# Abstract

In dieser Arbeit wird ein konkretes Problem der Polygonzerlegung, das sog. „Problem der verankerten Flächenaufteilung“ (eng. anchored area partition problem“) vorgestellt, welches von Susan Hert und Vladimir Lumelsky in [XXX] veröffentlicht wurde. Die vorliegende Arbeit basiert maßgebend auf dieser Veröffentlichung und beschreibt, wie dieses Problem mittels sweepline- und divide-and-conquer-Techniken effizient gelöst werden kann. Die Lösung erfolgt zunächst für konvexe Polygone und wird anschließend auf nicht konvexe, nicht einfache Polygone erweitert.

# Stichwörter: Polygonzerlegung, Flächenaufteilung, Flächenerkundung, Roboterplanung

# Einleitung

~~Die Arbeit basiert auf der Veröffentlichung~~ *~~„Polygon Area Decomposition for Multiple-Robot Workspace Division“~~* ~~von Susan Hert und Vladimir Lumelsky [<Literaturverweis>]. In dieser Veröffentlichung wird ein konkretes Problem der Polygonzerlegung, das sogenannte „Problem der verankerten Flächenaufteilung“ (eng.~~ *~~„anchored area partition problem“~~*~~) beschrieben und mittels sweepline- und divide-and-conquer-Techniken gelöst. Die Lösung erfolgt zunächst für konvexe Polygone und wird anschließend auf nicht konvexe, nicht einfache Polygone erweitert.~~

Die Polygonzerlegung ist eines der zentralen Probleme in der algorithmischen Geometrie und hat viele Anwendungsfälle, wie beispielsweise in der Kartographie, Bildverarbeitung oder in der Computergrafik. In vielen Fällen wird die Polygonzerlegung benötigt, um aus einem beliebigen Polygon eine Menge aus Teilpolygonen mit bestimmten Eigenschaften zu berechnen. Als Beispiel einer vielfach verwendeten Polygonzerlegung kann die Triangulation genannt werden, bei welcher ein gegebenes Polygon in eine Menge von Dreiecken zerlegt wird. Für die so berechnete Menge von Dreiecken stehen dann effiziente Algorithmen zur Lösung von Problemen zur Verfügung. Anschließend können die Lösungen der Teilpolygone zu einer Lösung für das Ausgangspolygon zusammengefasst werden.

Bei dem hier vorgestellten Problem der verankerten Flächenaufteilung ist die Anforderung an die resultierenden Teilpolygone nicht durch eine bestimmte Geometrie (z. B. ein Dreieck), sondern durch die Lage und Fläche der Teilpolygone gegeben. Bezüglich der Lage besteht die Anforderung darin, dass ein gegebener Punkt (*Standort* genannt) auf dem resultierenden Polygon liegen muss. Jeder Standort weist als Eigenschaft eine Flächenanforderung auf, welche durch die Größe des Teilpolygons erfüllt werden soll. Die Flächenanforderung kann je Standort den gleichen Wert aufweisen, kann aber auch unter den Standorten variieren. Somit bezieht sich das hier beschriebene Problem / die Lösung sowohl für eine gleichmäßige, als auch eine ungleichmäßige Flächenaufteilung. Das beschriebene Problem ist unter anderem durch die Flächenerkundung von Robotern motiviert:

Auf einem Polygon werden n Roboter auf Standorten Si, i = 1,…,n positioniert, welche die Aufgabe erhalten, zusammen die gesamte Fläche des Polygons zu erkunden. Hierzu muss jede Position innerhalb des Polygons von einem der n Roboter abgefahren werden. Um die Arbeit unter den Robotern aufzuteilen, ist es sinnvoll, jedem Roboter einen Polygonteil zuzuweisen, der jeweils zu bearbeiten ist. Die Teilpolygone sollen sich nicht überlappen, um ein mehrfaches Überfahren zu vermeiden. Bei der Flächenaufteilung muss berücksichtigt werden, dass der Startpunkt eines jeden Roboters auf dem zugewiesenen Teilpolygon beziehungsweise in diesem liegt. Eine unterschiedliche Leistung der Roboter kann über die Flächenanforderung je Standort berücksichtigt werden.

Zur formalen Beschreibung des Problems sind als Eingangsdaten ein Polygon P sowie eine (nicht leere) Liste von Standorten S(P) gegeben. Für jeden der n Standorte Si ist der benötigte Flächenanteil ci mit 0 < ci < 1 gegeben, sodass gilt. Das Polygon P soll in n, nicht-überlappende Polygone zerlegt werden, sodass jeder Standort Si auf dem Polygon Pi mit Fläche ci \* Area(P) liegt. Aus der Fläche des Polygons P kann für jeden der n Standorte die benötigte Fläche mit ci \* Area(P) bestimmt werden.

In der Praxis werden Flächenaufteilungen beispielsweise bei der Einteilung von Zustellbezirken der Post oder Einsatzgebieten von Rettungskräften verwendet. Die zuvor beschriebene, konkrete Problembeschreibung der verankerten Flächenaufteilung kann in Bezug auf die Roboterplanung beispielsweise auf Saug- oder Mähroboter übertragen werden, welche inzwischen in einigen Haushalten zu finden sind.

