Karlsruher Institut für Technologie

Load-Balancing für parallele Delaunay-Triangulierung

Bachelorarbeit

von
Vincent Winkler

Angefertigt am

Institut für Theoretische Informatik
Algorithmik II

Gutachter: Prof. Dr. P. Sanders
Betreuer: D. Funke, M.Sc.

19. März 2018

Abstract

Calculating the Delaunay triangulation of large point clouds with over a million points is time-consuming. Point clouds of over a billion points may even require distributed memory. Previous work by Funke and Sanders [„Parallel d-D Delaunay Triangulations in Shared and Distributed Memory“] presents a novel parallel algorithm that calculates the Delaunay triangulation on shared or distributed memory. With more realistic inputs however, the algorithm has to reprocess about 5% of the triangulation due to its simple work division strategy.

This bachelor's thesis addresses the issue to decrease the overhead and further speed up the triangulation algorithm. An elaborate load balancing approach that partitions the input point cloud in an additional preprocessing step is provided. To find the partitioning, the Delaunay triangulation of a small input sample is partitioned by a graph partitioning algorithm and the resulting partitioning is extended to a partitioning of the entire input. Different extension strategies are developed and examined for the overall runtime, the reprocessing overhead and the balance of the partition sizes.

The number of multiply processed points can be reduced to less than 0.5%. Partition sizes that deviate less than 1% from each other can be achieved.

Inhaltsverzeichnis

Ich versichere wahrheitsgemäß, die Arbeit selbstständig angefertigt, alle benutzten Hilfsmittel vollständig und genau angegeben und alles kenntlich gemacht zu haben, was aus Arbeiten anderer unverändert oder mit Abänderungen entnommen wurde.

Karlsruhe, den _____ _____

Vincent Winkler

1 Einleitung

Eine oft angewendete Technik zur Repräsentation geometrischer Figuren sind Punktwolken. Punktwolken werden häufig von 3D-Scannern, die zur Abtastung von realen Gegenständen genutzt werden, produziert. Doch auch nicht-geometrische Daten können als Punktwolken, in denen ein Punkt ein Datum mit d verschiedenen Parametern darstellt, verstanden werden.

Um Punktwolken und das, was sie repräsentieren darzustellen oder um Punktwolken weiter zu verarbeiten, wird jedoch nicht selten eine Vernetzung bzw. Triangulierung der Punkte verwendet. Dabei werden die Punkte einer 2D-Punktwolke durch Dreiecke, einer 3D-Puntwolke durch Tetraeder und einer d-dimensionalen Punktwolke durch d-Simplizes vermascht. In der Computergrafik existieren vor allem Lösungen zum Zeichnen von Meshes, was eine Darstellung der Oberfläche gescannter realer Gegenstände durch eine triangulierte Punktwolke empfiehlt. Die Finite-Elemente-Methode ist eine häufig genutzte numerische Methode, um Probleme der Physik und der Ingenieurswissenschaften, wie die Simulation struktureller Verformungen eines verunfallten Fahrzeugs [1], zu lösen. Die Triangulierung einer Punktwolke kann eingesetzt werden, um das für die Finite-Elemente-Methode nötige Raster zu generieren [2] [3].

Ein spezielle Art der Triangulierung ist die Delaunay-Triangulierung. Sie empfiehlt sich vor allem dadurch, dass sie eine (fast) eindeutige Triangulierung der Punktwolke ist und seltener geometrisch ungünstige Simplizes produziert, die sehr geringe Innenwinkel haben und sehr schmal sind. In [4] wird eine Variante der Delaunay-Triangulierung bei der Planung von Wegen, die Hindernisse vermeiden, eingesetzt.

Die Berechnung der Delaunay-Triangulierung großer Punktwolken von mehreren Millionen Punkten ist jedoch zeitaufwändig. Um dem zu begegnen, existieren Algorithmen zur parallelen Berechnung der Delaunay-Triangulierung. Punktwolken von sogar über einer Milliarde Punkten können Systeme mit verteiltem Speicher erforderlich machen.

In ihrer Arbeit zu paralleler d-dimensionaler Delaunay-Triangulierung beschreiben Funke und Sanders [5] einen Teile-und-herrsche-Algorithmus zur Berechnung der Delaunay-Triangulierung einer d-dimensionalen Punktwolke. Je eine Variante für Systeme mit gemeinsam genutztem und verteiltem Speicher wird angegeben. Hierbei wird die Punktwolke in mehrere Teilmengen zerlegt und für jede Teilmenge rekursiv die Delaunay-Triangulierung bestimmt. Sobald die Punktwolke hinreichend klein ist, ist der Basisfall erreicht, in dem die Delaunay-Triangulierung durch einen sequentiellen Algorithmus bestimmt wird.

Nicht alle Simplizes der Triangulierungen der Teilmengen sind auch Simplizes der

Triangulierung der gesamten Punktwolke. Die Eckpunkte jener Simplizes, auf die das möglicherweise nicht zutrifft, werden in einer Punktmenge zusammengefasst, die „Rand“ (engl. border) genannt wird. Nachdem der Rand ermittelt wurde, wird auch für ihn die Delaunay-Triangulierung bestimmt, um anschließend den Rand und die Triangulierungen der Teilmengen zusammenzusetzen und die Delaunay-Triangulierung der gesamten Punktwolke zu erhalten.

Abschließend stellen Funke und Sanders in ihrer Veröffentlichung Arbeiten in Aussicht, die den Algorithmus aufgreifen und aufwändigere Load-Balancing-Strategien untersuchen. Load-Balancing durch gezieltes Zerlegen der Punktwolke hat großen Einflussauf den Aufwand, der zum Zusammensetzen der einzelnen Triangulierungen betrieben werden muss. Ziel ist, dass jeweils so wenige Punkte in den Rand fallen wie möglich, damit nur eine geringe Zahl an Punkten mehrmals sequentiell trianguliert werden muss.

Diese Arbeit liefert Load-Balancing-Ansätze, um eine effizientere Anwendung des Algorithmus von Funke und Sanders zu ermöglichen. Eine Verkleinerung des Randes um einen Faktor von bis zu 10 kann erreicht werden.

Kapitel 2 definiert grundlegende Begriffe und Algorithmen, die in den folgenden Kapiteln verwendet werden. Der in dieser Arbeit erbrachte Beitrag wird in Kapitel 3 festgehalten. Es werden verschiedene Ansätze zur Partitionierung einer Punktwolke eingeführt und ihre Parameter beschrieben. Schließlich dokumentiert Kapitel 4, inwiefern eine Leistungssteigerung gelungen ist und wie eine günstige Wahl der Ansätze und ihrer Parameter getroffen werden kann.

2 | Theoretische Grundlagen

In diesem Kapitel werden die grundlegenden in dieser Arbeit verwendeten Begriffe der Geometrie und der Graphpartitionierung definiert. Im Anschluss wird beispielhaft ein sequentieller Algorithmus zur Erzeugung einer Delaunay-Triangulierung beschrieben und der parallele Ansatz, der dieser Arbeit zu Grunde liegt, erläutert.

In dieser Arbeit sei stets $d, n \in \mathbb{N}_0$ und $d > 0$ sowie $n \geq 0$.

2.1. Geometrische Grundlagen

Definition - Punkte und Mengen.

(i) Eine endliche Teilmenge $P \subset \mathbb{R}^d$ der Menge reeller d-dimensionaler Koordinaten heißt Punktwolke.

(ii) Sei $p \in \mathbb{R}^d$, dann ist $\|p\|$ die euklidische Norm von p.

(iii) Sei $P \subset \mathbb{R}^d$ und $p \in \mathbb{R}^d$. Es gilt

$$dist(p, P) = \min \{ \|p - v\| \mid v \in P \}$$

.

Definition - konvexe Hülle.

(i) Eine Punktmenge $X \subset \mathbb{R}^d$ ist konvex genau dann, wenn für zwei Punkte $p_1, p_2 \in X$

$$\{ p_1 \cdot \alpha + p_2 \cdot (1 - \alpha) \mid \alpha \in [0, 1] \} \subset X \tag{2.1}$$

gilt.

(ii) Die konvexe Hülle $convexHull(X)$ von $X \subset \mathbb{R}^d$ ist definiert als die bezüglich Inklusion kleinste konvexe Punktmenge, sodass $X \subset convexHull(X)$. [6, S. 17]

Definition - geometrische Figuren.

(i) Eine geometrische Figur, die durch die Parameter $r \in \mathbb{R}$ und $m \in \mathbb{R}^d$ die Punkte $S = \{ p \in \mathbb{R}^d \mid \|p - m\| \leq r \}$ enthält, heißt geschlossene d-Sphäre. Dabei heißt r Radius und m Mittelpunkt der d-Sphäre.

(ii) Eine geometrische Figur, die durch die Parameter $r \in \mathbb{R}$ und $m \in \mathbb{R}^d$ die Punkte $S = \{ p \in \mathbb{R}^d \mid \|p - m\| < r \}$ enthält, heißt offene d-Sphäre. Dabei heißt r Radius und m Mittelpunkt der d-Sphäre.

(iii) Die konvexe Hülle X von $d + 1$ Punkten $V(X) \subset \mathbb{R}^d$ heißt d-Simplex. Die Elemente der Menge $V(X)$ heißen Vertizes.

(iv) Sei X ein d-Simplex. Die konvexe Hülle Y einer Teilmenge $V(Y) \subset V(X)$ der Punkte eines d-Simplex mit $|(V(Y))| = d$ heißt Seite.

(v) Eine geometrische Figur, die durch zwei Parameter $l, h \in \mathbb{R}^d$ definiert ist und die Punkte B mit

$$B = \{ p \in \mathbb{R} \mid \forall_{i \in \{1,\dots,d\}} : e_i \cdot l \leq e_i \cdot p \leq e_i \cdot h \}$$
$$\text{mit den Einheitsvektoren } e_1, \cdots, e_d \in \mathbb{R}^d$$

einschließt, heißt d-Box.

In dieser Arbeit wird eine offene d-Sphäre auch einfach d-Sphäre genannt.

Alternativ zur obigen Definition lässt sich ein d-Simplex auch als Figur aus $d + 1$ paarweise durch Kanten verbundener Punkte in $\mathbb{R}^d$ verstehen, da diese Figur immer eine konvexe Punktmenge einschließt und minimal bzgl. Inklusion ist. Ein 0-Simplex ist ein Punkt, ein 1-Simplex eine Strecke, ein 2-Simplex ein Dreieck und ein 3-Simplex ein Tetraeder.

Eine offene/geschlossene d-Sphäre S, die alle Punkte $V(X)$ eines d-Simplex X enthält und minimalen Radius hat, heißt offene/geschlossene Umkugel von X. Eine offene Umkugel wird in dieser Arbeit auch einfach Umkugel genannt. Der Mittelpunkt der Umkugel S von X hat zu allen Punkten $V(X)$ die euklidische Distanz r, wobei r der Radius von S ist.

Definition - Triangulierung.

Sei $P \subset \mathbb{R}^d$ endlich. Eine endliche Menge T von Simplizes heißt Triangulierung von P genau dann, wenn jede der folgenden Bedingungen zutrifft [5]:

(i) Die durch die Simplizes in T eingeschlossene Punktmenge ist die konvexe Hülle $convexHull(P)$ von P.

(ii) $\forall_{X_1, X_2 \in T} : X_1 \neq X_2 \Rightarrow X_1$ und X_2 schneiden sich nicht oder in einer Seite beider Simplizes.

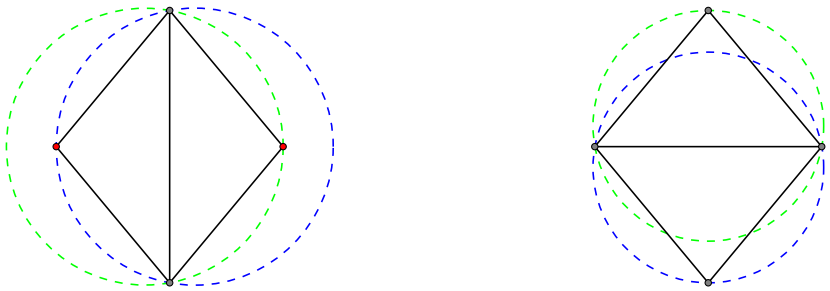
(iii) $P = \bigcup_{X \in T} V(X)$.

Die Triangulierung T einer Punktwolke $P \subset \mathbb{R}^d$ bildet einen ungerichteten Graphen $G = (P, E)$. Zwei Punkte $p_1, p_2 \in P$ haben genau dann eine Kante in G, wenn sie eine Kante in T haben. Es gilt also

$$(p_1, p_2) \in E \Leftrightarrow \exists_{X \in T} \colon p_1 \in V(X) \wedge p_2 \in V(X)$$

Definition - Delaunay-Triangulierung.

Eine Triangulierung T von $P \subset \mathbb{R}^d$ heißt genau dann Delaunay-Triangulierung, wenn für alle Simplizes $X \in T$ gilt, dass die offene Umkugel von X keinen Punkt $p \in P$ enthält.



(a) (b)

Abbildung 2.1.: Triangulierungen von vier Punkten: (b) zeigt eine Delaunay-Triangulierung; (a) ist keine Delaunay-Triangulierung

Abbildung 2.1b zeigt eine Delaunay-Triangulierung von vier Punkten. Die Umkugeln (hier auch: Umkreise) der Dreiecke enthalten jeweils keine Punkte. Abbildung 2.1a zeigt eine Triangulierung, die jedoch keine Delaunay-Triangulierung ist. Die roten Punkte sind jeweils in einem Umkreis enthalten.

Für jede Punktwolke existiert eine Delaunay-Triangulierung. Außerdem ist die Delaunay-Triangulierung einer Punktwolke $P \subset \mathbb{R}^d$ sogar eindeutig, wenn nie mehr als drei Punkte in P auf dem Rand einer offenen d-Sphäre liegen, die keine Punkte aus P enthält. [6, S. 32]

Die Voronoi-Region R_p des Punktes $p \in P$ der Punktwolke $P \subset \mathbb{R}^d$ ist durch

$$R_p = \{\, v \in P \mid \forall_{w \in P, w \neq v} \colon \|v - p\| < \|v - w\| \,\}$$

definiert.

Das Voronoi-Diagramm einer Punktwolke $P \subset \mathbb{R}^d$ ist dann die Menge der verbleibenden Punkte $R = \mathbb{R}^d \setminus \bigcup_{p \in P} R_p$.

2.2. Partitionen

Definition - Partitionierung.

Sei M eine Menge und $Q \subset Pot(M)$ eine endliche Teilmenge der Potenzmenge von M. Q ist eine Partitionierung von M genau dann, wenn folgende Bedingungen gelten:

(i) $\forall_{Q_1,Q_2 \in Q} \colon Q_1 \neq Q_2 \Rightarrow Q_1 \cap Q_2 = \emptyset$

(ii) $M = \bigcup_{Q_0 \in Q} Q_0$

Die Elemente von Q heißen Partitionen.

Beispielsweise ist durch die Voronoi-Regionen $R_{p_1}, \ldots, R_{p_n}$ und das Voronoi-Diagramm R einer Punktwolke $P = \{ p_1, \ldots, p_n \}$ eine Partitionierung Q von $\mathbb{R}^d$ definiert:

$$Q = \{ R_{p_1}, \ldots, R_{p_n}, R \}$$

Definition - Graphpartitionierung.

Sei $G = (V, E)$ ein ungerichteter Graph.

(i) Eine Partitionierung Q der Knotenmenge V heißt auch Partitionierung des Graphen G.

