant für die Bearbeitung. Die Kante, die durch die Knoten (wm, w1) erzeugt wird, sei nun die Kante zu NextNeighbor(CPi). Hat CPi keinen nächsten Nachbarn, muss wm gleich einem Standort sein.

Die Prozedur lässt nun erneut eine Kante L gegen den Uhrzeigersinn durch das Polygon CPi wandern, wie es auch schon in *ConvexDivide* der Fall war. L wird durch (Ls, Le) = (w1, Si) initialisiert, wobei Si der erste Standort aus S(CPi) ist.

Nun können zwei Fälle eintreten, in denen die Schleife stoppt.

1. Die Fläche rechts der Linie größer oder gleich der benötigten Fläche der Standorte, die sich in diesem Gebiet befinden. Es gilt:  
   Area(PrL) >= AreaRequired(S(CPi))
2. Das Ende des Polygons wird erreicht, also Le = wm.

Durch die Bearbeitung von vorherigen Teilpolygonen, kann es sein, dass nicht zugewiesene Teile dieser Polygone in die Aufteilung von Polygon CPi miteingezogen werde müssen. Außerdem kann nun der Fall eintreten, dass Die Fläche des Teilpolygons kleiner ist, als AreaRequired(S(CPi)). Aus diesem Grund, müssen die oberen beiden Fälle noch feingranularer aufgeteilt werden.

**Fall 1:** Wie auch in der Prozedur *ConvexDivide* wird ein Ende der Linie L bewegt um die Fläche Area(PrL) zu verkleinern. Hierbei unterscheiden wir zwei Fälle. Falls Le = Si für irgendein i gilt, dann wird der Startpunkt Ls bewegt, sonst wird der Endpunkt bewegt.

Nun seien L1 und L2 zwei Liniensegmente, die einen gemeinsamen, festen Endpunkt haben. Dieser Endpunkt ist entweder Ls oder Le und jeweils das Gegenstück zum oben bestimmten Punkt. Es gilt nun Area(PrL1) < AreaRequired( PrL1) und Area(PrL2) > AreaRequired( PrL2). Die Linie L = (w1, wk) wird also zu L2 und daher gilt für Linie L1 = (w1, wk-1).

Dadurch entsteht ein Dreieck T = (t1, t2, t3), dass die Differenz zwischen CPrL1 und CPrL2 bildet. Außerdem sei (t1, t2) die Linie von CPi, die L1 und L2 verbindet. Der gemeinsame Endpunkt von L1 und L2 ist demnach t3.

Nun müssen CPrL1 mit Teilen des Dreiecks T und Teilen der Reste der Vorgängerpolygone verbunden werden, damit die Flächenanforderungen der Standorte in CPrL1 erfüllt ist. Dabei entstehen 3 Fälle:

* 1. Area(PrL1 + T) > AreaRequired(S(CPrL))  
     Die Flächenanforderung der Standorte kann bereits durch die Fläche rechts von L1 und T vollständig gedeckt werden
  2. Area(PrL1 + T) <= AreaRequired(S(CPrL)) und  
     Area(PrL1 + PredPoly(CP,(t1, t2)) < AreaRequired(S(CPrL)) (\*)  
     Die Flächenanforderung der Standorte kann genau oder nicht komplett durch die Fläche rechts von L1 und T komplett abgedeckt werden, jedoch nicht durch die Fläche rechts von L1 und die Reste der Vorgängerpolygone.
  3. Area(PrL1 + T) <= AreaRequired(S(CPrL)) und  
     Area(PrL1 + PredPoly(CP,(t1, t2)) >= AreaRequired(S(CPrL))  
     Die Flächenanforderung der Standorte kann genau oder nicht komplett durch die Fläche rechts von L1 und T komplett abgedeckt werden, jedoch durch die Fläche rechts von L1 und die Reste der Vorgängerpolygone.

Durch diese Fälle wird nun entschieden, welche Teilpolygone an die Prozedur *DetachAndAssign* weitergegeben werden, um Flächen gewissen Standorten zuzuordnen.

**Fall 1.1:** Da die Flächenanforderung bereits durch das Polygon PrL1 und einen Teil des Dreiecks gedeckt werden kann, reicht es aus mittels linearer Interpolation einen Punkt t zwischen t1 und t2 zu finden, sodass das für das Dreieck T‘ = (t1, t, t3) gilt:

Area(PrL1 + T‘ - PredPoly(CP,(t1, t)) = AreaRequired(S(CPrL))

Es müssen also keine Reste der Vorgängerpolygone genutzt werden, die links von L1 liegen. Durch diese Aufteilung entstehen die beiden Polygone (PrL1 + T‘ - PredPoly(CP, (t1, t))) und (PlL1 – T‘). Diese beiden Polygone werden nun der Prozedur *DetachAndAssign* übergeben und dort gegebenenfalls weiter rekursiv aufgeteilt.