Hinweis: Die im Folgenden verwendeten Abkürzungen werden in Kapitel XXX erläutert und bei ihrer ersten Verwendung nicht explizit beschrieben. Einzelne Codezeilen der Listings wurden ggü. der Veröffentlichung [XXX] zur besseren Lesbarkeit angepasst.

# Aufteilung eines einfachen, konvexen Polygons

## Grundidee

Bei der nachfolgend beschriebenen Lösung des Problems wird das konvexe Eingabepolygon CP mithilfe von Liniensegmenten schrittweise zerlegt. Jedes Liniensegment L ist hierbei vom Startpunkt Ls zum Endpunkt Le orientiert, wobei beide Punkte auf dem Rand von CP liegen. Wenn für Ls und Le eine Position gefunden wurde, erfolgt eine Zerlegung in zwei Teilpolygone. Die bei jeder Teilung entstehen Teilpolygone erhalten entsprechend ihrer Lage zum Liniensegment L die Bezeichnungen für das *rechts* und für das *links* des Liniensegments liegenden Polygons. Die Liniensegmente (bzw. Ls und Le) werden so positioniert, dass die Fläche von der benötigten Fläche der auf dem Rand von liegenden Standorte entspricht ( analog). Die Zerlegung wird für jedes Teilpolygon und rekursiv aufgerufen, bis nur noch 1-Standort Polygone vorliegen. 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.

|  |  |
| --- | --- |
| (a) | (b) |
| (c) | (d) |

Abbildung 1: Zerlegung eines konvexen Polygons CP in vier konvexe Polygone CP1,...,CP4

Abbildung XX zeigt ein Beispiel für eine Zerlegung eines 4-Standort-Polygons in 4 Teilpolygone mit jeweils einem Standort. In (a) wird mit einer Fläche von 3.4 FE abgetrennt und dem Standort S01 zugeordnet. Die in (b) entstehenden Teilpolygone und weisen eine Fläche von 49.3 FE beziehungsweise 14.8 FE auf. Letzteres wird dem Standort S04 zugeordnet und ist als 1-Standort-Polygon fertig bearbeitet. wird in (c) erneut aufgeteilt, sodass die Flächenanforderung von S02 und S03 jeweils erfüllt werden. (d) zeigt die resultierende Aufteilung mit den 4 Teilpolygonen.

## Aufteilung eines einfachen, konvexen Polygons

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

1 // Input: Convex polygon CP

2 Function ConvexDivide(CP)

3 if Length(S(CP)) == 1 then return CP

4 // Here, the postion of L has to be calculated, see chapter XX

5 PrL, PlL = Cut(CP, L) // CP is cut into two pieces PrL and PlL

6 ConvexDivide(PrL) // recursive PrL

7 ConvexDivide(PlL) // recursive PlL

8 End ConvexDivide()[[1]](#footnote-1)

Bei jedem Aufruf von ConvexDivide(CP) wird zunächst geprüft, ob das übergebene Polygon nur noch einen Standort enthält. Falls ja, ist der Zielzustand für das übergebene Polygon erreicht und es ist keine weitere Flächenaufteilung erforderlich. Falls das Polygon mehrere Standorte enthält, erfolgt eine weitere Aufteilung des Polygons in zwei Teil-Polygone und , welche dann rekursiv mit ConvexDivide aufgerufen werden.

## Positionierung der Schnittlinie

Aus vorangegangenem Kapitel bleibt noch offen, wie genau die Aufteilung eines konvexen Polygons CP in die Polygone und erfolgt, sodass anschließend Area() = AreaRequired(S1,...,Si) und Area() = AreaRequired(Si+1,...,Sn) gilt. Konkret ist zu klären, wie Anfangs- und Endpunkt der Schnittlinien positioniert werden (siehe Listing XX, Zeile 4).

Initialisierung von Ls und Le beim 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 (und kein Standort) ist. Es gilt daher 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 dem der erste Standort liegt. Da die Standorte nach ihrem Vorkommen auf dem Weg von v1 nach vn geordnet sind, ist bei einem konvexen Polygon 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() = S2,…,Sn gelten, wobei S1 in einer Ecke von liegen würde. Je nach Fläche von und *AreaRequired*(S1) werden folgende 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 als Drehpunkt fungiert und Ls inkrementell gegen den Uhrzeigersinn entlang des Polygons verschoben wird, bis Area() = AreaRequired(S1) gilt. Zur Verdeutlichung dieser Vorgehensweise sollen folgende Punkte nochmals herausgestellt werden:

* Durch die Initialisierung kann auf dem Weg von w1 zu wk kein weiterer Standort liegen, d. h. *AreaRequired*(S()) ist konstant.
* Area() wird mit Verschiebung von Ls stetig kleiner. Bei Ls = S1 gilt Area() = 0.
* Le ist fest, d. h. S1 ist stets Teil von .

Wenn die Bedingung Area() = AreaRequired(S1) eintritt, erfolgt eine Polygonzerlegung. Für erfolgt keine weitere Zerlegung beim Aufruf von ConvexDivide (siehe Listing XX, Zeile 3), da nur S1 auf dessen Rand liegt. Falls auf dem Rand von mehr als ein Standort verbleibt, erfolgt eine erneute Zerlegung beim Aufruf von ConvexDivide() (siehe Listing XX, Zeilen 4 und 5).