(ii) Sei $G = (V, E)$ ein Graph und Q eine Partitionierung des Graphen. Kanten, die durch die Partitionierung geschnitten werden, heißen Schnittkanten: Die Menge der Schnittkanten E_c ist definiert durch:

$$E_c = \{ e = \{ v, w \} \in E \mid \forall_{Q_0 \in Q} \colon v \notin Q_0 \vee w \notin Q_0 \}$$

(iii) Ist G ein Graph mit Kantengewichten $\omega \colon E \to \mathbb{N}_0$, dann ist der gesamte Schnitt *cut* des partitionierten Graphen definiert als

$$cut = \sum_{e \in E_c} \omega(e)$$

.

(iv) Ist G ein Graph ohne Kantengewichte, dann ist der gesamte Schnitt *cut* des partitionierten Graphen definiert als die Anzahl der geschnittenen Kanten

$$cut = |E_c|$$

.

2.3. Sequentielle Berechnung einer Delaunay-Triangulierung

Der Bowyer-Watson-Algorithmus eignet sich zur sequentiellen Bestimmung einer Delaunay-Triangulierung. Sei $DT(P)$ eine Delaunay-Triangulierung einer Punktwolke $P \subset \mathbb{R}^d$ und $p \in \mathbb{R}^d$ mit $p \notin P$ ein Punkt. Für den Fall, dass p in $DT(P)$ eingeschlossen ist, wird die Delaunay-Triangulierung $DT(P \cup \{ p \})$ von $P \cup \{ p \}$ in folgenden Schritten bestimmt [6, S. 59] [6, S. 106].

Algorithm 1 Bowyer-Watson-Algorithmus

procedure ADDPOINT(p, DT)
  $S \leftarrow \{ s \in DT \mid$ Umkugel von s umschließt $p. \}$
  $F \leftarrow$ Seiten, die je ein Simplex aus S und ein Simplex aus $DT \setminus S$ trennen.
  $DT \leftarrow DT \setminus S$
  for all Seite $f \in F$ **do**
    Erzeuge einen Simplex s aus f und p.
    $DT \leftarrow DT \cup \{ s \}$
  return DT

Um den Aufwand, die Simplizes S zu finden, zu verringern, können ausgehend von dem Simplex, das den neuen Punkt p enthält, alle weiteren Simplizes durch eine Tiefensuche gefunden werden. Das Loch, das durch das Löschen von S aus DT entsteht, ist stets so geformt, dass sich die hinzugefügten und die verbliebenen Simplizes nicht überschneiden. Die so entstehende Triangulierung ist auch eine Delaunay-Triangulierung.

Bevor ADDPOINT ausgeführt werden kann, muss zuerst eine Delaunay-Triangulierung generiert werden. Hierzu eignet sich ein d-Simplex, das so gewählt wird, dass es alle Punkte, die anschließend hinzugefügt werden, enthält.

Nach dem letzten Aufruf von ADDPOINT werden die Eckpunkte des initialen Simplex und alle inzidenten Kanten aus der Triangulation entfernt.

2.4. Parallele Berechnung einer Delaunay-Triangulierung

Der folgende Pseudocode zeigt eine modifizierte Version des Teile-und-herrsche-Algorithmus in [5] zur Bestimmung der Delaunay-Triangulierung in $\mathbb{R}^d$ auf Shared-Memory-Systemen. Der Algorithmus lässt zwei Operationen *partition* und *intersects* offen, die im Rahmen dieser Arbeit implementiert und untersucht werden.

Die in dieser Arbeit implementierte Version des Algorithmus geht in zwei Schritten vor: Zunächst wird die Partitionierung der Punktwolke berechnet und anschließend die Triangulierung der partitionierten Punktwolke. Jede Partition der Partitionierung wird separat von den anderen durch einen sequentiellen Algorithmus trianguliert. Diese Teiltriangulierungen $T_1, \ldots, T_p$ und der Rand B (Border) werden parallel erzeugt, entsprechen also im Sinne der Laufzeituntersuchung dem parallelisierbaren Teil des Algorithmus.

Die Triangulierung von B hingegen erfolgt sequentiell und das anschließende Zusammensetzen (Merging) des triangulierten B mit den Teiltriangulierungen $T_1, \ldots, T_p$ wieder parallel. Funke und Sanders sehen auch für die Triangulierung des Randes einen rekursiven

Aufruf und somit eine Parallelisierung vor. Da sich diese Arbeit primär mit dem Vergleich von Partitionierungsansätzen und weniger mit dem Finden eines optimalen Algorithmus beschäftigt, verzichtet sie auf eine parallele Triangulierung des Randes.

Algorithm 2 Partitionieren und Triangulieren

procedure CALCULATEDELAUNAYTRIANGULATION(points)
  $partitioning = \{ Q_1, \ldots, Q_p \} \leftarrow partition(points)$
  return triangulate(partitioning)
procedure TRIANGULATE($\{ Q_1, \ldots, Q_p \}$)
  $(T_1, \ldots, T_p) \leftarrow (triangulateBase(Q_1), \ldots, triangulateBase(Q_p))$
  $B \leftarrow \emptyset; S \leftarrow convexHull(T_1) \cup \cdots \cup convexHull(T_p)$
  while $S \neq \emptyset$ **do**
    Entferne einen Simplex $s \in S$, der zur Teiltriangulierung T_i gehört.
    if $\exists_{j \neq i} intersects_j(circumsphere(s))$ **then**
      $B \leftarrow B \cup \{ s \}$
      $S \leftarrow S \cup neighbors(s)$
  $T \leftarrow (T_1 \cup \cdots \cup T_p) \setminus B$
  $T_B \leftarrow triangulateBase(vertices(B))$
  Merge T_B into T
  return T

Ein besonders auffälliger Unterschied ist die Aufgabe des Teile-und-herrsche-Ansatzes, der durch eine Vorabpartitionierung der Eingabe ersetzt wurde. Der Algorithmus von Funke und Sanders sieht zur Partitionierung eine Zweiteilung des $\mathbb{R}^d$ vor, die durch den rekursiven Abstieg verfeinert wird. Diese Arbeit tauscht die schrittweise Teilung durch verschiedene Implementierungen der Operation *partition* aus, die die Punktwolke global und im Voraus in p Teile partitioniert.

Die *intersects*-Operation verallgemeinert den Ansatz aus Funke und Sanders Arbeit, die Partitionen mit einer einfachen Bounding-Box (d-Box) zu umschließen und den Schnitt mit $circumsphere(s)$ zu prüfen. Eine alternative Implementierung von *intersects* wird in dieser Arbeit besprochen.

Aus dem Pseudocode lässt sich grob die Gesamtlaufzeit von der Punktwolke bis zur fertigen Delaunay-Triangulierung ermitteln:

$$T_{ges} \approx T_{part}(|P|) + T_{seq}(\max_{Q_0 \in Q}(|Q_0|)) + T_{seq}(|B|) \tag{2.2}$$

mit P: Punktwolke,
$Q = \{ Q_1, \ldots, Q_p \}$: Partitionierung von P,
B: Rand,
$T_{part}(n)$: Laufzeit der Partitionierung von n Punkten und
$T_{seq}(n)$: Laufzeit der sequentiellen Triangulierung von n Punkten.

Die Gleichung vernachlässigt das ermitteln des Randes B und die Zusammensetzung

(Merging) der Teiltriangulierungen und der Triangulierung des Randes, zeigt jedoch den Einfluss der Wahl der Partitionierung Q und der Größe $|B|$ des Randes.

Der Rest dieses Kapitels beschäftigt sich mit verwandten Arbeiten, die im Zusammenhang mit Load-Balancing für parallele Delaunay-Triangulierung nicht unerwähnt bleiben sollen.

Farhat et al. [2] beschreiben einen Algorithmus zur Zerlegung von gegebenen Netzen und wenden ihn als Vorberechnungsschritt für die Finite-Elemente-Methode an. Insbesondere erfüllt der Algorithmus folgende Anforderungen: Die Zerlegung des Netzes muss balanciert sein und die Anzahl der Knoten, die an den Schnittstellen zu mehreren Teilnetzen gehören, muss minimiert werden.

Said et al. [7] beschreiben einen verteilten, parallelen Algorithmus, der eine Delaunay-Triangulierung erzeugt. Dabei wird der Algorithmus von Farhat et al. für das Load-Balancing genutzt, indem die Grenzen der geometrischen Eingabedaten trianguliert werden und die Triangulierung zerlegt wird. Die Anzahl der Teilnetze wird bewusst größer als die Anzahl der Recheneinheiten gewählt, um dynamisches Load-Balancing zu ermöglichen. Said beschränkt sich in seiner Veröffentlichung auf 2- und 3-dimensionale Probleme.

Eine Arbeit von Chrisochoides et al. [3] beschreibt ein Verfahren zur Erzeugung und Partitionierung einer Delaunay-Triangulierung. Der Algorithmus wird für den 3D-Fall beschrieben, er sei aber für beliebige Dimensionen anwendbar. Zur Triangulierung wird eine parallele Variante des inkrementellen Bowyer-Watson-Algorithmus [8, 9] verwendet. Die Partitionierung des Netzes wird simultan mit der Triangulierung durchgeführt, was einen Effizienzvorteil gegenüber Ansätzen bringt, bei denen erst im Anschluss an die Triangulierung partitioniert wird. Wie in der Veröffentlichung von Farhat et al. handelt diese nicht von Partitionierung einer gegebenen Punktwolke, sondern beschreibt ein Verfahren zur Erzeugung einer partitionierten Delaunay-Triangulierung einer Punktwolke. Auch hier wird als Anwendungsfall die Lösung partieller Differentialgleichungen mit der Finite-Elemente-Methode genannt.

Lee et al. [10] beschreiben einen parallelen Algorithmus, der die Delaunay-Triangulierung auf verteiltem Speicher berechnet, und konnten durch eine geeignete Partitionierungsstrategie das aufwändige Zusammensetzen der Teilnetze vermeiden. Die Partitionierungsphase nimmt jedoch 75% der Gesamtzeit in Anspruch [11].

3 | Load-Balancing durch Partitionierung der Punktwolke

Durch Load-Balancing wird die Eingabe auf mehrere parallel arbeitende Recheneinheiten aufgeteilt. Die Aufteilung der Arbeit beim Load-Balancing für parallele Delaunay-Triangulierung wird durch die Partitionierung der eingegebenen Punktwolke gelöst. Gleichung 2.2 zeigt worauf es dabei ankommt. Die Partitionierung Q muss balanciert sein, sodass $\max_{Q_0 \in Q}(|Q_0|)$ möglichst klein ist. Im Idealfall gilt $|Q_1| = |Q_2| = \cdots = |Q_p|$. Um eine geringe Laufzeit für die Triangulierung des Randes zu erreichen, ist außerdem $|B|$ klein zu halten. Der Rand hängt von der Partition und der Wahl der *intersects*-Funktion ab.

In diesem Kapitel werden die Ansätze beschrieben, mit denen das Problem der Partitionierung der eingegebenen Punktwolke zu lösen versucht wird. Zunächst folgt eine allgemeine Beschreibung des generellen Partitionierungsansatzes, um anschließend im Detail auf den Vorgang und dessen Variationen einzugehen.

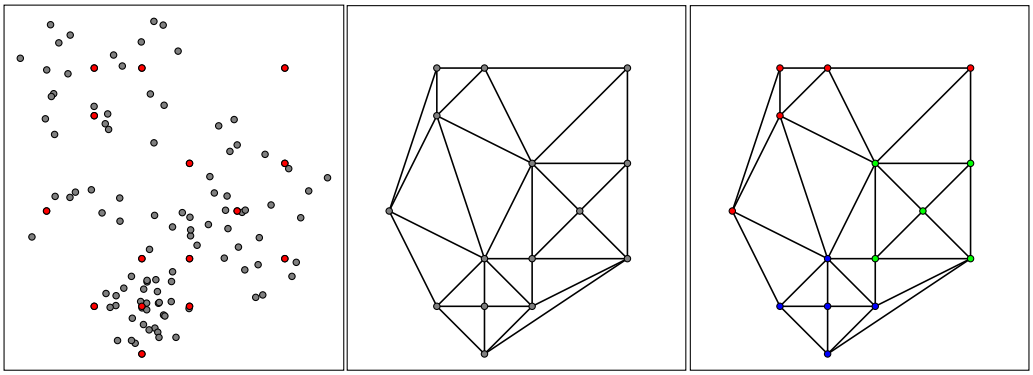
Ziel ist es einen Algorithmus zu finden, der für eine eingegebene Punktwolke $P \subset \mathbb{R}^d$ eine Partitionierung $Q = \{ Q_1, Q_2, \ldots, Q_p \}$ bildet.

Alle Partitionierungsstrategien, die in dieser Arbeit untersucht werden, verfolgen denselben Ansatz. Das Problem auf einer Teilmenge der Eingabe zu lösen und ausgehend davon eine Lösung für die gesamte Eingabe zu entwickeln, ist eine Idee, die auf dem Sortieralgorithmus „Samplesort“ [12] basiert, der 1970 von Frazer und McKellar beschrieben wurde. Samplesort lässt sich als eine Variante von Quicksort verstehen, bei der in jedem Rekursionsschritt mehrere zufällige Pivots gewählt werden. Dadurch haben die Pivots dieselbe Verteilung wie die gesamte Eingabe, die dadurch gleichmäßiger aufgeteilt werden kann.

Das Sample in dieser Arbeit ist im Vergleich zu Samplesort jedoch eine partitionierte Triangulierung. Somit ist zur Erzeugung dieses Samples ein Algorithmus nötig, um eine Triangulierung zu partitionieren. Dazu verwendet diese Arbeit den von Sanders und Schulz entwickelten Algorithmus [13] zur Berechnung von Graphpartitionierungen mit minimalem gesamten Kantenschnitt.

Insgesamt lässt sich der Ablauf von der eingegebenen Punktwolke bis zur fertigen Partitionierung folgendermaßen beschreiben.

Mehrere Varianten der Subroutinen SAMPLE, EXTEND und GENERATEBOUNDS existieren und werden in dieser Arbeit untersucht.



(a) Zufälliges Sample der Punktwolke (b) Sampletriangulierung (c) Samplepartitionierung

Abbildung 3.1.: Erzeugung einer Samplepartitionierung: Die roten Punkte in (a) sind die zufällig gewählten Samplepunkte, deren Delaunay-Triangulierung in (b) dargestellt wird. (c) zeigt drei Samplepartitionen deren Punkte je durch die Farbe Rot, Grün oder Blau markiert sind.

Algorithm 3 Allgemeiner Partitionierungsalgorithmus

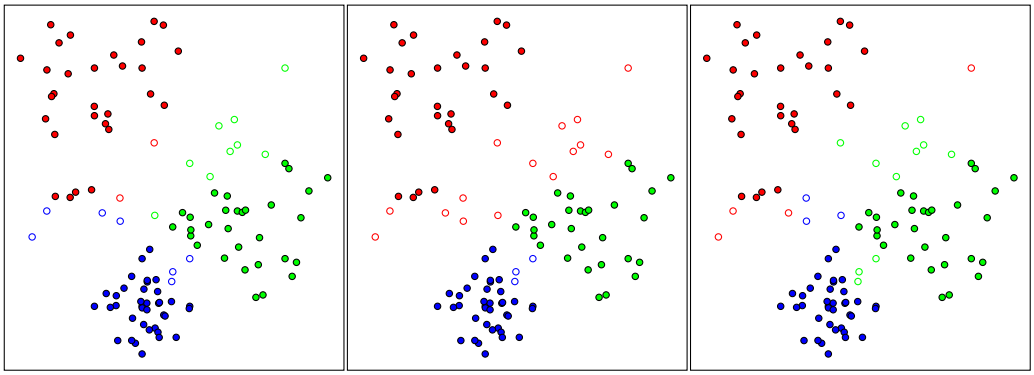
```
procedure PARTITION(points)
    samplePartitioning ← SAMPLE(points)
    partitioning ← EXTEND(points, samplePartitioning)
    boundPartitioning ← GENERATEBOUNDS(partitioning)
    return boundPartitioning
```

In diesem Kapitel werden diese Subroutinen und ihre Varianten ausführlich beschrieben und hier nur kurz einleitend zusammengefasst. Die SAMPLE-Routine trianguliert und partitioniert eine Teilmenge der Punktwolke (Abbildung 3.1). Daraufhin generiert die EXTEND-Routine auf Grundlage der kleinen Partitionierung eine Partitionierung der gesamten Punktwolke (Abbildung 3.2). Schließlich werden in der GENERATEBOUNDS-Routine die Begrenzungen der Partitionen ermittelt, um dem Triangulierungsalgorithmus zu erlauben, effizient Schnitte von Sphären und Partitionen zu testen.