**Fall 2.2:** Damit die Flächenanforderung erfüllt werden kann, müssen zusätzlich zu den Flächen rechts von PrL1 und Teilen des Dreiecks T noch die Reste der Vorgängerpolygone mit einbezogen werden. Erneut wird durch lineare Interpolation der Punkt t gefunden und wie oben das Dreieck T‘ gebildet, sodass die Flächenanforderung erfüllt ist. Dreieck T‘ kann durch die strikte Ungleichung (\*) nicht kollabieren. Somit entstehen die beiden Polygone (PrL1 + T‘) und (PlL1 – T‘ – PredPoly(CP, (t1, t))), die ebenfalls der Prozedur *DetachAndAssign* übergeben werden und dort weiter rekursiv aufgeteilt werden.

**Fall 3.3:** Die Restfläche der Vorgängerpolygone ist größer als die Flächenanforderung der Standorte. In diesem Fall muss die Fläche der Vorgängerpolygone wiederrum aufgeteilt werden. Ein Teil wird dann S(CPrL1) zugeordnet, der andere Teil bleibt unbehandelt. Um dies zu erreichen, wird auf der Kante von CP und zwischen t1 und t2 willkürlich ein Pseudostandort PS hinzugefügt. Dadurch entsteht das Dreieck T‘ = (t1, PS, t3). Es gilt:

AreaRequired(PS) = AreaRequired(S(CPrL)) – Area(PrL1 + T‘)

PS bekommt also die fehlende Flächenanforderung zugewiesen. Dadurch kann PredPoly(CP, (t1, PS)) ebenfalls durch *NonconvexDivide* aufgeteilt und ein Teilpolygon PS zugewiesen werden. Das zugewiesene Teilpolygon kann dann dem Polygon (PrL1 + T‘) hinzugefügt werden. Dieses Polygon und das Polygon (PlL1 – T‘) werden dann an *DetachAndAssign* übergeben und weiter aufgeteilt.

Weiterhin betrachtet werden, muss der Fall, dass das Liniensegment L das Polygon einmal komplett durchlaufen hat.

**Fall 2:** Dieser Fall tritt ein, wenn ein Teilpolygon und die Reste der Vorgängerpolygone weniger Fläche enthält, als die Standorte beanspruchen. In diesem Fall ist CPi Flächen-unvollständig und es muss mindestens ein Pseudostandort erzeugt werden, für mindestens einen Standort aus S(CPrL). Dazu wirkt ein Punkt t auf der Kante (wm, w1), also der Kante zu NextNeighbor(CPi), erzeugt. Nun sei L = (t, Si) der erste Standort aus S(CPi), wenn man CP gegen den Uhrzeigersinn von w1 aus durchläuft. Nun wird W(PrL) so geordnet, dass t = w1 ist. Dann werden PrL und PlL an *DetachAndAssign* übergeben und aufgeteilt. Hierbei entsteht auf der Seite (w1, w2) entweder ein Pseudostandort oder ein Teil von PrL wird dem Standort Si zugeordnet. Wenn ein Pseudostandort entsteht, dann wird diesem die Flächenanforderung von Si abzüglich der Fläche von PrL zugeordnet und das Polygon PrL von CPi entfernt. Mit den restlichen Standorten von CPi wird dasselbe Verfahren angewandt.

Die Pseudostandorte werden nun bei der Aufteilung von NextNeighbor(CPi) behandelt. Wenn den Pseudostandorten bei dieser Bearbeitung ein Polygon zugeteilt wird, dann wird dieses Polygon auf die korrespondierenden Standorte übertragen.

Durch *NonconvexDivide* wird ein q-Standort Polygon entweder in ein q1-Standort Polygon und ein q2-Standort Polygon aufgeteilt mit q1, q2 > 0 und q1 + q2 = q oder es wird ein 1-Standort Polygon abgetrennt und einem Standort zugeteilt und es bleibt ein q‘-Standort Polygon mit kleinerer Fläche und mit q = q‘ + 1 über.

Die Prozedur *DetachAndAssign* teilt ein Teilpolygon einem Standort zu oder teilt ein Teilpolygon erneut mittels *NonconvexDivide* auf. *DetachAndAssign* ist durch Listing XX beschrieben.

*DetachAndAssign* wird nun das Polygon Poly(CP) übergeben, das, wie oben beschrieben, mittels *NonconvexDivide* ermittelt wurde. Nun können 3 Fälle auftreten.

1. PredPoly(CP) ist Flächen-vollständig
2. PredPoly(CP) ist Flächen-unvollständig
3. PredPoly(CP) ist Standort-unvollständig

Im ersten Fall kann es nun sein, dass PredPoly(CPi) lediglich einen Standort besitzt. Dann kann PredPoly(CPi) komplett diesem Standort zugeteilt (*Assign*) werden und vom Polygon Poly(CP) getrennt (*Detach*) werden.

Falls PredPoly(CP) mehrere Standorte enthält, dann wird PredPoly(CP) von Poly(CP) getrennt und PredPoly(CP) wird rekursiv mittels *NonconvexDivide* aufgeteilt.