**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 fungiert Ls als Drehpunkt und Le wird auf den nächsten in W vorkommenden Polygonpunkt oder Standort (wk+1) gesetzt. Die Anforderung wird erneut geprüft. In diesem Schritt wird Le von Polygonpunkt zu Polygonpunkt verschoben. Eine inkrementelle Verschiebung entlang des Polygons erfolgt dann unter Fall 2.1 beziehungsweise 2.2.

Hierbei kann es nun vorkommen, dass Le auf die Koordinaten eines Punktes wj in W gesetzt wird, welcher ein Standort S1 ist. Dieser Standort wird dann beim nächsten Vorrücken (also bei wj+1) zur benötigten Fläche von hinzugenommen. Bei Fall 2 kann AreaRequired() demnach ansteigen, sodass ein Vorücken von Le zwar zu einer größeren Fläche von , nicht aber unbedingt zu einem günstigeren Verhältnis aus Area() / AreaRequired(S()) führt.

Le wird so oft verschoben, bis eine der folgenden Bedingungen eintritt:

* Area() > AreaRequired(S())
* Le = Sn

Je nachdem, wie weit Le vorrückt und wie die Fläche von PrL zur Flächenanforderung von S(PrL) ist, werden nun weiter zwei Fälle unterschieden:

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

Hinweis: Angenommen die Ausgangsposition von Le ist wj, dann muss es zwischen wj und wj-1 eine Position geben, bei der Area() = AreaRequired(S()) gilt, da beim Vorrücken Area() bei wj-1 zu klein und bei wj zu groß war. Dieser Zwischenpunkt wird durch Interpolation gefunden.

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

Diese Vorgehensweise entspricht im Wesentlichen Fall 1, wobei Ls (initialisiert mit w1) nun nicht im Uhrzeigersinn *zum ersten* Standort S1, sondern gegen den Uhrzeigersinn *zum letzten* Standort Sn bewegt wird. Vergleiche auch Abbildung XX, Fall (c) und (i).

|  |  |  |  |
| --- | --- | --- | --- |
| Fall | Initialisierung  Ls = w1, Le = s1 | Fall 1 und 2.1:  Area() > AreaRequired(S())  Fall 2.2: wk = sn (2.2) | Ergebnis  Area() = AreaRequired(S()) |
| 1 | (a) | (b) | (c) |
| 2.1 | (d) | (e) | (f) |
| 2.2 | (g) | (h) | (i) |

Abbildung xxx: Fall 1, 2.1 und 2.2 inklusive der jeweiligen Zwischenschritte. Die unterschiedlichen Fälle werden durch angepasste Flächenanforderungen von S01/S02 hervorgerufen (Fall 1: 0.20/0.80, Fall 2.1: 0.70/0.30, Fall 2.2: 0.95/0.05). Die Serie (a) - (c) bezieht sich hierbei auf Fall 1. Nach der Initialisierung (a) liegt bereits Area(PrL) > AreaRequired(S(PrL)) vor, sodass Le nicht gegen den Uhrzeigersinn bewegt werden muss und fest bleibt, siehe (b). In (c) wird Ls dann inkrementell gegen den Uhrzeigersinn bewegt, bis Area(PrL) = AreaRequired(S(PrL)) gilt. Fall 2.1 wird in Serie (d) - (f) dargestellt, bei welchem nach der Initialisierung (d) zunächst Area(PrL) < AreaRequired(S(PrL)) gilt. Nach einer schrittweisen Verschiebung von Le gegen den Uhrzeigersinn (e) folgt letztendlich eine Interpolation in (f), sodass Area(PrL) = AreaRequired(S(PrL)) gilt. Die Serie (g) - (i) bezieht sich auf Fall 2.2, wobei Le in (h) zuerst bis zum letzten Standort wandert, bevor anschließend eine inkrementelle Verschiebung von Ls in (i) erfolgt, sodass Area(PrL) = AreaRequired(S(PrL)) gilt. Hierbei lässt sich gut die Analogie zwischen Fall (c) und (i) erkennen.

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

1 // Input: Convex polygon CP

2 Function ConvexDivide(CP)

3 if Length(S(CP)) == 1 then return CP

4 Ls = W(1), Le = W(k) // k = index of first Site in W

5 PrL, PlL = Cut(CP, L) // partitioning, returns PrL and PlL

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

7 if W(k-1) != S1 and W(k-1) in S then

8 S(PrL) = S(PrL) + W(k-1) // add previous Site to S(PrL)

9 k += 1

10 Le = W(k) // move Le to next point in W

11 PrL, PlL = Cut(CP, L)

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

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

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

15 if Le != Sn then

16 interpolate Le that Area(PrL) == AreaRequired(S(PrL))