In dieser Arbeit wird stets der Fall untersucht, in dem die Anzahl der Partitionen p gleich der Anzahl der Threads ist, auf denen die anschließende parallele Triangulierung ausgeführt wird. Der verfolgte Load-Balancing-Ansatz ist also statisch. Eine große Anzahl Partitionen führt zu einer Vergrößerung des Randes und damit zu einem größeren Anteil nicht parallelisierbarer Berechnungen. Auch wird bei den untersuchten Partitionierungsstrategien auf balancierte Partitionsgrößen geachtet. Dynamisches Load-Balancing ist für diese Arbeit also keine verheißungsvolle Alternative und wird nicht weiter betrachtet.

Im Folgenden wird an mehreren Stellen die Generierung von Zufallszahlen nötig sein. Die Implementierung verwendet die Mersenne-Twister-Engine der C++-Standardbibliothek.

Die verbleibenden Abschnitte dieses Kapitels gehen im Detail auf die Implementierungen der SAMPLE-, EXTEND- und GENERATEBOUNDS-Subroutinen und ihre Variationen ein.



(a) (b) (c)

Abbildung 3.2.: Drei verschiedene Partitionierungen derselben Punktwolke; die Punkte derselben Farbe gehören zur selben Partition. Die nicht ausgefüllten Punkte sind Punkte, die nicht in jeder der Partitionierungen derselben Partition zugeordnet sind. Jedes Partitionierung ist das Ergebnis einer Partitionierungsstrategie: (a) Zuweisung zu nächstem Samplepartitionszentrum; (b) Zuweisung zu nächster Samplepartitionsbegrenzung; (c) Zuweisung zu Partition des nächsten Samplepunkts

3.1. Partitionieren anhand einer Sampletriangulierung

Im ersten Schritt wird eine zufällige Teilmenge $P_s \subset P$ der Eingabepunkte ausgewählt. Anschließend wird die Delaunay-Triangulierung T_s dieser Samplepunkte bestimmt und der durch die Triangulierung entstehende Graph $G_s = (P_s, E_s)$ partitioniert:

$$Q_s = \{ Q_{s,1}, \ldots, Q_{s,p} \}$$

Für die Graphpartitionierung wird KaFFPa [14] verwendet. Der Ablauf wird in Abbildung 3.1 veranschaulicht.

Allein dieser erste Schritt lässt bereits einige Parameter zur Konfiguration übrig. Hierzu zählen einerseits die Anzahl der Samplepunkte $|P_s|$ und der Graphpartitionen p_G und andererseits die Parameter des KaFFPa-Algorithmus:

- Das Ungleichgewicht der Graphpartitionierung
- Ein Seed zur Generierung von Zufallszahlen
- Der Modus der Graphpartitionierung

Des Weiteren beachtet KaFFPa auch Knoten- und Kantengewichte des Graphen G_s.

Einige Parameter werden für Optimierungszwecke in Betracht gezogen, während andere einen festen Wert erhalten. Um später aus den Graphpartitionen die Punktwolkenpartitionen $\{ Q_1, \ldots, Q_p \}$ zu bilden, wird für die Anzahl der Graphpartitionen $p_G = p$ gewählt.

Die Anzahl der Samplepunkte $|P_s|$ stellt einen wichtigen Parameter dar, der in der Evaluierung genauer untersucht wird.

Für das Ungleichgewicht der Graphpartitionierung wird 5% gewählt. Das heißt für jede Partition $i \in \{ 1, \ldots, p_G \}$ weicht die Größe $|\{ v \in P_s \mid i = part(v) \}|$ höchstens um 5% von der mittleren Graphpartitionsgröße ab.

Der Seed wird zufallsgeneriert und der Modus ist „fast“. Genaue Informationen insbesondere zum Modus sind in [15] zu finden.

Diese Arbeit verzichtet auf Knotengewichte, jedoch wird versucht mit Kantengewichten, die abhängig von der euklidischen Distanz gewählt werden, günstigere Samplepartitionierung zu erzeugen. Geringe Kantengewichte an langen Kanten und umgekehrt hohe Kantengewichte an kurzen Kanten könnten die Zuordnung von Samplepunkten, die sich im selben Cluster befinden, zu unterschiedlichen Graphpartitionen seltener machen. Das wiederum würde es erleichtern, solche Cluster im nächsten Schritt nicht zu zerteilen, und so die Bildung von Partitionen, die für das Load-Balancing günstig sind, fördern.

Untersucht werden drei Strategien Kantengewichte zu vergeben: konstant, linear und quadratisch. Sei $e = (p_0, p_1)$ die Kante, die die Samplepunkte p_0 und p_1 verbindet, dann ist ihr Gewicht $c(e)$ abhängig von der verwendeten Strategie $c(e) = 1$ (konstant), $c(e) = \frac{1}{\|p_0 - p_1\|}$ (linear) oder $c(e) = \frac{1}{\|p_0 - p_1\|^2}$ (quadratisch).

Tatsächlich lässt KaHIP nur positive, ganzzahlige Kantengewichte zu, weshalb die tatsächlichen Kantengewichte noch diskretisiert werden müssen.

Algorithm 4 Erzeugung einer Samplepartitionierung

procedure SAMPLE(P)
  Wähle Samplepunkte $P_G \subset P$
  $sampleDT \leftarrow$ TRIANGULATE(P_G)
  Erstelle den Graph (P_G, E) von $sampleDT$
  $Q_s gets$ partitioniere (P_G, E)
  return Q_s

Ausgehend von der Samplepartitionierung Q_s ist nun die am Anfang des Kapitels geforderte Partitionierung $Q = \{ Q_1, \ldots, Q_p \}$ gefragt. Im Folgenden werden drei Strategien vorgestellt, die mit Hilfe der in diesem Abschnitt besprochenen partitionierten Sampletriangulierung eine Partitionierung der gesamten Punktwolke gewinnen. An die Beziehung von Q und Q_s werden keinerlei formale Forderungen gestellt. Die folgenden Strategien sind also frei in der Art und Weise, wie sie Q aus Q_s konstruieren. Insbesondere gilt nicht notwendigerweise, dass für alle $i \in \{ 1, \ldots, p \}$ die Samplepartition $Q_{s,i}$ eine Teilmenge der Partition Q_i ist. Wie sich die Wahl einer der folgenden Strategien auf die Qualität der Partitionierung Q auswirkt, kann und wird in dieser Arbeit nur empirisch ermittelt werden.

3.2. Zuweisung zu Samplezentren

Das Zentrum z_i der Samplepartition $Q_{s,i}$ sei als das arithmetische Mittel

$$z_i = \frac{1}{|Q_{s,i}|} \sum_{p \in Q_{s,i}} p$$

der Punkte $p \in Q_{s,i}$ definiert.

Für jeden Punkt $p \in P$ wird nun das euklidisch nächste Zentrum gefunden und die entsprechende Partition zugewiesen.

$$part(p) = \underset{i}{\operatorname{argmin}} \left(\|p - z_i\| \right)$$

Falls mehrere Zentren denselben euklidischen Abstand zum Punkt p haben, ist $part(p)$ nicht eindeutig. Für die theoretische Betrachtung und die praktische Implementierung genügt jedoch, dass die Partition in einem solchen Fall beliebig aus allen euklidisch nächsten Zentren gewählt wird. Auch auf die Laufzeit hat dieser Fall aufgrund seiner geringen Auftrittswahrscheinlichkeit keinen Einfluss.

Es ergibt sich die gesuchte Partitionierung $Q = \{ Q_1, \ldots, Q_p \}$ durch

$$Q_i = \{ p \mid part(p) = i \}, \qquad i = 1, \ldots, p$$

.

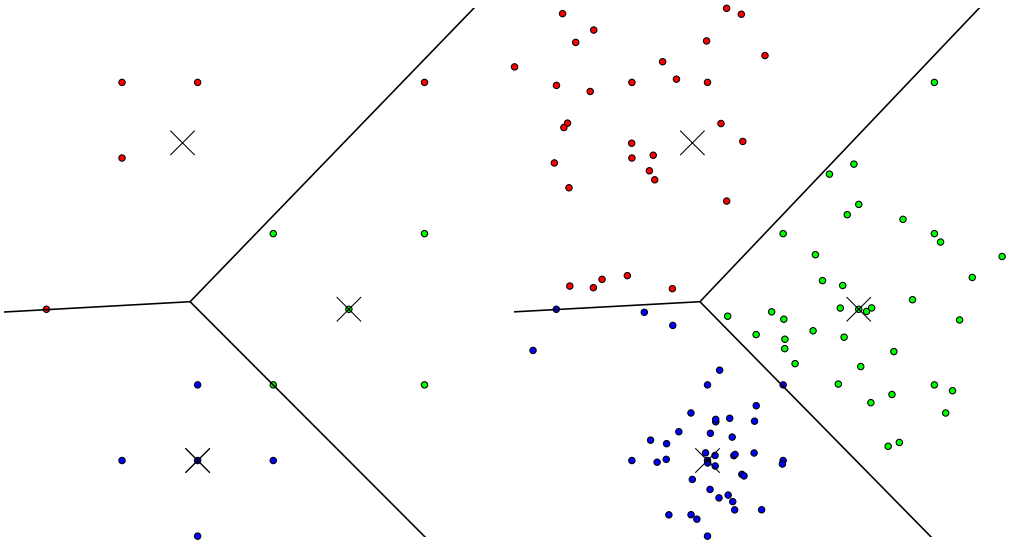
Algorithm 5 Zuweisung zu arithmetischen Mittelpunkten der Samplepartitionen

procedure EXTEND(P, $Q_s = \{ Q_{s,1}, \ldots, Q_{s,p} \}$)
  $Q_1, \ldots, Q_p \leftarrow \emptyset$
  for all $i \in \{ 1, \ldots, p \}$ **do**
    $z_i \leftarrow \frac{1}{|Q_{s,i}|} \sum_{p \in Q_{s,i}} p$
  for all $p \in P$ **do**
    $part = \operatorname{argmin}_i \left(\|p - z_i\| \right)$
    $Q_{part} \leftarrow Q_{part} \cup \{ p \}$
  return $\{ Q_1, \ldots, Q_p \}$

Wird das Voronoi-Diagramm der Zentren $\{ z_1, \ldots z_p \}$ betrachtet, entsteht eine alternative Anschauung: Wie durch Abbildung 3.3 verdeutlicht, wird die Punktwolke P durch Voronoi-Regionen partitioniert. Sei $V_i \subset \mathbb{R}^d$ die Voronoi-Region mit $z_i \in V_i$, dann ergibt sich äquivalent zur obigen Gleichung

$$Q_i = V_i \cap P.$$

Auch mit dieser Anschauung ist Q nicht eindeutig, da die Punkte entlang der Grenzen der Voronoi-Regionen nicht eindeutig einer Region zugeordnet sind.



(a) (b)

Abbildung 3.3.: Punkte zugewiesen zu nächstem Zentrum: Die Kreuze markieren die Zentren der Samplepartitionen, die in (a) dargestellt sind. Die Punkte in (b) sind je der Partition mit dem euklidisch nächsten Zentrum zugeordnet. Die Zuordnung der Punkte wird auch durch das dargestellte Voronoi-Diagramm der Zentren deutlich.

3.3. Zuweisung zu Samplebegrenzungen

Sei $i \in \{1, \ldots, p\}$. Eine Punktmenge $B_i \subset \mathbb{R}^d$ ist eine Begrenzung der Samplepartition $Q_{s,i}$ genau dann, wenn

$$Q_{s,i} \subset B_i$$

gilt. Die Wahl der Begrenzung ist entscheidend für die Qualität der Partitionierung und die nötige Zeit für ihre Berechnung. Zunächst sei die Wahl jedoch offen. Nun lässt sich alternativ zur oben beschriebenen Partitionierung durch Samplezentren eine Partitionierung durch Samplebegrenzungen definieren.

Jedem Punkt $p \in P$ wird die euklidisch nächste Samplebegrenzung zugewiesen:

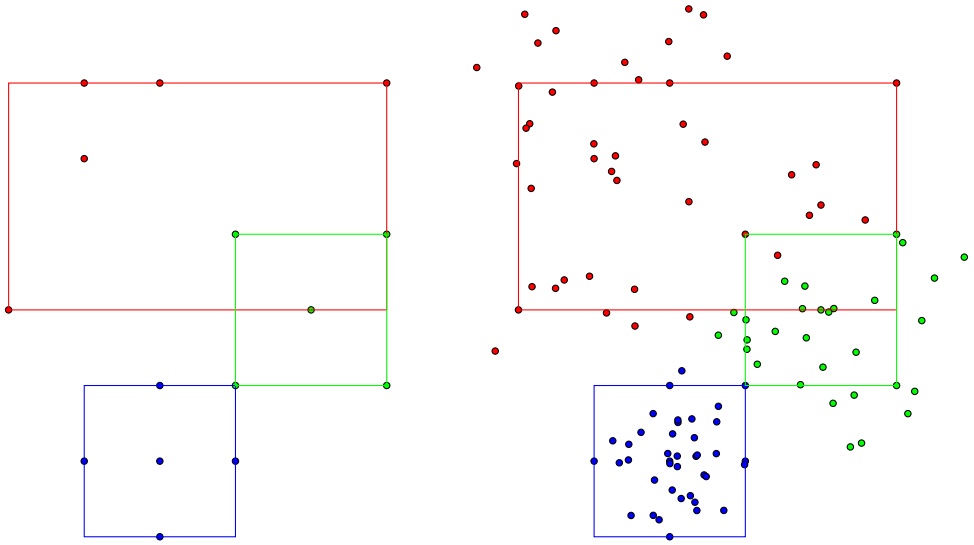
$$part(p) = \operatorname*{argmin}_i \left(dist(p, B_i)\right)$$

Die Definition der Begrenzungen $B_1, \ldots B_p$ garantiert nicht, dass sie paarweise disjunkt sind. In anderen Worten heißt dies, dass die Begrenzungen sich überlappen können. Ein Punkt $p \in B_a \cap B_b$ innerhalb der Überlappung der Begrenzungen B_a und B_b hat dieselbe Distanz $dist(p, B_a) = dist(p, B_b) = 0$ zu beiden Begrenzungen. Das Problem mit der Mehrdeutigkeit von $part(p)$ wird dadurch gegenüber der Strategie mit den Samplezentren noch gewaltig verschärft.

Die gesuchte Partitionierung $Q = \{ Q_1, \ldots, Q_p \}$ ergibt sich dann genau wie zuvor durch

$$Q_i = \{ p \mid part(p) = i \}, \qquad i = 1, \ldots, p$$

.