Im zweiten Fall treten ebenfalls die beiden Unterfälle auf. Falls PredPoly(CP) lediglich einen Standort Si hat, dann kann PredPoly(CP) dem Standort zugeteilt werden und danach von Poly(CP) getrennt werden. Da PredPoly(CP) Flächen-unvollständig war, muss nun ein Pseudostandort auf der Kante zum NextNeighbor(CP) erzeugt werden, der die restliche Flächenanforderung von Si enthält. Wenn Flächen später dem Pseudostandort zugeteilt werden, dann werden diese stattdessen Si zugeteilt.

Falls PredPoly(CP) mehrere Standorte hat, dann wird PredPoly(CP) zuerst neu geordnet, sodass (wm, w1) die Kante zu NextNeighbor(CP) ist und dann rekursiv mittels *NonconvexDivide* aufgeteilt, da nicht klar ist, welcher Standort für die Flächen-unvollständigkeit verantwortlich ist.

Im dritten Fall hat PredPoly(CP) mehr Fläche, als die Standorte von CP benötigen. In diesem Fall wird PredPoly(CP) ebenfalls neu geordnet, sodass (wm, w1) die Kante zu NextNeighbor(CP) ist und dann rekursiv mittels *NonconvexDivide* aufgeteilt.

## Behandlung innen liegender Standorte

Falls Standorte innerhalb von P liegen, wird P so in konvexe Teilpolygone aufgeteilt, dass die Standorte auf Kanten der konvexen Aufteilung liegen. Ist dies nicht direkt möglich, können für die Standorte auch weitere Kanten eingefügt werden und die Aufteilung in konvexe Teilpolygone wird etwas feingranularer. Für den korrekten Ablauf des Algorithmus spielt diese Art der Einteilung keine Rolle.

## Beispiel

Das Beispiel aus Abbildung XX ist aus dem Artikel von XX übernommen.

Abbildung 17 zeigt verschiedene Stadien der gleichmäßigen Aufteilung eines nicht konvexen Polygons mit 12 Ecken und sieben Standorten. Polygon *(a)* zeigt die initiale Aufteilung des Polygons in 5 konvexe Teilpolygone CP1, …, CP5. In den darauffolgenden Polygonen *(b) – (f)* werden die Teilpolygone, die bereits einem Standort zugeteilt sind dunkelblau markiert. Die Teilpolygone, die einem Standort zugeteilt sind, aber noch nicht ausreichend viel Fläche beinhalten, sind hellblau markiert.

In Polygon *(b)* wurde das Teilpolygon CP1 bearbeitet. Dabei wurde es in zwei Teilpolygone aufgeteilt. P3 wurde dem Standort S3 zugeordnet und P4 wurde dem Standort S4 zugeordnet. P4 erfüllt die Flächenanforderung von S4 nicht vollständig, weshalb ein Pseudostandort S‘4 an der Kante zu NextNeighbor(CP1) erzeugt wurde. Dieser Pseudostandort wird zu einem späteren Zeitpunkt bearbeitet.

Polygon *(c)* zeigt den Stand nach der Bearbeitung von Teilpolygon CP2. Hier tritt erneut der Fall auf, dass die Fläche, die dem Standort S2 zugeteilt wird, zu klein ist. Aus diesem Grund wird der Pseudostandort S‘2 an der Kante zu NextNeighbor(CP2) erzeugt.

Als nächstes wird in Polygon *(d)* der Stand nach der Bearbeitung von Teilpolygon CP3 gezeigt. Dort wurde das Teilpolygon P7 dem Standort S7 zugeteilt und das Teilpolygon P1 dem Standort S1. Der Rest von CP3 wurde mit P2 vereint und daher ebenfalls S2 zugewiesen. Da die Fläche von P2 weiterhin nicht groß genug ist um die Flächenanforderung von S2 zu genügen, wird ein neuer Pseudostandort S‘2 an der Kante zu NextNeighbor(CP3) erzeugt.

Polygon *(e)* zeigt Stand nach der Bearbeitung von CP4. Dort hat die Sweepline das Ende von CP4 erreicht, ohne an einer Stelle genug Fläche zu beinhalten für die enthaltenen Standorte. Aus diesem Grund wird P5 und der Pseudostandort S‘5 erzeugt und diese S5 zugeordnet. Anschließend wird derselbe Schritt für S6 wiederholt. Nach abziehen der Fläche von P5 und P6 von CP4 werden Teile von CP4 den Pseudostandorten S‘2 und S‘4 zugeordnet und mit den Teilpolygonen P2 und P4 vereint.

Im letzten Polygon (Polygon *(f)*) wurde CP5 und der verbliebene Teil von CP4 den Pseudostandorten S‘5 und S‘6 zugeordnet. Damit wurde das Polygon vollständig in gleichmäßige Teilpolygone P1, …, P7 aufgeteilt.