17 else if Le == Sn then

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

19 PrL, PlL = Cut(CP, L) // CP is cut into two pieces PrL and PlL

20 ConvexDivide(PrL) // recursive PrL

21 ConvexDivide(PlL // recursive PlL

22 End ConvexDivide()

## Beispiel

|  |  |  |
| --- | --- | --- |
| (a) | (b) | (c) |
| (d) | (e) | (f) |
| (g) | (h) | (i) |

Abbildung xxx: Beispiel einer gleichmäßigen Aufteilung eines konvexen Polygons mit acht Standorten und einem jeweils benötigen Flächenanteil von 12,5 %. (a) zeigt hierbei das Ausgangspolygon CP mit dem ersten Polygonpunkt bei Koordinate (8, 9) und den im Gegenuhrzeigersinn geordneten Standorten S01 bis S08. In (b) liegt zunächst Fall 2.1 vor, bei welchem Le bis V04 bewegt wird, sodass Area(PrL) > AreaRequired(S(PrL)) gilt. S(PrL) enthält in diesem Zustand die Standorte S01 bis S03. Für Le wird anschließend eine Interpolation zwischen V04 und V03 durchgeführt, sodass Area(PrL) = AreaRequired(S(PrL)) gilt und es wird ein Schnitt durchgeführt. Für das rechts der Schnittlinie entstandene Polygon wird in (c) erneut ConvexDivide aufgerufen. Hierbei tritt Fall 2.2 ein. Le erreicht bei der Verschiebung S03, welcher letzter Standort in S() ist. Ls wird anschließend im Uhrzeigersinn inkrementell bewegt, bis Area(PrL) = AreaRequired(S(PrL)) gilt. Erneut muss das rechts der Schnittlinie entstandene Polygon aufgeteilt werden, siehe (d). Le wandert wieder bis zum letzten Standort in S(), sodass erneut Fall 2.2 vorliegt. Dass Area(PrL) bei Le = Sn eine Fläche von 0 aufweist, stellt für den Algorithmus kein Problem dar. In (e) wird nun das in (b) links der Schnittlinie entstandene Polygon weiter aufgeteilt. Es liegt Fall 1 vor, da nach der Initialisierung mit Le = S1 bereits Area(PrL) > AreaRequired(S(PrL)) gilt. Ls wird anschließend gegen den Uhrzeigersinn verschoben, bis Area(PrL) = AreaRequired(S(PrL)) vorliegt. In den nächsten Schritten (f)-(h) gilt analog Fall 1. Speziell in (f) soll darauf hingewiesen werden, dass das inkrementelle Verschieben des Punktes Ls auch „um die Ecke“ möglich sein muss. (i) zeigt die resultierende Flächenaufteilung aus den Schritten (b) - (h) mit Teilflächen zu je 12.5 %.

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

Es wurde gezeigt, dass ein einfaches, konvexes Polygon rekursiv in n 1-Standort-Polygone aufgeteilt werden kann. Dieses Kapitel dient dazu einen verallgemeinerten Algorithmus zu skizzieren, damit auch für nicht einfache, nicht konvexe Polygone (Abbildung XX) das Problem der verankerten Flächenaufteilung gelöst werden kann.

Bevor dieser Algorithmus vorgestellt wird, soll die Beschreibung der dahinterliegenden Grundidee einen Überblick über die Vorgehensweise verschaffen. Danach werden die vorbereitenden Schritte vorgestellt und die Aufteilung des Polygons erläutert. Ein Beispiel dient anschließend zur Veranschaulichung des vorgestellten Algorithmus und zum Schluss des Kapitels wird der Sonderfall geschildert, dass Standorte im Inneren des Polygons liegen.

## Grundidee

In Kapitel XX wurde bereits erläutert, wie ein einfaches, konvexes Polygon aufgeteilt werden kann. Dieses Vorgehen kann auch bei der Aufteilung nicht konvexer 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,…,CPp zerlegt wurde. Im ersten Schritt werden die Teilpolygone mithilfe 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 XX gezeigt wurde. Allerdings können nun Sonderfälle auftreten, die bei der Zerlegung eines einfachen, konvexen Polygons nicht vorkommen konnten. Einerseits kann CPi weniger Fläche ausfüllen, 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 Flächen 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] werden Möglichkeiten einer solchen Aufteilung vorgestellt. Ein Vorgehen wäre zum Beispiel, eine Triangulation eines Polygons zu erzeugen. In diesem Fall würde jedoch eine hohe Anzahl von Teilpolygonen entstehen. Um Teilpolygone zusammenzufassen, können nacheinander 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 vorbereitenden Schritte auf den Verlauf des vorgestellten Algorithmus haben können.

1 // Input: Nj - Node of the connectivity Graph

2 Function OderPieces(Nj)

3 if Nj has not beeing marked then

4 if Nj is a leaf node then

5 Mark(Nj)

6 Output(CPj)

7 for each Nk in Neighbors(Nj)

8 OrderPieces(Nk)

9 else

10 Mark(Nj)

11 for each Nk in Neighbors(Nj)

12 OrderPieces(Nk)

13 Output(CPj)

14 End OrderPieces(Nj)

## Ordnung der Teilpolygone

Es kann nun davon ausgegangen werden, dass das Polygon P bereits in konvexe Teilpolygone CP1,…,CPp zerlegt wurde. Die Teilpolygone können willkürlich geordnet sein und die Indizes treffen keine Aussage über die tatsächliche Anordnung im Polygon P. Aus diesem Grund werden die Teilpolygone zuerst neu geordnet, was für den anschließend dargestellten Algorithmus erforderlich ist. Dazu wird ein Verbindungsgraph G gebildet und anhand dessen mittels einer Tiefensuche eine Ordnung erzeugt. Jedes Teilpolygon CPi wird durch einen Knoten Ni in G abgebildet. Für jeden Nachbarn CPk (i ungleich k) des Teilpolygons CPi wird eine Kante zum jeweils korrespondierenden Knoten Nk eingefügt.