(a) (b)

Abbildung 3.4.: Punkte zugewiesen zu nächster Box: (a) zeigt die partitionierten Samplepunkte und ihre umschließenden Boxen. Die Punkte in (b) sind der jeweils nächsten Box zugeordnet. Auch sichtbar ist, dass in den Überlappungen der Boxen die Zuordnung der Punkte nicht eindeutig ist.

Der einfachste Ansatz eine Begrenzung zu wählen, der zugleich auch keine aufwändigen Berechnungen notwendig macht, ist die Wahl von Boxen, die an den Achsen ausgerichtet sind. Abbildung 3.4 veranschaulicht dies anhand eines Beispiels mit zwei Dimensionen. Für $i \in \{ 1, \ldots, p \}$ wird die kleinste Box B_i gewählt, die auch eine Begrenzung von $Q_{s,i}$ ist. Weitere Ansätze, die kleinere Begrenzungen $B'_i \subset B_i$ liefern und so besser die Form der Samplepartition wiedergeben, sind denkbar, diese Arbeit untersucht jedoch nur den einfachen Box-Ansatz.

Auch auf das Problem der Überlappungen hat die Wahl der Art der Begrenzungen einen großen Einfluss. Ein Beispiel illustriert die Schwere dieses Umstandes: Die Dimension sei $d = 2$, also sind die Begrenzungen Rechtecke mit Kanten, die parallel zu den Achsen ausgerichtet sind. Die Samplepartitionierung $Q_s = \{ Q_{s,1}, Q_{s,2} \}$ enthalte zwei Partitionen. $Q_{s,1} \cup Q_{s,2}$ forme ein Quadrat, wobei $Q_{s,1}$ den unteren linken und $Q_{s,2}$ den oberen rechten Teil ausfüllen und jeweils ein rechtwinkliges Dreieck darstellen. Es ist nun festzustellen, dass sich die Begrenzungen von $Q_{s,1}$ und $Q_{s,2}$ fast überdecken. Dies kann dazu führen, dass fast alle Punkte $p \in P$ zur selben Partition Q_1 oder Q_2 zugeordnet werden. In diesem nicht unwahrscheinlichen Extremfall wäre die Partitionierung wertlos.

Algorithm 6 Zuweisung zu samplepartitionsumschließende Boxen

procedure EXTEND(P, $Q_s = \{ Q_{s,1}, \ldots, Q_{s,p} \}$)
  $Q_1, \ldots, Q_p \leftarrow \emptyset$
  for all $i \in \{ 1, \ldots, p \}$ **do**
    Erzeuge Box B_i, die $Q_{s,i}$ enthält.
  for all $p \in P$ **do**
    $part = \text{argmin}_i \, (dist(p, B_i))$
    $Q_{part} \leftarrow Q_{part} \cup \{ p \}$
  return $\{ Q_1, \ldots, Q_p \}$

Die Anschauung als Voronoi-Diagramm wie bei den Samplezentren ist leider nicht so einfach möglich und wird daher hier nicht weiter motiviert.

3.4. Zuweisung zum nächstem Samplepunkt

Mit dieser Strategie ist die Partition jedes Punktes $p \in P$ durch

$$part(p) = \operatorname*{argmin}_i \left(dist(p, Q_{s,i})\right)$$

definiert.

Streng genommen ist dies ein Spezialfall der Strategie mit Samplebegrenzungen, in dem für alle $i \in \{\, 0, \ldots, p-1 \,\}$ die Begrenzung B_i durch

$$B_i = Q_{s,i}$$

gegeben ist, wird aber in dieser Arbeit nicht weiter als solcher betrachtet.

Das Problem mit der Mehrdeutigkeit von $part(p)$ für bestimmte Punkte $p \in P$ besteht auch hier, jedoch sind die Begrenzungen $B_1, \ldots, B_p$ paarweise disjunkt, da die Samplepartitionen $Q_{s,p}, \ldots, Q_{s,p}$ per Definition paarweise disjunkt sind.

Die gesuchte Partitionierung $Q = \{\, Q_1, \ldots, Q_p \,\}$ ergibt sich dann abermals genau wie zuvor durch

$$Q_i = \{\, p \mid part(p) = i \,\}, \qquad i = 1, \ldots, p$$

.

Algorithm 7 Zuweisung zu Samplepunkten

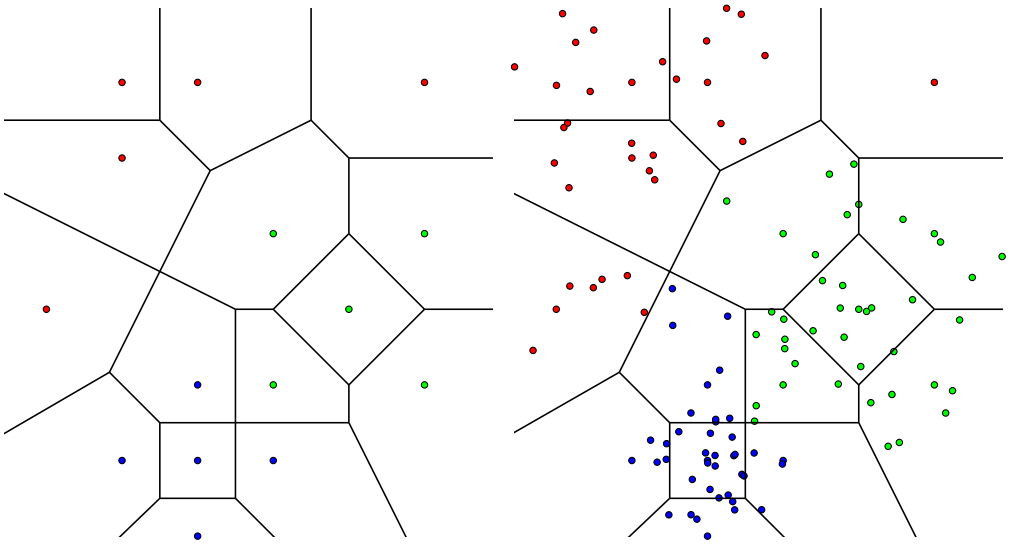
procedure EXTEND(P, $Q_s = \{\, Q_{s,1}, \ldots, Q_{s,p} \,\}$)
  $Q_1, \ldots, Q_p \leftarrow \emptyset$
  for all $p \in P$ **do**
    $p_s = \operatorname{argmin}_{p_s \in P_s} (\|p - p_s\|)$ ▷ Nächste-Nachbar-Suche
    $i \leftarrow$ Index der Partition Q_i mit $p_s \in Q_i$
    $Q_i \leftarrow Q_i \cup \{\, p \,\}$
  return $\{\, Q_1, \ldots, Q_p \,\}$

Für diese Strategie kann wieder die Anschauung durch ein Voronoi-Diagramm bemüht werden. Sei $V_p \subset \mathbb{R}^n$ die Voronoi-Region mit $p \in V_p$ im Voronoi-Diagramm der Samplepunkte P_s. Dann entsteht durch Zusammensetzen der Voronoi-Regionen, die zu Punkten derselben Samplepartition gehören, die Einteilung des Raumes $\mathbb{R}^d$, die genau der Partitionierung Q der Punktwolke P entspricht:

$$Q_i = \{\, V_p \mid p \in Q_{s,i} \,\} \cap P$$

Der zweidimensionale Fall wird in Abbildung 3.5 illustriert.

Auch hier sei darauf hingewiesen, dass Q_i durch die Gleichung nicht eindeutig bestimmt ist.



(a) (b)

Abbildung 3.5.: Punkte zugewiesen zu nächstem Samplepunkt: (a) zeigt die partitionierten Samplepunkte und ihr Voronoi-Diagramm. Die Punkte in (b) sind ihrem euklidisch nächstem Samplepunkt zugeordnet. Anschaulich teilen die Samplepunkte den Raum in Gebiete, die ihrerseits aus Voronoi-Regionen derselben Samplepartition bestehen.

3.5. Präzision der Partitionsgrenzen

Nachdem der Triangulierungsalgorithmus die Partitionen der Punktwolke trianguliert hat, ermittelt er, welche Punkte zum Rand gehören und erneut trianguliert werden müssen. Hierzu ist für $i \in \{1, \ldots, p\}$ eine Funktion

$$intersects_i \colon \mathbb{R}^d \times \mathbb{R} \to \{true, false\},$$

mit $intersects_i(v, r) = true$, falls mindestens ein Punkt der Partition Q_i in der Sphäre mit dem Mittelpunkt v und dem Radius r enthalten ist, nötig. Wird diese Bedingung als strikte Äquivalenz verstanden, erhält man genau die folgende Vorschrift:

$$intersects_i(v, r) = \begin{cases} true, & \text{falls } \emptyset \neq Q_i \cap S \\ false, & \text{sonst} \end{cases},$$
$$\text{mit } S = \{ x \in \mathbb{R}^d \mid \|x - v\| \leq r \}$$

Der Triangulierungsalgorithmus erlaubt entsprechend der Definition von $intersects_i$ jedoch auch, dass, entgegen dieser Gleichung, $intersects_i(v, r)$ selbst dann zu $true$ auswertet, wenn kein Punkt aus Q_i in der Sphäre enthalten ist. Dies ist wegen der hohen Zahl an Aufrufen von $intersects_i$ für eine effiziente Implementierung nötig und die zentrale Abwägung bei der Wahl der $intersects_i$-Funktionen. Feinere Schnittprüfungen führen zu einem kleineren Rand, mit gröberen kann jedoch eine geringere Laufzeit der $intersects_i$-Funktionen

erreicht werden. In dieser Arbeit werden zwei Alternativen untersucht, $intersects_i$ zu implementieren, von denen die zweite feiner als die erste ist.

Die erste, die bereits von Funke und Sanders in ihrem Triangulierungsalgorithmus implementiert wurde, verwendet je Partition eine an den Achsen ausgerichtete Box, die die Partition umfasst. Sei $i \in \{ 1, \dots, p \}$ und B_i die Box mit dem geringsten Volumen, die die Partition Q_i enthält, sodass $Q_i \subset B_i$ gilt. Dann kann $intersects_i(v, r)$ formal durch

$$intersects_i(v, r) = \begin{cases} true, & \text{falls } \emptyset \neq B_i \cap S \\ false, & \text{sonst} \end{cases},$$
$$\text{mit } S = \{ x \in \mathbb{R}^d \mid \|x - v\| \leq r \}$$

beschrieben werden. Das Ergebnis ist dann der Wahrheitswert von $\|v\| \leq r$.

Hierbei sei erneut angemerkt, dass die $intersects_i$-Funktionen erst während der anschließend stattfindenden Triangulierung aufgerufen werden. Lediglich die Erstellung der Box $B_i \supset Q_i$ ($i \in \{ 1, \dots, p \}$) ist Aufgabe des GENERATEBOUNDS-Schritts.

Die zweite Alternative teilt den Raum $\mathbb{R}^d$ in ein Raster ein. Jeder Partition werden dann Elemente des Rasters zugewiesen, die zusammengesetzt die Partition umfassen. Ein d-dimensionales Raster lässt sich allein durch die Breite seiner Zellen $w \in \mathbb{R}$ definieren. Das Raster setzt sich also unendlich in alle $2 \cdot d$ Richtungen fort. Ein Element der Menge $\mathbb{Z}^d \subset \mathbb{R}^d$ sind die diskreten Koordinaten genau einer Zelle des Rasters. Die Operatoren auf $\mathbb{Z}^d$ seien denen aus $\mathbb{R}^d$ entsprechend. Der Mittelpunkt $center(c)$ einer Zelle mit den Koordinaten $c \in \mathbb{Z}^d$ im euklidischen Raum ist

$$center \colon \mathbb{Z}^d \to \mathbb{R}^d,$$
$$center(c) = w \cdot c$$

Die Menge $box(c)$, die den Inhalt einer Zelle im euklidischen Raum angibt, ist der Inhalt der Box B_c, die durch die Koordinaten $v_{low} = center(c) - 1_d \cdot w$ und $v_{high} = center(c) + 1_d \cdot w$ eingeschlossen ist. Wobei der Vektor $1_d \in \mathbb{R}^d$ in jeder Koordinate 1 ist. Entgegen der Definition einer Box seien aber die Seitenflächen von B_c aus $box(c)$ ausgeschlossen, die v_{low} enthalten. Formal ausgedrückt bedeutet dies

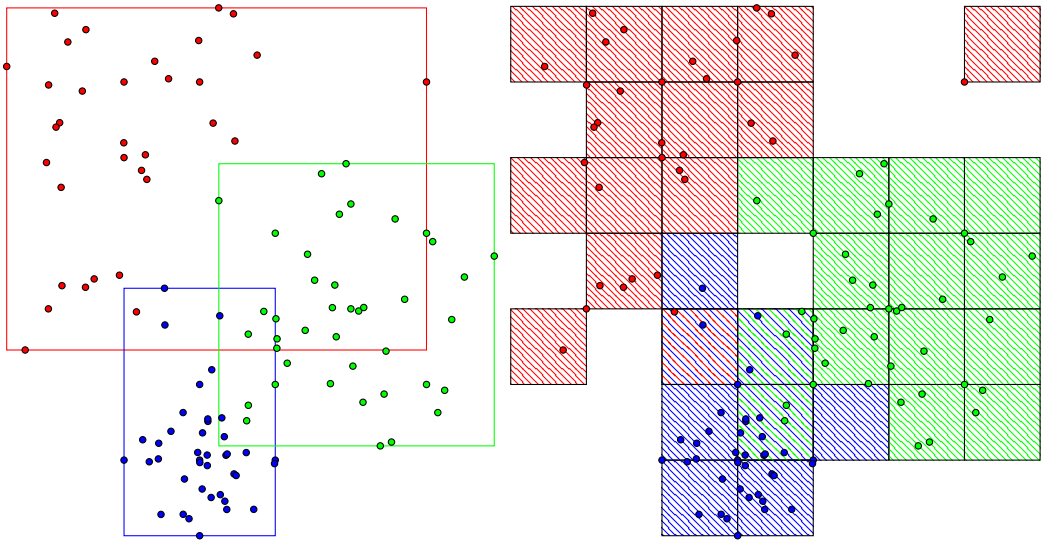
$$box(c) = B_c \setminus M$$
$$\text{mit } M = \{ p \in \mathbb{R}^d \mid \exists_{i \in \{ 1, \dots, d \}} \colon p \cdot e_i = v_{low} \cdot e_i \}.$$

Ohne diese Anpassung wären die Inhalte zweier benachbarter Zellen nicht disjunkt. Mehrdeutigkeiten wären die Folge.

Um sich das Raster während der Triangulierung in den $intersects_i$-Funktionen zu nutze machen zu können, müssen die Partitionen im GENERATEBOUNDS-Schritt vorbereitet werden. Jeder Partition Q_i werden die Zellen zugewiesen, die einen ihrer Punkte $p \in Q_i$ enthalten. Zusätzlich zur Punktmenge Q_i gehört also auch die Menge der Zellen $C_i \subset \mathbb{Z}^d$, die die Partition begrenzen zur Partition. Es gilt

$$C_i = \{ c \in \mathbb{Z}^d \mid \exists_{p \in Q_i} \colon p \in box(c) \}.$$

Die Anzahl der Elemente in C_i ist endlich, da für jedes $p \in \mathbb{R}^d$ genau eine Zelle $c \in \mathbb{Z}^d$ mit $p \in box(c)$ existiert und Q_i endlich ist. Mit einer effizient implementierten Funktion, die für $c \in \mathbb{Z}^d$ und $p \in \mathbb{R}^d$ überprüft, ob $p \in box(c)$, lässt sich auch C_i effizient bestimmen.