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

Die Prozedur *OrderPieces* beschreibt nun die Neuordnung der Teilpolygone. *OrderPieces* wird mit einem beliebigen Knoten Ni von G initial aufgerufen. 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 nach obiger Definition 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 aller Nachbarn von CPi 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 der CPi ist nun die Reihenfolge, in der die Teilpolygone ausgegeben wurden.

1 // Input: Convex polygon CP

2 Function NonConvexDivide(CP)

3 Ls = W(1), Le = W(k) // k = index of first Site CCW from w1 in W

4 PrL, PlL = Cut(CP, L) // partitioning, returns PrL and PlL

5 while Area(PrL) < AreaRequired(S(PrL)) and Le != wm do

6 if W(k-1) != S1 and W(k-1) in S then

7 S(CPrL) = S(CPrL) + W(k-1) // add previous Site to S(CPrL)

8 k += 1

9 Le = W(k) // move Le to next point in W

10 PrL, PlL = Cut(CP, L)

11 if Area(PrL) > AreaRequired(S(CPrL)) then

12 if Le == Si then

13 k1 = 1

14 while Area(PrL) > AreaRequired(S(PrL)) do

15 k1 = k1 + 1

16 Ls = w(k1)

17 L1 = (w(k1), Le)

18 T(t1,t2,t3) = (w(k1), w(k1-1), Le)

19 else

20 L1 = (Ls, w(k-1))

21 T(t1,t2,t3) = (w(k-1), w(k1), Ls)

22

23 if Area(PrL1 + T) > AreaRequired(S(CPrL)) then

24 interpolate point t on (t1,t2) until Area(PrL + T) == AreaRequired(S(CPrL))

25 T(t1,t2,t3) = (t1, t, t3)

26 DetachAndAssign(PrL1 + T)

27 DetachAndAssign(PlL1 – T – PredPoly(CP,(t1,t))

28 else if Area(PrL1 + PredPoly(CP,(t1,t2)) < AreaRequired(S(CPrL)) then

29 interpolate point t on (t1,t2) until Area(PrL + T) == AreaRequired(S(CPrL))

30 T(t1,t2,t3) = (t1, t, t3)

31 DetachAndAssign(PrL1 + T)

32 DetachAndAssign(PlL1 – T – PredPoly(CP,(t1,t))

33 else

34 PS = interiorPoint(t1,t2) //PS is new Pseudosite

35 T(t1,t2,t3) = (t1,PS,t3)

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

37 Order(W(PredPoly(CP,(t1,t2)) //such that w1 = PS if Le != Si and wm = PS if Le == Si

38 DetachAndAssign(PredPoly(CP,(t1,t2))

39 DetachAndAssign(PrL1 + T)

40 DetachAndAssign(PlL1 + T)

41 else

42 t = InteriorPoint(wm, w1)

43 L1 = (t,Si)

44 DetachAndAssign(PrL1)

45 DetachAndAssign(PlL1)

46 End NonConvexDivide()

## Aufteilung eines nicht einfachen, nicht konvexen Polygons

Für die Aufteilung wird jedes Teilpolygon CP1,…,CPp betrachtet. Konkret wird das Polygon PredPoly(CPi) so aufgeteilt, dass ein Teilstück einem Standort in CPi (falls vorhanden) zugeordnet wird und der Rest dem Polygon PredPoly(CPk) mit k > i angehangen wird. Diese Aufteilung wird durch die sich gegenseitig rekursiv aufrufenden Prozeduren *NonconvexDivide* und *DetachAndAssign* erreicht. Erstere erzeugt ein Liniensegment, welches PredPoly(CPi) in zwei Teile aufteilt und letztere ordnet die Teile entweder einem Standort zu oder teilt sie erneut 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, beschrieben durch die Liste W(CPi) (= wk, k = 1,…,m) mit allen Polygonpunkten inklusive Steiner-Punkten und die Liste S(CPi) mit den Standorten des Teilpolygons inklusive der jeweils benötigten Fläche. Anders als bei *ConvexDivide* aus Kapitel 7 ist für die Bearbeitung relevant, welcher Punkt w1 in W(CPi) ist. Die Kante, die durch die Polygonpunkte (wm, w1) erzeugt wird, sei nun die Kante zu NextNeighbor(CPi). Hat CPi keinen nächsten Nachbarn, muss wm gleich einem Standort sein.

Wie es auch schon bei *ConvexDivide* der Fall war, lässt die Prozedur erneut ein Liniensegment L gegen den Uhrzeigersinn durch das Polygon CPi wandern, wobei Ls als Drehpunkt dient. 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 ist größer oder gleich der benötigten Fläche der Standorte, die sich in diesem Gebiet befinden. Es gilt:  
   Area(PrL) >= AreaRequired(S(CPrL))
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 CPi miteinbezogen werden 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 entlang des Polygons bewegt, um die Fläche Area(PrL) zu verkleinern. Hierbei unterscheiden wir zwei Fälle. Falls Le = Si für einen beliebigen Wert für i gilt, dann wird der Startpunkt Ls gegen den Uhrzeigersinn bewegt, ansonsten wird der Endpunkt Le im Uhrzeigersinn bewegt.

Nun seien L1 und L2 zwei Liniensegmente, die einen gemeinsamen, festen Endpunkt haben. Dieser gemeinsame Endpunkt ist für beide Linien entweder Ls oder Le und damit das Gegenstück zum oben bestimmten Punkt, welcher entlang des Polygons bewegt wurde. Die Linien sind so positioniert, dass Area(PrL1) < AreaRequired(S(CPrL1)) und Area(PrL2) > AreaRequired(S(CPrL2)) gilt. Die Linie L2 wird demnach durch (w1, wk) und die Linie L1 durch (w1, wk-1) beschrieben.

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

Nun müssen CPrL1 mit Teilen des Dreiecks T und gegebenenfalls 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 durch die Fläche rechts von L1 und T vollständig gedeckt werden. Insbesondere wird kein Flächenanteil von PredPoly(CP, (t1, t2)) benötigt.
  2. Area(PrL1 + T) <= AreaRequired(S(CPrL)) und  
     Area(PrL1 + PredPoly(CP, (t1, t2)) < AreaRequired(S(CPrL)) (\*)  
     Die Flächen von PrL1 und T reichen zusammen nicht (<) oder exakt (=) aus, um die Flächenanforderung der Standorte von CPrL zu erfüllen (1. Bedingung). Weiterhin liegt der Fall vor, dass die Fläche von PrL1 in Kombination mit dem Vorgängerpolygon PredPoly(CP, (t1, t2)) kleiner als die geforderte Fläche ist (2. Bedingung).
  3. Area(PrL1 + T) <= AreaRequired(S(CPrL)) und  
     Area(PrL1 + PredPoly(CP, (t1, t2)) >= AreaRequired(S(CPrL))  
     Bedingung 1 ist analog zu Fall 1.2. Weiterhin liegt nun jedoch der Fall vor, dass die Fläche von PrL1 in Kombination mit dem Vorgängerpolygon PredPoly(CP, (t1, t2)) zur Erfüllung der Anforderung genügen (2. Bedingung).

Die je nach Fall entstehenden Polygone werden anschließend an die Prozedur *DetachAndAssign* übergeben und dort entweder Standorten zugewiesen oder durch einen rekursiven Aufruf von *NonConvexDivide* erneut aufgeteilt.

**Fall 1.1:** Die Flächenanforderung von S(CPrL) kann durch das Polygon PrL1 zusammen mit dem Dreieck T erfüllt werden. In diesem Fall reicht es aus, mittels linearer Interpolation einen Punkt t zwischen t1 und t2 zu finden, sodass für das Dreieck T’ = (t1, t, t3) gilt:

Area(PrL1 + T’ - PredPoly(CP, (t1, t)) = AreaRequired(S(CPrL))

Durch diese Aufteilung entstehen die beiden Polygone (PrL1 + T’ - PredPoly(CP, (t1, t))) und (PlL1 – T’), die der Prozedur *DetachAndAssign* übergeben werden.

**Fall 1.2:** Damit die Flächenanforderung erfüllt werden kann, wird zunächst das Vorgängerpolygon PredPoly(CP, (t1, t2)) hinzugenommen und dieses um die Fläche PrL1 und einen Teil des Dreiecks T erweitert. Erneut wird durch lineare Interpolation der Punkt t gefunden und wie oben das Dreieck T’ gebildet, sodass die Flächenanforderung erfüllt ist. Das Dreieck T’ kann durch die strikte Ungleichung (\*) nicht kollabieren. Somit entstehen die beiden Polygone (PrL1 + T’) und (PlL1 – T’ – PredPoly(CP, (t1, t))), die der Prozedur *DetachAndAssign* übergeben werden.

**Fall 1.3:** Die Fläche der Vorgängerpolygone ist größer als die Flächenanforderung der Standorte. In diesem Fall muss die Fläche der Vorgängerpolygone wiederrum aufgeteilt werden. Ein Teil wird zur Erfüllung der Flächenanforderung genutzt und S(CPrL1) zugeordnet. Der übrige Teil wird im Weiteren durch einen sogenannten *Pseudostandort* PS abgebildet. PS wird auf der Kante zwischen t1 und t2 willkürlich hinzugefügt, sodass das Dreieck T’ = (t1, PS, t3) entsteht. 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.