(a) (b)

Abbildung 3.6.: Begrenzungen durch Boxen (a) und Raster (b): Den Partitionen sind die Zellen entsprechend ihrer Farben zugeordnet. Manche Zellen enthalten Punkte unterschiedlicher Partitionen und sind somit mehreren Partitionen zugeordnet. Die gezeigte Partitionierung der Punktwolke wurde durch Zuweisung zum nächsten Sample gewonnen.

Nach diesen Vorbereitungen im GENERATEBOUNDS-Schritt kann für die Triangulierung eine neue $intersects_i(v, r)$-Funktion für jedes $i \in \{1, \ldots, p\}$ durch

$$intersects_i(v, r) = \begin{cases} true, & \text{falls } \exists_{i \in \{1,\ldots,p\}} : C_i \cap S \\ false, & \text{sonst} \end{cases}$$

$$\text{mit } S = \{ x \in \mathbb{R}^d \mid \|x - v\| \leq r \}$$

definiert werden.

Die Implementierung untersucht für jede Zelle in C_i, ob sie die Sphäre schneidet. Der Schnitt mit der Sphäre ist jedoch nicht exakt. Zu Gunsten der Laufzeit wird tatsächlich nur der Schnitt mit dem Umkreis der Zelle geprüft. Dies ist eine, wie bereits erläutert, mögliche Vergröberung der $intersects_i$-Funktionen.

Abbildung 3.6 veranschaulicht die Festlegung der Partitionsgrenzen durch Boxen und ein Raster. Die Sphären-Schnitte mit dem Raster zu berechnen, hat zwei Vorteile. Zum einen ist es die feinere Methode, wodurch weniger Punkte in den Rand fallen. Und zum anderen bietet sie mit der Zellenbreite w einen weiteren Parameter, mit dem die Feinheit der $intersects_i$-Funktionen justiert und zwischen Größe des Randes und Aufwand der Schnittprüfung abgewogen werden kann.

Algorithm 8 Zuweisung der Zellen zu Partitionen

procedure GENERATEBOUNDS($Q = \{ Q_1, \ldots, Q_p \}$)
  $C_1, \ldots, C_p \leftarrow \emptyset$
  for all $Q_i \in Q$ **do**
    for all $p \in Q_i$ **do**
      Finde Zelle c mit $p \in box(c)$
      $C_i \leftarrow C_i \cup \{ c \}$
  return $C_1, \ldots, C_p$

4 | Evaluation

Die in diesem Kapitel vorgestellten Ergebnisse wurden mit folgender Hardware ermittelt:

Architektur	x86_64
CPUs	2
Cores pro CPU	12 (+12 Hyperthreading)
Taktfrequenz	2300 MHz
CPU-Modell	Intel® Xeon® CPU E5-2670 v3 @ 2.30GHz
L1d-Cache	32 kB
L1i-Cache	32 kB
L2-Cache	256 kB
L3-Cache	30720 kB
Arbeitsspeicher	ca. 126 GB

Die folgende Software wurde eingesetzt:

Kernel	Linux version 3.13.0-115-generic (gcc version 3.4.1 20040714)
Betriebssystem	Ubuntu 4.8.4-2ubuntu1 14.04.3
Programmiersprache	C++ (C++17)
Compiler	GCC 7.2.0

Folgende Bibliotheken finden Anwendung:

- Boost 1.61 [16]
- TBB: Intel® Threading Building Blocks 2018 Update [17]
- CGAL: Computational Geometry Algorithms Library 4.12-I-900 [18]
- KaHIP: Karlsruhe High Quality Partitioning v2.0 [19] [13]
- libkdtree++ 0.7.1 [20]

Wie in der Veröffentlichung von Funke und Sanders, wird für die sequentielle Triangulierung CGAL und für die Parallelisierung TBB eingesetzt. Für die sequentielle Triangulierung wird, wie auch in der Veröffentlichung von Funke und Sanders, CGAL eingesetzt.

Die für den Ansatz der Zuweisung zum nächsten Samplepunkt nötige Nächste-Nachbar-Suche für alle Punkte $p \in P$ in allen Samplepunkten P_s wird effizient durch einen k-d-Baum

implementiert. Der k-d-Baum enthält einen Knoten für jeden Samplepunkt $p \in P_s$ und jeweils den Index $i \in \{1, \dots p\}$ der entsprechenden Samplepartition $Q_{s,i} \ni p$. In dieser Arbeit wird dazu „libkdtree++“ verwendet.

Um den durch diese Arbeit gemachten Fortschritt nachzuweisen, wird die Leistung der eingeführten Strategien mit einem Partitionierungsansatz verglichen, der bereits von Funke und Sanders [5] angewandt wurde. Das im Folgenden als „Cycle-Partitionierer“ bezeichnete Vorgehen partitioniert die Punktwolke rekursiv. Zu Beginn jeder Rekursion wird zyklisch eine neue Dimension gewählt. In dieser wird die Punktwolke anhand ihres Medians geteilt und die Rekursion jeweils mit den Teilen fortgesetzt. Die Rekursion endet, sobald eine bestimmte Rekursionstiefe erreicht wurde oder die Anzahl der Punkte eine Schranke unterschreitet. [5] Insbesondere entspricht die Anzahl der Partitionen beim Cycle-Partitionierer nicht notwendigerweise der Anzahl der Threads.

Die eingegebenen Punktwolken sämtlicher in diesem Kapitel vorgestellten Experimente wurden automatisch generiert. Sie sind aufgebaut aus mehreren kreis- bzw. kugelförmigen Punktclustern (Bubbles), die einander nicht überlappen.

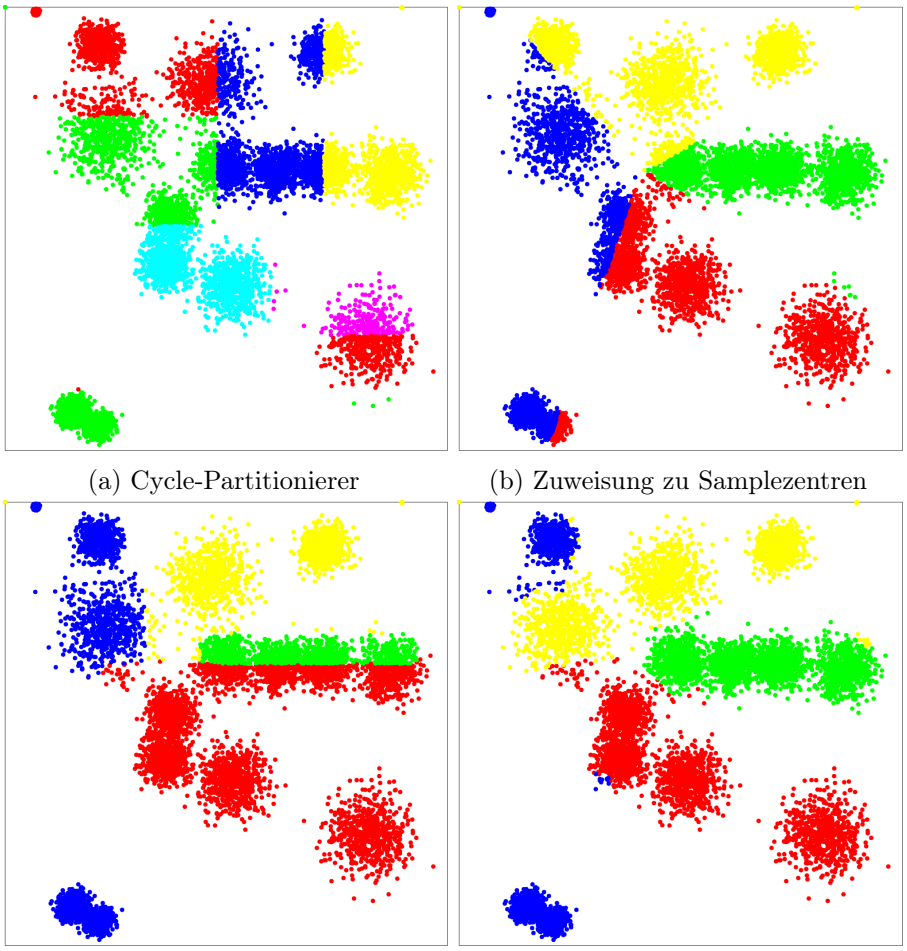
Zur Generierung werden 4^d Punkte, die als Zentren der Bubbles dienen, gleichverteilt in einer gegebenen d-Box generiert. Anschließend werden um jedes Zentrum $z = (z_1, \dots, z_d)^T$ etwa gleich viele Punkte generiert. Die Koordinate g_i ($i \in \{1, \dots, d\}$) eines Punktes $g = (g_1, \dots, g_d)$ um das Zentrum z wird dabei jeweils zufällig aus einer Normalverteilung mit Median z_i und Varianz r gewählt. Die Varianz r ist dabei die Hälfte der Strecke von z zum nächsten Zentrum oder zum Rand der Box, falls dieser näher ist. Der Pseudocode verdeutlicht das Vorgehen.

Algorithm 9 Generierung einer Punktwolke

```
procedure GENERATEPOINTS(n, boundingBox)
    centers ← ∅
    for i = 1 upto 4^d do
        c ← uniformlyDistributedRandomPointIn(boundingBox)
        centers ← centers ∪ { c }
    P ← ∅
    for all c ∈ centers do
        for i = 1 upto min(⌈n / 4^d⌉, |P| − n) do
            boundsDistance ← dist(c, ℝ^d \ boundingBox)
            nearestNeighbourDistance ← dist(c, centers \ { c })
            r ← (1/2) min(boundsDistance, nearestNeighbourDistance)
            p ← normalDistributedRandomPoint(c, r)
            P ← P ∪ { p }
    return P
```

Obwohl Abbildung 4.1a aufgrund der geringen Anzahl der Punkte nicht repräsentativ für Cycle-Partitionierer sein kann, zeigt sie, welchen Nachteil er beim Partitionieren einer, wie beschrieben, generierten Punktwolke birgt. Die Bubbles werden oft in zwei unterschiedliche Partitionen aufgeteilt, was bei der Triangulierung zu einem größeren Rand

führt. Die anderen Partitionierungsstrategien aus 4.1 nutzen die Sampletriangulierung, um dies möglichst zu vermeiden.



(a) Cycle-Partitionierer

(b) Zuweisung zu Samplezentren

(c) Zuweisung zu Samplepartitionsboxen

(d) Zuweisung zu nächstem Samplepunkt

Abbildung 4.1.: Partitionierungen einer generierten zweidimensionalen Punktwolke mit 10000 Punkten und 1000 Samplepunkten

In Kapitel 3 wurden bereits die Parameter besprochen, mit denen die Partitionierung variiert werden kann. Hier werden diese wieder aufgegriffen und ihre Auswirkung auf die Qualität der Partitionierung getestet.

Partitionierer p**:** Einer aus: Zuweisung zu Samplezentren (Z) aus Abschnitt 3.2, Zuweisung zu Samplepartitionsbegrenzung mit Boxen (B) aus Abschnitt 3.3, Zuweisung zum nächsten Samplepunkt (S) aus Abschnitt 3.4 oder der Cycle-Partitionierer (C) von Funke und Sanders

Dimension d: Die Dimension der eingegebenen Punktwolke
Größe der Punktwolke n: Anzahl der Punkte in der eingegebenen Punktwolke
Anzahl der Threads t: Anzahl der Threads, auf denen die Triangulierung durchgeführt wird
Anzahl der Samplepunkte s: Anzahl der Samplepunkte aus denen die Sampletriangulierung gebildet wird (beschrieben in Abschnitt 3.1)
Kantengewichte w: Gewichte der Kanten (konstant, linear oder quadratisch)
Breite der Rasterzellen c: Parameter, der das Raster definiert (beschrieben in Abschnitt 3.5)

In den folgenden Abschnitten werden neben der Gesamtlaufzeit auch die Laufzeit der Partitionierung und der Triangulierung separat sowie die Balance der Größe der Partitionen und die Größe des Randes untersucht.

Abschnitt 4.1 ermittelt eine günstige Anzahl an Threads für die nachfolgenden Untersuchungen, die brauchbare Werte für die verbleibenden Parameter finden. Das Verhalten der Gesamtlaufzeit und der Qualität der Partitionierung wird in Abhängigkeit der Größe n der Punktwolke betrachtet, um schließlich die Erfolge dieser Arbeit zusammenzufassen.

4.1. Anzahl der Threads

Der Speedup $S(p)$ für die Anzahl p der Threads wird folgendermaßen bestimmt:

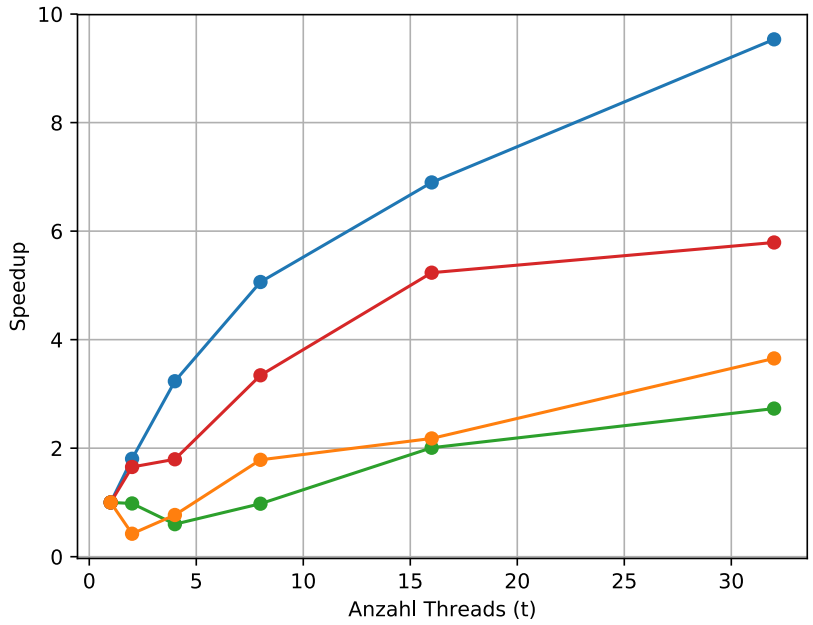
$$S_s(p) = \frac{T_s(1)}{T_s(p)},$$

mit s: Zuweisungsstrategie
und $T_s(p)$: Gesamtaufzeit mit Zuweisungsstrategie s und p Threads

Die Anzahl der Threads entspricht der Anzahl der Partitionen. Daher führt eine größere Parallelität zu einer größeren Anzahl an Punkten, die als Teil des Randes ein zweites Mal trianguliert werden müssen. Somit ist der Speedup stets sublinear.

Abbildung 4.2 zeigt den Speedup, der mit den drei Zuweisungsstrategien erreicht wird. Zum Vergleich ist auch der Speedup mit dem Cycle-Partitionierer dargestellt. Zu sehen ist, dass vor allem die Zuweisung zum nächstem Samplepunkt (S) nahe an den Speedup des Cycle-Partitionierers herankommt, während die beiden anderen Zuweisungsstrategien auch bei einer hohen Parallelität nur einen Speedup von ca. 3 gegenüber Verwendung von nur einem Thread haben. Der Speedup flacht insbesondere ab 16 Threads ab, was den bereits erläuterten sublinearen Speedup bestätigt. Es ist jedoch auch zu beachten, dass insgesamt nur 24 echte CPU-Cores zur Verfügung stehen und ein Abflachen des Speedups auch auf Intels Hyperthreading-Technologie zurückzuführen ist.