Weiterhin muss der Fall betrachtet werden, bei dem 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 enthalten, als die Standorte beanspruchen. In diesem Fall ist CPi Flächen-unvollständig und es muss für mindestens einen Standort aus S(CPrL) ein Pseudostandort erzeugt werden. Dazu wird ein Punkt t auf der Kante (wm, w1) erzeugt, also der Kante zu NextNeighbor(CPi). Nun sei L = (t, Si), wobei Si der erste Standort in S(CPi) gegen den Uhrzeigersinn von w1 aus ist. Nun wird W(PrL) so geordnet, dass t = w1 gilt. Anschließend werden PrL und PlL an *DetachAndAssign* übergeben. Hierbei entsteht entweder auf dem Liniensegment (w1, w2) 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 hierbei ein Polygon zugeteilt wird, dann wird dieses Polygon auf die korrespondierenden Standorte übertragen.

Zusammenfassend wir durch den Algorithmus von *NonconvexDivide* ein q-Standort-Polygon entweder in ein q1-Standort-Polygon und ein q2-Standort-Polygon mit q1, q2 > 0 und q1 + q2 = q aufgeteilt oder es wird ein 1-Standort Polygon abgetrennt und es bleibt ein q’-Standort Polygon mit q’ = q - 1 übrig.

1 // Input: Poly(CP) - Polygon rooted at convex piece CP

2 Function DetachAndAssign(Poly(CP))

3 if Length(S(CP)) == 0 then return

4 if PredPoly(CP) is AreaComplete then

5 if S(CP) == {Si} then //for some i

6 Assign PredPoly(CP) to Si

7 Detach PredPoly(CP) from Poly(CP)

8 else

9 Detach PredPoly(CP) from Poly(CP)

10 Order(W(CP)) //such that wm = Si for some i

11 NonConvexeDivide(CP)

12 else if PredPoly(CP) is areaIncomplete then

13 if S(CP) == {Si} then //for some i

14 Assign PredPoly(CP) to Si

15 Detach PredPoly(CP) from Poly(CP)

16 PS = interiorPoint(w(j),w(k)) //with (w(j),w(k)) is edge to NextNeighbor(CP)

17 else

18 Order(W(CP)) //such that edge (w(m),w(1)) is edge to NextNeighbor(CP)

19 NonConvexeDivide(CP)

20 else

21 Order(W(CP)) //such that edge (w(m),w(1)) is edge to NextNeighbor(CP)

22 NonConvexeDivide(CP)

23 End DetachAndAssign()

## Der Algorithmus DetachAndAssign

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

Beim Aufruf von DetachAndAssign(Poly(CP)) können 3 Fälle auftreten. Das Polygon

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

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

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

Im zweiten Fall treten die gleichen zwei Unterfälle auf. Falls PredPoly(CP) nur einen Standort Si hat, kann PredPoly(CP) von Poly(CP) getrennt und dem Standort zugeteilt werden. Da PredPoly(CP) Flächen-unvollständig ist, muss nun ein Pseudostandort auf der Kante zu NextNeighbor(CP) erzeugt werden, der die restliche Flächenanforderung von Si enthält. Flächen, die im weiteren Verlauf dem Pseudostandort zugeteilt werden, werden so mittelbar dem Standort Si zugeteilt.

Falls PredPoly(CP) mehrere Standorte hat, dann wird PredPoly(CP) zunächst neu geordnet, sodass (wm, w1) die Kante zu NextNeighbor(CP) ist. Anschließend erfolgt wiederum ein rekursiver Aufruf von *NonconvexDivide*, da nicht bekannt ist, durch welchen Standort die Flächen-Unvollständigkeit resultiert.

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. Mit diesem Polygon erfolgt ein Aufruf von *NonconvexDivide*.

## Behandlung innen liegender Standorte

Für innen liegende Standorte muss die in Kapitel XX beschriebene Zerlegung in konvexe Teilpolygone so erfolgen, dass jeder Standort anschließend auf einer Kante liegt. Ist dies nicht direkt möglich, können für die Standorte auch weitere Kanten eingefügt werden und die Aufteilung in konvexe Teilpolygone wird etwas detaillierter. Die neuen Kanten laufen durch die innenliegenden Standorte. Für den korrekten Ablauf des Algorithmus spielt diese Art der Zerlegung keine Rolle.

## Beispiel

Das Beispiel aus Abbildung XX ist aus dem Artikel [Quelle], Abbildung XX übernommen.

Abbildung XX zeigt verschiedene Stadien der gleichmäßigen Aufteilung eines nicht konvexen Polygons mit 12 Ecken und sieben Standorten. *(a)* zeigt die initiale Aufteilung des Polygons in 5 konvexe Teilpolygone CP1, …,CP5. In *(b) – (f)* werden die Teilpolygone, die bereits einem Standort zugeteilt sind, dunkelblau markiert. Die Teilpolygone, die bereits einem Standort zugeteilt, aber noch Flächen-unvollständig sind, werden hellblau markiert.

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

*(c)* zeigt den Zustand 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.

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

*(e)* zeigt den Zustand nach der Bearbeitung von CP4. Dort hat das Liniensegment das Ende von CP4 erreicht, ohne dass zuvor die Flächenanforderung der Standorte erfüllt werden konnte. 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 Flächen 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 Schritt *(f)* wird CP5 und der verbliebene Teil von CP4 den Pseudostandorten S’5 und S’6 zugeordnet. Die Abbildung zeigt das in gleichmäßige Teilpolygone P1,…,P7 zerlegte Polygon.