Eine weitere Beobachtung, die aus dem Speedup-Plot hervorgeht, ist, dass die Zuweisung zu Samplepartitionszentren (Z) für eine geringe Anzahl an Partitionen eine ausgesprochen



-	●	●	●	●
p	C	Z	B	S
d	3			
n	6666666			
t	x			
s	-	$\frac{2}{100}n$		
w	-	konstant		
c	-	1		

C: Cycle-Partitioner; **Z**: Zuweisung zu Samplezentrum; **B**: Zuweisung zu Samplebegrenzung; **S**: Zuweisung zu nächstem Samplepunkt

Abbildung 4.2.: Speedup mit Samplegröße $\frac{2}{100}n \approx 133333$

schlechte Partitionierung produziert. Es ist zu beachten, dass die Anzahl t der Partitionen um ein Vielfaches durch die Anzahl $4^d = 4^3 = 64$ der Bubbles in der Punktwolke überstiegen wird. Alle Punkte der Punktwolke werden also dem nächsten der t Zentren zugewiesen. Die Anzahl der Bubbles in der Punktwolke ist 64, während alle Punkte der Punktwolke dem nächsten von nur t Samplepartitionszentren zugewiesen werden. Somit erfasst die Partitionierung mit zunehmendem t die Verteilung der Punktwolke immer besser, was jedoch für geringe t nicht gelingen kann.

Die Probleme mit der Zuweisung zu Samplepartitionsboxen wurden bereits in Abschnitt 3.3 besprochen und bestätigen sich in diesen und den folgenden Plots.

Dieser Abschnitt hat ersten Aufschluss darüber geliefert, welcher Ansatz am vielversprechendsten ist. Die Zuweisung zum nächsten Samplepunkt trägt als einzige der drei Strategien zu einem Delaunay-Triangulierungs-Algorithmus bei, der ähnlich gut wie der bereits von Funke und Sanders implementierte Algorithmus mit dem Cycle-Partitionierer skaliert. Außerdem konnte in Erfahrung gebracht werden, dass $t = 16$ Threads eine günstige Wahl für die in den folgenden Abschnitten erläuterten Untersuchungen ist.

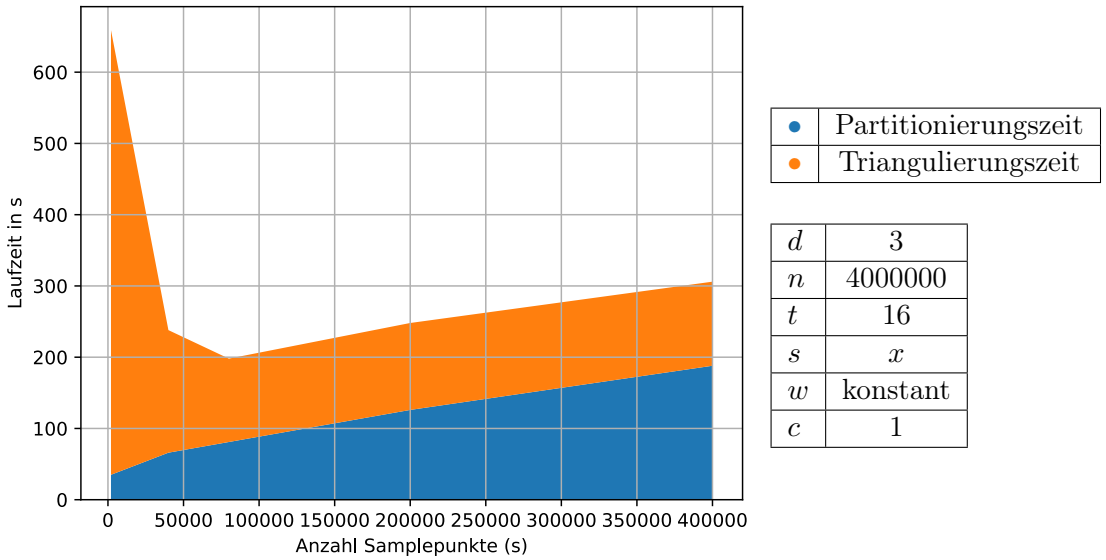
Welche Anzahl an Samplepunkten ideal ist, wird im nächsten Abschnitt untersucht.

4.2. Anzahl der Samplepunkte

Mit der Anzahl der Samplepunkte wächst der Grad der Genauigkeit, mit der die Verteilung der Punktwolke durch die Samplepunkte angenähert wird. Durch das Wissen um die Verteilung kann eine Partitionierung gefunden werden, die sowohl besser balanciert ist als auch

eine kleinere Border-Triangulierung hervorbringt. Jedoch bedeutet eine größere Anzahl an Samplepunkten nicht nur einen geringeren Parallelisierungsoverhead sondern auch einen größeren Aufwand die Triangulierung durch den eingesetzten Partitionierungsalgorithmus vorzubereiten.

Mit Abbildung 4.3 kann ermittelt werden, mit welcher Samplegröße eine ausgewogene Wahl zwischen Vorbereitungs- und Partitionierungsoverhead gemacht werden kann.



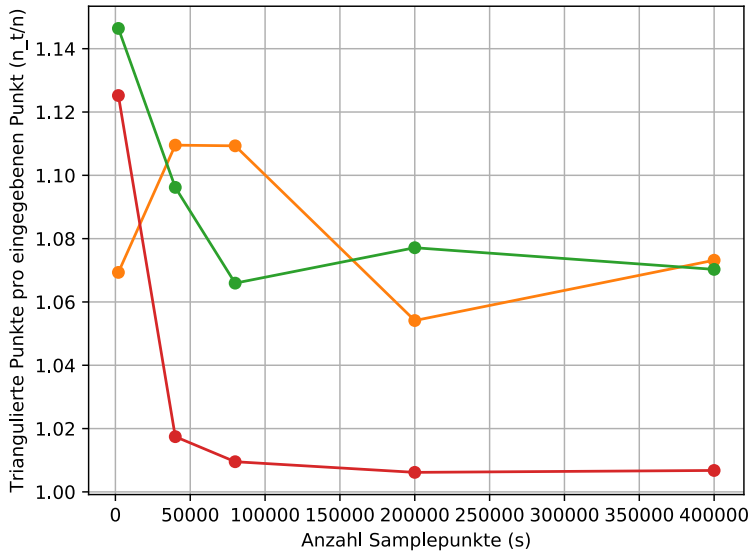
d	3
n	4000000
t	16
s	x
w	konstant
c	1

Abbildung 4.3.: Partitionierungs- und Triangulierungszeit bei Zuweisung zum nächsten Samplepunkt. Aufgetragen über die Anzahl der Samplepunkte: $\sqrt{n} = 2000$, $\frac{1}{100}n = 40000$, $\frac{2}{100}n = 80000$, $\frac{5}{100}n = 200000$ und $\frac{10}{100}n = 400000$

Gezeigt ist die Laufzeit als Funktion der Anzahl der Samplepunkte für eine feste Anzahl n an Punkten. Da die Laufzeit die Summe der Partitionierungs- und der Triangulierungszeit ist, bietet sich ihre Darstellung in gestapelter Form an. Verwendet wird die Zuweisung zum nächsten Samplepunkt (S).

Die Partitionierungszeit wächst annähernd linear mit der Anzahl der Samplepunkte, die auch ihre Eingabegröße ist. Die Triangulierungszeit hat ihr Maximum bei der minimalen Anzahl der Samplepunkte von $s = \sqrt{n}$ und fällt rasch mit wachsender Samplegröße. Von einer Steigerung der Samplegröße über $s = \frac{2}{100}n$ profitiert die Triangulierungszeit kaum. Diese untere Schranke wird durch die Abbildungen 4.4 und 4.5 verständlich.

Insgesamt zeugt die Darstellung von der nötigen Abwägung zwischen kürzerer Partitio nierungs- oder kürzerer Triangulierungszeit. In den folgenden Abschnitten des Kapitels wird die Anzahl der Samplepunkte im Sinne dieses Diagramms auf $s = \frac{2}{100}n$ festgesetzt.



-	●	●	●
p	Z	B	S
d	3		
n	4000000		
t	16		
s	x		
w	konstant		
c	1		

Z: Zuweisung zu Samplezentrum; **B**: Zuweisung zu Samplebegrenzung; **S**: Zuweisung zu nächstem Samplepunkt

Abbildung 4.4.: Anzahl der triangulierten Punkte pro Punkt der Punktwolke aufgetragen über die Anzahl der Samplepunkte: $\sqrt{n} = 2000$, $\frac{1}{100}n = 40000$, $\frac{2}{100}n = 80000$, $\frac{5}{100}n = 200000$ und $\frac{10}{100}n = 400000$

Abbildung 4.4 zeigt das Verhältnis der triangulierten Punkte zur Gesamtzahl der Punkte als Funktion der Anzahl der Samplepunkte. Jedes Mal, wenn ein Punkt Teil der Eingabe des sequentiellen Triangulierungsalgorithmus ist, wird er im Sinne dieser Untersuchung trianguliert. Für eine korrekte Delaunay-Triangulierung muss jeder Punkt mindestens einmal trianguliert werden. Für jeden Punkt der Punktwolke, die mit dem Algorithmus aus Abschnitt 2.4 trianguliert wird, gilt:

- Er wird höchstens zweimal trianguliert.
- Er wird genau dann zweimal trianguliert, wenn er Teil des Randes ist.

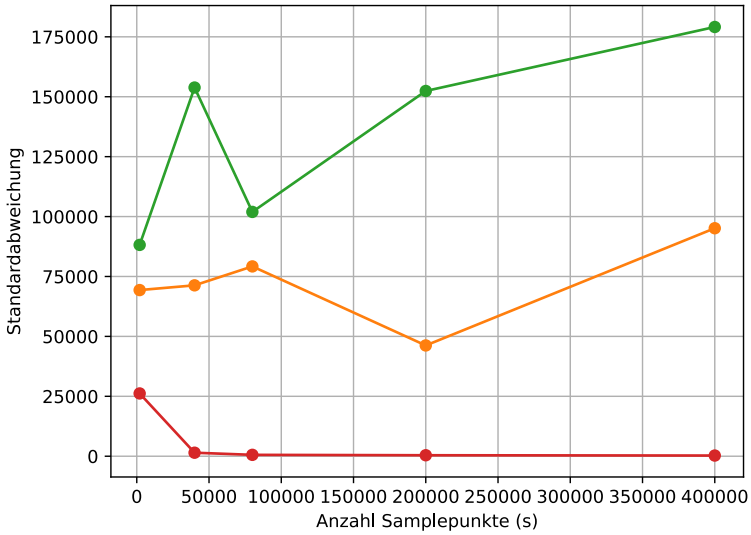
Es folgt:

- Das Verhältnis V der triangulierten Punkte zur Gesamtzahl der Punkte liegt stets zwischen 1 und 2.
- Die Größe des Randes ist $n \cdot (V - 1)$.

Für den Algorithmus von Funke und Sanders treffen diese Eigenschaften aufgrund des Teile-und-herrsche-Ansatzes nicht zu, da ein Punkt Teil verschiedener Ränder sein kann.

Das Verhältnis V ist eine Größe, die den Parallelisierungsoverhead angibt. V möglichst gering zu halten ist eine Aufgabe des Load-Balancings.

Mit Abbildung 4.4 kann der Zuweisung zum nächsten Samplepunkt (S) ein besonders vorteilhaftes Verhältnis der triangulierten Punkte zur Gesamtzahl der Punkte bereits für $s = \frac{1}{100}n$ Samplepunkte nachgewiesen werden. Die anderen beiden Ansätze weisen keinen so deutlichen Verlauf auf.



-	●	●	●
p	Z	B	S
d	3		
n	4000000		
t	16		
s	x		
w	konstant		
c	1		

Z: Zuweisung zu Samplezentrum; **B**: Zuweisung zu Samplebegrenzung; **S**: Zuweisung zu nächstem Samplepunkt

Abbildung 4.5.: Standardabweichung der Größe der Partitionen aufgetragen über die Anzahl der Samplepunkte: $\sqrt{n} = 2000$, $\frac{1}{100}n = 40000$, $\frac{2}{100}n = 80000$, $\frac{5}{100}n = 200000$ und $\frac{10}{100}n = 400000$

Abbildung 4.5 zeigt die Standardabweichung $s_{|Q|}$ der Partitionsgrößen, die für die Partitionierung Q durch

$$s_{|Q|} = \sqrt{\frac{1}{n-1} \sum_{Q_0 \in Q} (|Q_0| - \bar{x}_{|Q|})^2}$$
$$\text{mit } \bar{x}_{|Q|} = \frac{1}{n} \sum_{Q_0 \in Q} |Q_0|$$

berechnet wird.

Ein geringes $s_{|Q|}$ spricht dafür, dass die Partitionierung gleichmäßig ist. Im Sinne des Load-Balancings bedeutet dies, dass die Last gleichmäßig auf die Threads verteilt wird. Die Standardabweichung der Partitionsgrößen ist also eine weitere wichtige Größe zur Bewertung der Qualität der Partitionierung.

Abbildung 4.5 zeigt vor allem für die Zuweisung zum nächsten Samplepunkt (S) einen ausgesprochen günstigen Verlauf.

Für eine Samplegröße von $s = \frac{2}{100}$ und eine Punktwolke mit $n = 4000000$ entsteht zum Beispiel die folgende sortierte Liste der Partitionsgrößen:

1	2	3	4	5	6	7	8
248870	249236	249515	249576	249815	250008	250008	250030
9	10	11	12	13	14	15	16
250032	250048	250070	250119	250218	250416	250875	251292

Die größte Partition enthält nur 1% mehr Punkte als die kleinste. Die Standardabweichung beträgt $s_{|Q|} = 576{,}6$.

Die anderen beiden Ansätze weisen keinen derart positiven Verlauf auf. Auf die Zuweisung zu Samplezentren (Z) hat eine größere Anzahl an Samplepunkten keinen positiven sondern sogar negativen Effekt. Dies ist auch in Kurven für andere n beobachtbar und lässt die Interpretation zu, dass für die Annäherung an die Verteilung der Punkte eine größere Anzahl an Samplepunkte hinderlich ist. Dieselbe Beobachtung kann in noch extremerem Ausmaß auch für die Zuweisung zu Samplepartitionsboxen (B) gemacht werden.

Eine mögliche Erklärung ist, dass durch größere Samples auch die Wahl von ungünstigen Samplepunkten, die für jede Samplepartition einen Ausreißer darstellen, wahrscheinlicher wird. Der Einfluss eines Ausreißers auf die d-Box, die eine Samplepartition umschließt, ist enorm. Ferner sind die außen liegenden Punkte sogar die einzigen, die einen Einfluss auf die Ausmaße der Box haben. Auf das arithmetische Mittel der Samplepartition, das dem Samplezentrum entspricht, hat ein Ausreißer immerhin denselben Einfluss wie jeder andere Samplepunkt.

Für die meisten Punkte der Punktwolke ist es hingegen egal, welcher Samplepartition der Ausreißer zugeordnet wird, da für die meisten Punkte der Ausreißer nicht den nächsten Samplepunkt darstellt. Der Einfluss eines Samplepunkts ist bei der Zuweisung zum nächsten Samplepunkt nur lokal.

Somit hat dieser Abschnitt neben der Erkenntnis, dass $s = \frac{2}{100}n$ eine gute Wahl für die Samplegröße ist, auch Erklärungen für die Qualitätsunterschiede zwischen den in dieser Arbeit eingeführten Ansätzen geliefert.

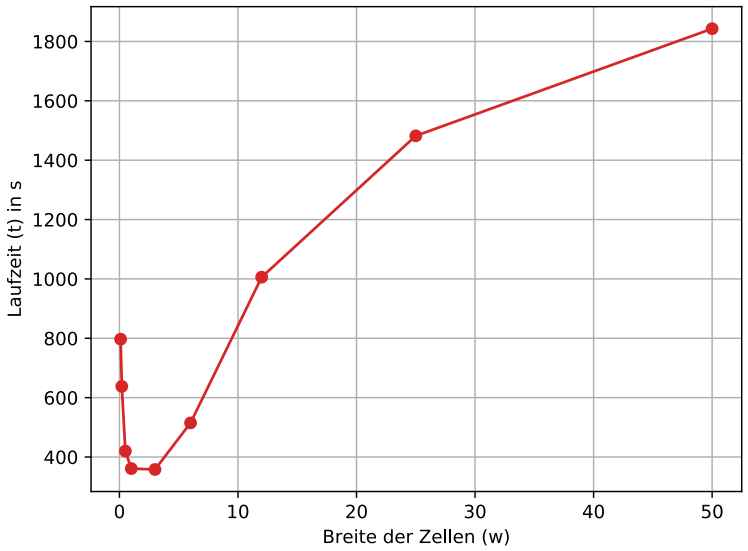
4.3. Breite der Rasterzellen

Die Breite der Rasterzellen ist für alle anderen Untersuchungen dieses Kapitels konstant auf $c = 1$ gesetzt. Die Punktwolke liegt in einer 3-Box der Größe $100 \times 100 \times 100$. Somit wird der Raum durch ein Raster mit Zellenbreite c in etwa $\frac{10^6}{c^3}$ gleiche Boxen geteilt. Je kleiner c ist, desto feiner ist also das Raster und ein umso kleinerer Rand ist zu erwarten. In diesem Abschnitt wird dargestellt, welchen Beitrag sie zur Leistungsfähigkeit der präsentierten Ansätze geben.

Abbildung 4.6 weist diesen Beitrag für die Zuweisung zum nächsten Samplepunkt (S) nach. Eine global minimale Laufzeit wird mit $c = 1$ oder $c = 3$ erreicht.

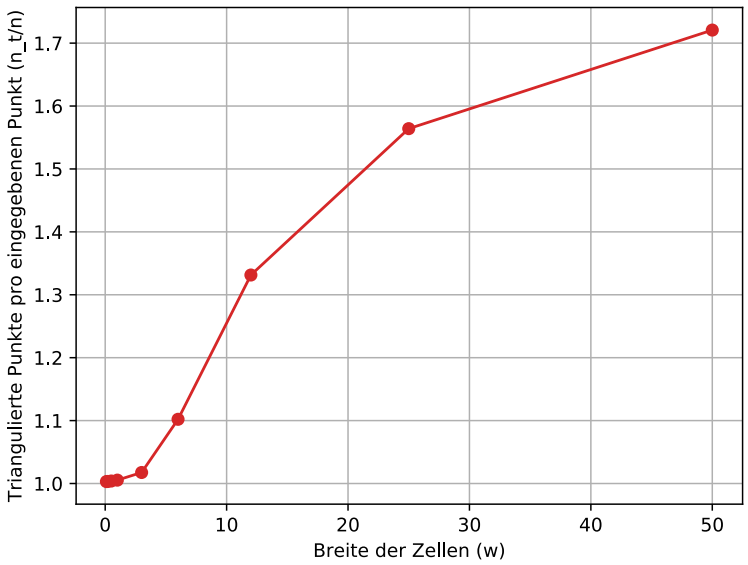
Das in Abschnitt 3.5 eingeführte Raster bringt also einen Laufzeitvorteil. Ein zu feines Raster steigert jedoch die Zeit, die für jede Berechnung von *intersects* nötig ist so stark, dass sich der Vorteil durch den verkleinerten Rand nicht auszahlt.

Abbildung 4.7 zeigt, dass die Größe des Randes durch ein feineres Raster (mit kleinem c) verringert wird, wie in Abschnitt 3.5 angekündigt.



-	●
p	S
d	3
n	7000000
t	16
s	$\frac{2}{100}n$
w	konstant
c	x

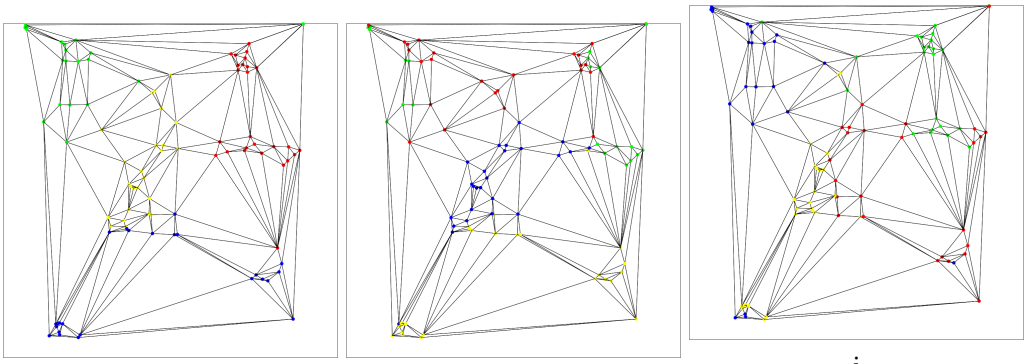
Abbildung 4.6.: Gesamtlaufzeit bei Zuweisung zum nächsten Samplepunkt aufgetragen über die Breite der Rasterzellen: 0,1, 0,2, 0,5, 1, 3, 6, 12, 25, 50



-	●
p	S
d	3
n	7000000
t	16
s	$\frac{2}{100}n$
w	konstant
c	x

Abbildung 4.7.: Anzahl der triangulierten Punkte pro Punkt der Punktwolke bei Zuweisung zu Samplepunkten aufgetragen über die Breite der Rasterzellen: 0,1, 0,2, 0,5, 1, 3, 6, 12, 25, 50

4.4. Kantengewichte



(a) $w = konstant$ (b) $w = linear$ i (c) $w = quadratisch$

Abbildung 4.8.: Partitionierte Sampletriangulierung einer generierten zweidimensionalen Punktwolke mit 10000 Punkten und 100 Samplepunkten

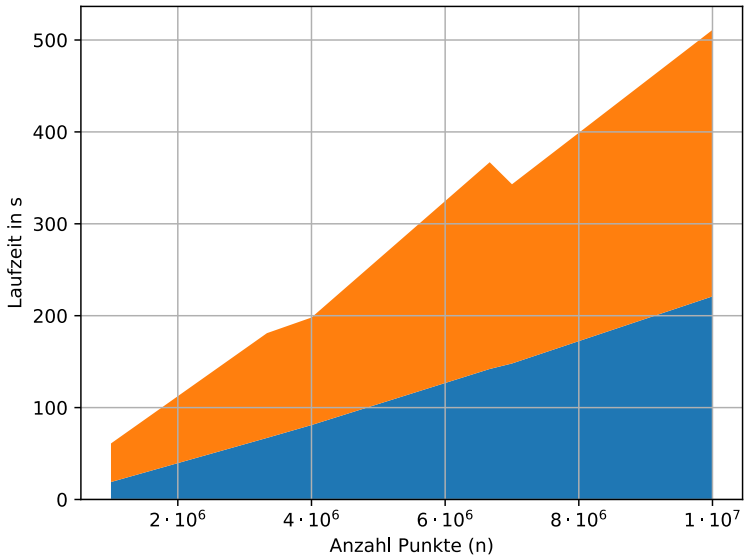
Die Kantengewichte wurden in Abschnitt 3.1 als mögliche Option zur Verbesserung der Qualität der Graphpartitionierung der Sampletriangulierung vorgestellt. Ziel ist es, dass Samplepunkte, die zur selben Bubble gehören, auch derselben Graphpartition zugeordnet werden. Abbildung 4.8a zeigt in einem zweidimensionalen Beispiel, dass mit konstanten Kantengewichten auch einige nah beieinander liegende Samplepunkte nicht derselben Partition zugeordnet werden.

Weder lineare noch quadratische Kantengewichte können dieses Problem beheben. Entgegen der Erwartung verschlechtern sie die Graphpartitionierung sogar. Da schon diese visuelle Untersuchung die Überlegenheit der konstanten Kantengewichte bescheinigt, werden diesem Parameter keine Laufzeitexperimente gewidmet.

4.5. Größe der Punktwolke

Während sich die vorherigen Abschnitte mit dem Finden optimaler Parameter bschäftigt, wird in diesem Abschnitt das Laufzeitverhalten als Funktion der Größe der eingegebenen Punktwolke untersucht. Die Parameter sind entsprechend der Ergebnisse der vorherigen Abschnitte gewählt worden: Die Anzahl der Threads ist $t = 16$ und die Samplegröße ist $s = \frac{2}{100}n$.

Abbildung 4.9 zeigt die Laufzeit als Funktion der Größe der Punktwolke. Die Laufzeit ist als gestapelter Plot der Partitionierungs- und der Triangulierungszeit dargestellt. Es



d	3
n	x
t	16
s	$\frac{2}{100}n$
w	konstant
c	1

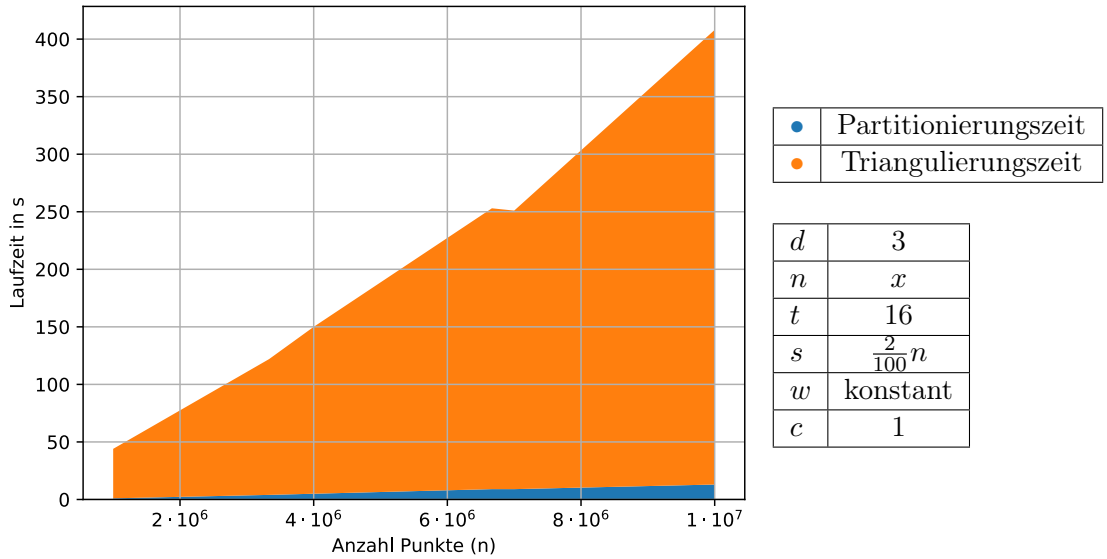
Abbildung 4.9.: Partitionierungs- und Triangulierungszeit bei Zuweisung zu Samplepunkten aufgetragen über die Größe der Punktwolke: 1000000, 3333333, 4000000, 6666666, 7000000 und 10000000 Punkte

ist zu sehen, dass sowohl die Laufzeit der Partitionierung als auch der Triangulierung linear in der Eingabegröße n und in etwa gleich sind.

Letzteres gilt nicht für den Cycle-Partitionierer, wie Abbildung 4.10 zeigt. Die Triangulierungszeit überschreitet die Partitionierungszeit um weit mehr als das Zehnfache.

Abbildung 4.11 stellt die Gesamtlaufzeit der drei Zuweisungsstrategien und des Cycle-Partitionierers im Vergleich dar. Während die anderen Zuweisungsstrategien weit darüber liegen, bietet die Zuweisung zu Samplepunkten (S) eine Laufzeit, die der des Cycle-Partitionierers sehr ähnlich ist.

Abbildung 4.12 belegt einen der Vorteile der Zuweisung zum nächsten Samplepunkt gegenüber dem Cycle-Partitionierer. Dargestellt ist das Verhältnis triangulierter Punkte zur Gesamtzahl der Punkte als Funktion der Größe der Punktwolke. Mit einem besonders günstigen Verhältnis sticht die Zuweisung zum nächsten Samplepunkt heraus. Der Ansatz liefert unabhängig von der Größe der eingegebenen Punktwolke eine wesentlich bessere Partitionierung als alle anderen Partitionierungsstrategien. Für die beiden anderen Strategien, die auf Sampletriangulierungen basieren, ist das Verhältnis etwa mit dem Triangulierungsalgorithmus, der in seiner Rekursion Punkte teils mehrfach trianguliert, vergleichbar.

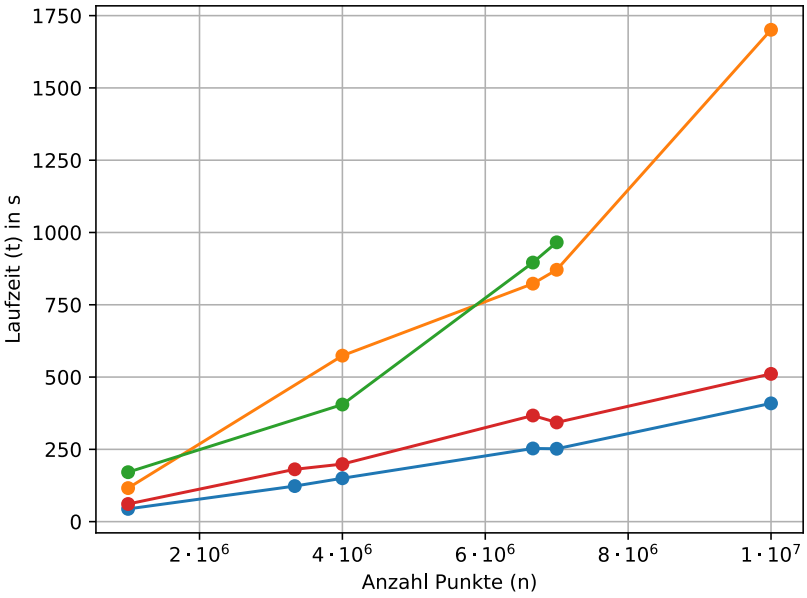


d	3
n	x
t	16
s	$\frac{2}{100}n$
w	konstant
c	1

Abbildung 4.10.: Partitionierungs- und Triangulierungszeit mit dem Cycle-Partitionierer aufgetragen über die Größe der Punktwolke: 1000000, 3333333, 4000000 und 6666666 Punkte

Die Standardabweichung der Partitionsgrößen als Funktion der Anzahl der Punkte wird in Abbildng 4.13 dargestellt. Im Gegensatz zu Abbildung 4.12 schneiden die Zuweisung zu Samplepartionszentren (Z) und die Zuweisung zu Samplepartitionsboxen (B) im Vergleich zum Cycle-Partitionierer schlechter ab. Hierbei ist jedoch zu beachten, dass, wie in Abschnitt 4.2 erläutert, geringere Samplegrößen bei diesen Ansätzen zu besseren Ergebnissen hinsichtlich der Standardabweichung führen.