# Komplexitätsanalyse

**Konvexes Polygon:**

Der Algorithmus *ConvexDivide* benötigt lineare Zeit bezogen auf die Anzahl der Elemente der Liste W(P), um einen einzelnen Schnitt durchzuführen. Der dabei erforderliche Aufbau der Polygone beziehungsweise sowie die Ermittlung der Fläche ist in konstanter Zeit möglich. Das Finden der Punkte, bei denen Area() = AreaRequired(S()) gilt, kann ebenso in konstanter Zeit erfolgen. Der Algorithmus ConvexDivide benötigt daher O(n+v) Zeit (n = Anzahl der Standorte, v = Anzahl an Polygonpunkten).

Im ungünstigste Fall trennt *ConvexDivide* von einem konvexen Polygon mit q Standorten nur ein Dreieck (v = 3, n = 1) ab. Neben dem Dreieck verbleibt dann ein konvexes Polygon mit q-1 Standorten und v+1 Polygonpunkten. Für dieses Polygon gilt wiederrum selbes. Um eine gesamte Flächenzerlegung eines konvexen Polygons zu berechnen, wird O((n-1)(n+v)) Zeit benötigt.

Hierbei sei angemerkt, dass dieser Algorithmus stets terminiert, da die Anzahl an Standorten konstant ist und die Anzahl der Teil-Polygone je Schnitt um 1 erhöht wird. Nach n-1 Schnitten entspricht die Anzahl der Teil-Polygone der Anzahl der Standorte.

**Nicht konvexes Polygon:**

Der Algorithmus *OrderPieces* besucht jeden Knoten im Nachbarschaftsgraphen maximal zwei mal, sodass die Ordnung in linearer Zeit O(p) bezogen auf die Anzahl p der konvexen Teile erfolgen kann.

Der Algorithmus *NonconvexDivide* benötigt O(pn2 + nv) Zeit, um alle p konvexen Teile unter den n Standorten aufzuteilen. Im ungünstigsten Fall wird jedes der p konvexen Teile in n Polygone zerlegt, wobei jedes der Polygone einen Teil eines Standorts abbildet.

Der Algorithmus *DetachAndAssign* wird getrennt für den Teil Detach beziehungsweise Assign betrachtet.

Um ein Polygon nach einem Schnitt zu lösen (Detach), müssen alle Zeiger auf Nachbar-Polygone aktualisiert werden. Dieser Vorgang ist in Zeit O(vj) für ein konvexes Polygon mit vj Polygonpunkten möglich. Der ungünstigste Fall beziehungsweise die maximale Anzahl an zu aktualisierenden Zeigern besteht dann, wenn je Zerlegung ein Dreieck (v=3) und ein Polygon mit vj+1 abgetrennt werden. Für alle Teilpolygone ergibt sich dann eine maximale Zeit von O(v + pn2).

Der Algorithmus Assign übernimmt die Zuweisung von Flächen zu Standorten, wobei eine Fläche aus einem Set von konvexen Teilen besteht. Die Vereinigung der konvexen Teile zu einem Polygon benötigt lineare Zeit bezogen auf die Anzahl der Polygonpunkte aller Teile, also maximal O(pn + v). Die Vereinigung aller n-Standort-Polygone benötigt daher O(pn2 + vn) Zeit.

Zusammenfassend ergibt sich aus den Laufzeiten für OrderPieces, NonconvexDivide und DetachAndAssign eine Gesamtlaufzeit von O(pn2 + vn).

# Schluss

In dieser Arbeit wurde das „Problem der verankerten Flächenaufteilung“ vorgestellt und anhand von Beispielen für konvexe und nicht konvexe Polygone erläutert. Sowohl der beschriebene Algorithmus für ein konvexes Polygon, als auch die verallgemeinerte Version für ein nicht-konvexes, nicht einfaches Polygon liefern für ein gegebenes Problem je Ausführung nur eine von vielen möglichen Lösungen. Spielraum für die Anpassung der Eingangsdaten liegt beispielsweise für den konvexen Fall in der Reihenfolge der Punkte (Welcher Punkt ist erster Punkt in W?) bzw. für den nicht-konvexen Fall in der Ordnung der Teilpolygone CPi (Mit welchem Knoten wird OrderPieces initial aufgerufen?). Je nachdem, wie die Punkte / Teilpolygone geordnet sind, ergeben sich daher unterschiedliche Lösungen. Weitergehende Untersuchungen könnten den Einfluss der Ordnung auf die Form der entstehenden Polygone betrachten. Beispielsweise wäre es für die Roboterplanung von Vorteil, wenn die resultierenden Flächen möglichst einfach und kompakt (im Sinne des kleinesten umschließenden Durchmessers) sind, da Roboter auf solchen Flächen typischerweise effizienter arbeiten.

1. Der in dieser Arbeit beinhaltete Pseudocode kann zum besseren Verständnis oder zur besseren Lesbarkeit von Inhalten aus [Literaturverweis] abweichen. [↑](#footnote-ref-1)