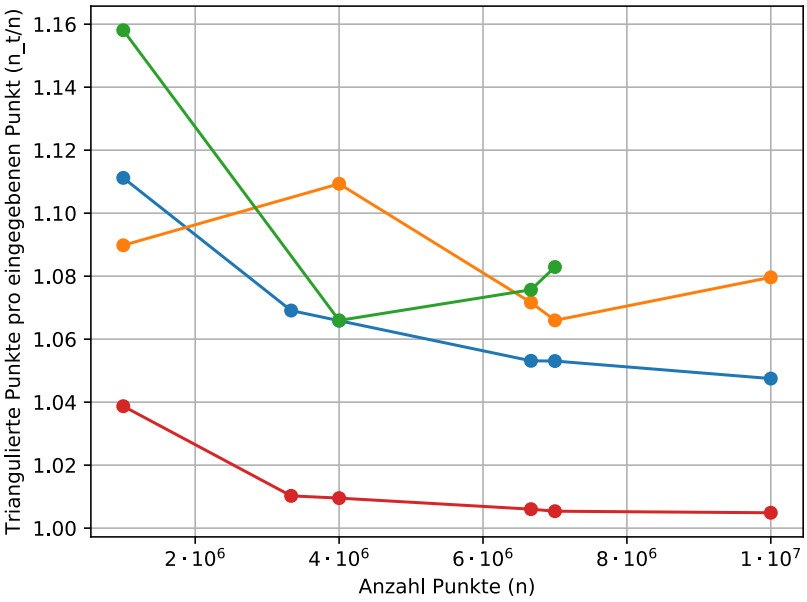
Unabhängig von der Samplegröße und der Anzahl der Punkte liefert die Zuweisung zum nächsten Samplepunkt eine nahezu perfekte Partitionierung.



-	●	●	●	●
p	C	Z	B	S
d	3			
n	x			
t	16			
s	-	$\frac{2}{100}n$		
w	-	konstant		
c	-	1		

C: Cycle-Partitioner; **Z**: Zuweisung zu Samplezentrum; **B**: Zuweisung zu Samplebegrenzung; **S**: Zuweisung zu nächstem Samplepunkt

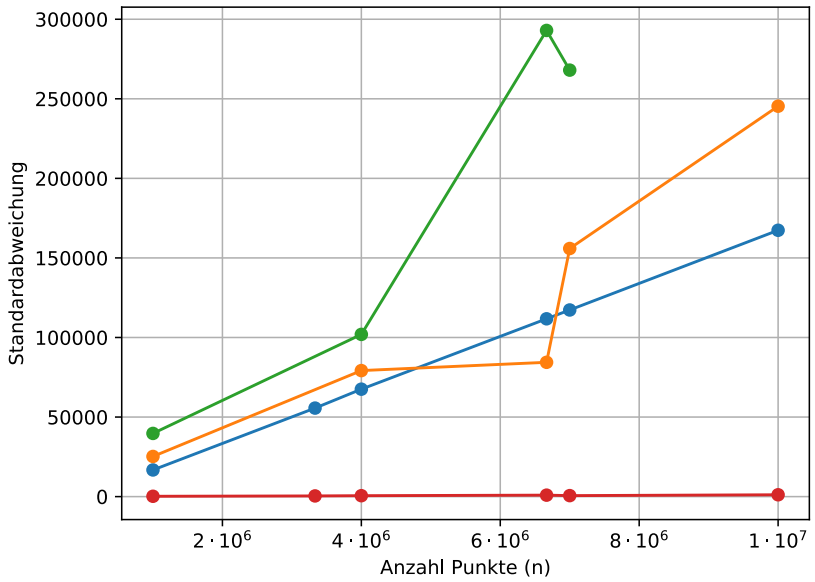
Abbildung 4.11.: Gesamtlaufzeit aufgetragen über die Größe der Punktwolke: 1000000, 3333333, 4000000, 6666666, 7000000 und 10000000 Punkte



-	●	●	●	●
p	C	Z	B	S
d	3			
n	x			
t	16			
s	-	$\frac{2}{100}n$		
w	-	konstant		
c	-	1		

C: Cycle-Partitioner; **Z**: Zuweisung zu Samplezentrum; **B**: Zuweisung zu Samplebegrenzung; **S**: Zuweisung zu nächstem Samplepunkt

Abbildung 4.12.: Anzahl der triangulierten Punkte pro Punkt der Punktwolke aufgetragen über die Größe der Punktwolke: 1000000, 3333333, 4000000, 6666666, 7000000 und 10000000 Punkte



-	●	●	●	●
p	C	Z	B	S
d	3			
n	x			
t	16			
s	-	$\frac{2}{100}n$		
w	-	konstant		
c	-	1		

C: Cycle-Partitioner; **Z**: Zuweisung zu Samplezentrum; **B**: Zuweisung zu Samplebegrenzung; **S**: Zuweisung zu nächstem Samplepunkt

Abbildung 4.13.: Standardabweichung der Größe der Partitionen; aufgetragen über die Größe der Punktwolke: 1000000, 3333333, 4000000, 6666666, 7000000 und 10000000 Punkte

4.6. Zusammenfassung und Ausblick

Funke und Sanders verkünden im Ausblick ihrer Arbeit zu paralleler Delaunay-Triangulierung den Bedarf an Load-Balancing-Strategien, die einen möglichst kleinen Rand anstreben. Diese Arbeit liefert Ansätze, die diesen Anforderungen nachkommen.

Die Zahl der Punkte, die als Teil eines Randes mehrfach trianguliert werden müssen, kann durch die hier vorgestellten Partitionierungsstrategien bis um den Faktor 10 verbessert werden.

Darüberhinaus ist auch eine deutliche Verbesserung der Balance der Partitionsgrößen gelungen. Es werden Partitionierungen erreicht, in denen die größte Partition um nur 1% größer ist als die kleinste, sodass eine sehr gleichmäßige Verteilung der Last erreicht wird.

Die Partitionierung einer Auswahl von Punkten auf die gesamte Punktwolke zu erweitern stellt somit einen vielversprechenden Ansatz zum Load-Balancing für parallele Delaunay-Triangulierung dar.

Von den eingeführten Strategien „Zuweisung zu Samplezentren“, „Zuweisung zu Samplebegrenzungen“ und „Zuweisung zum nächsten Samplepunkt“ hat vor allem letztere sich als sehr brauchbar herausgestellt.

Während diese Arbeit die Qualität verschiedener Ansätze untersucht, werden zukünftige Arbeiten sich mit ihrer Verbesserung beschäftigen.

Die Untersuchungen zeigen, dass bei den besten Ansätzen bis zu 50% der Gesamtlaufzeit für die Partitionierung anfällt. Verbesserungen der Implementierungen können also einen signifikanten Einfluss auf die Leistungsfähigkeit des gesamten Triangulierungsalgorithmus haben.

A | Literatur

[1] Z.Q. Cheng u. a. „Experiences in reverse-engineering of a finite element automobile crash model“. In: *Finite Elements in Analysis and Design* 37.11 (2001). Robert J. Melosh Medal Competition, S. 843–860.

[2] Charbel Farhat. „A simple and efficient automatic fem domain decomposer“. In: *Computers & Structures* 28.5 (1988), S. 579–602.

[3] Nikos Chrisochoides und Démian Nave. „Simultaneous mesh generation and partitioning for Delaunay meshes“. In: *Mathematics and Computers in Simulation* 54.4 (2000), S. 321–339.

[4] Sterling J Anderson, Sisir B Karumanchi und Karl Iagnemma. „Constraint-based planning and control for safe, semi-autonomous operation of vehicles“. In: *Intelligent Vehicles Symposium (IV), 2012 IEEE*. IEEE. 2012, S. 383–388.

[5] Daniel Funke und Peter Sanders. „Parallel d-D Delaunay Triangulations in Shared and Distributed Memory“. In: *2017 Proceedings of the Ninteenth Workshop on Algorithm Engineering and Experiments (ALENEX)*. 2017, S. 207–217.

[6] Siu-Wing Cheng, Tamal K Dey und Jonathan Shewchuk. *Delaunay mesh generation*. CRC Press, 2012.

[7] R. Said u. a. „Distributed parallel Delaunay mesh generation“. In: *Computer Methods in Applied Mechanics and Engineering* 177.1 (1999), S. 109–125.

[8] A. Bowyer. „Computing Dirichlet tessellations*“. In: *The Computer Journal* 24.2 (1981), S. 162–166.

[9] David F Watson. „Computing the n-dimensional Delaunay tessellation with application to Voronoi polytopes“. In: *The computer journal* 24.2 (1981), S. 167–172.

[10] Sangyoon Lee, Chan-Ik Park und Chan-Mo Park. „An improved parallel algorithm for Delaunay triangulation on distributed memory parallel computers“. In: *Parallel Processing Letters* 11.02n03 (2001), S. 341–352.

[11] Vicente HF Batista u. a. „Parallel geometric algorithms for multi-core computers“. In: *Computational Geometry* 43.8 (2010), S. 663–677.

[12] W Donald Frazer und AC McKellar. „Samplesort: A sampling approach to minimal storage tree sorting“. In: *Journal of the ACM (JACM)* 17.3 (1970), S. 496–507.

[13] Peter Sanders und Christian Schulz. „Think Locally, Act Globally: Highly Balanced Graph Partitioning“. In: *Proceedings of the 12th International Symposium on Experimental Algorithms (SEA'13)*. Bd. 7933. LNCS. Springer, 2013, S. 164–175.

[14] Peter Sanders und Christian Schulz. „Engineering multilevel graph partitioning algorithms“. In: *European Symposium on Algorithms*. Springer. 2011, S. 469–480.

[15] Peter Sanders und Christian Schulz. „KaHIP v0.53 - Karlsruhe High Quality Partitioning - User Guide“. In: *CoRR* (2013).

[16] Boost. *Boost C++ Libraries*. URL: `http://www.boost.org/` (besucht am 04. 03. 2018).

[17] Intel® TBB. *Intel® Threading Building Blocks*. URL: `https://www.threadingbuildingblocks.org/` (besucht am 04. 03. 2018).

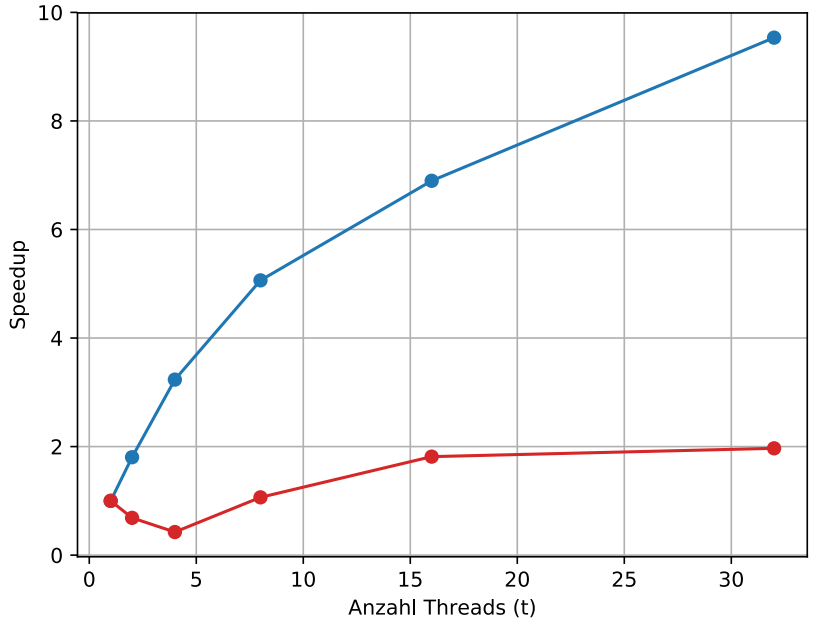
[18] CGAL. *The Computational Geometry Algorithms Library*. URL: `http://www.cgal.org/` (besucht am 04. 03. 2018).

[19] KaHIP. *Karlsruhe High Quality Partitioning*. URL: `http://algo2.iti.kit.edu/kahip/` (besucht am 04. 03. 2018).

[20] libkdtree++. *libkdtree++*. URL: `https://libkdtree.alioth.debian.org/` (besucht am 04. 03. 2018).

B | Anhang

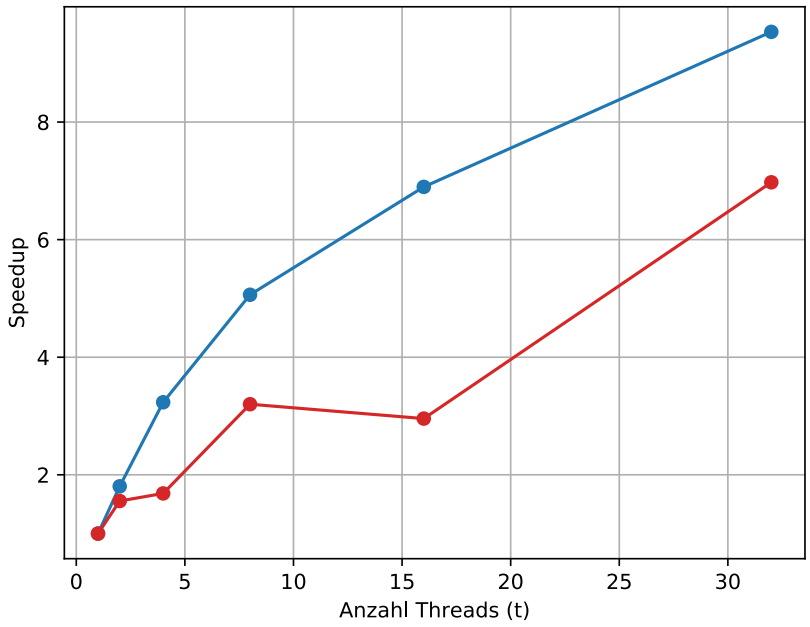
B.1. Zu Abbildung 4.2



-	●	●	●	●
p	C	Z	B	S
d	3			
n	6666666			
t	x			
s	-	$\sqrt{n}$		
w	-	konstant		
c	-	1		

C: Cycle-Partitioner; **Z**: Zuweisung zu Samplezentrum; **B**: Zuweisung zu Samplebegrenzung; **S**: Zuweisung zu nächstem Samplepunkt

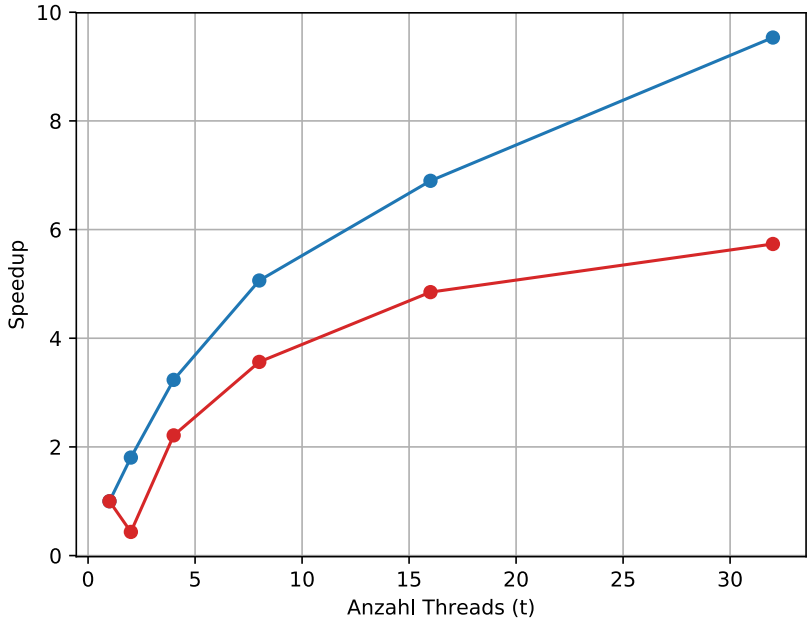
Abbildung B.1.



-	●	●	●	●
p	C	Z	B	S
d	3			
n	6666666			
t	x			
s	-	$\frac{1}{100}n$		
w	-	konstant		
c	-	1		

C: Cycle-Partitioner; **Z**: Zuweisung zu Samplezentrum; **B**: Zuweisung zu Samplebegrenzung; **S**: Zuweisung zu nächstem Samplepunkt

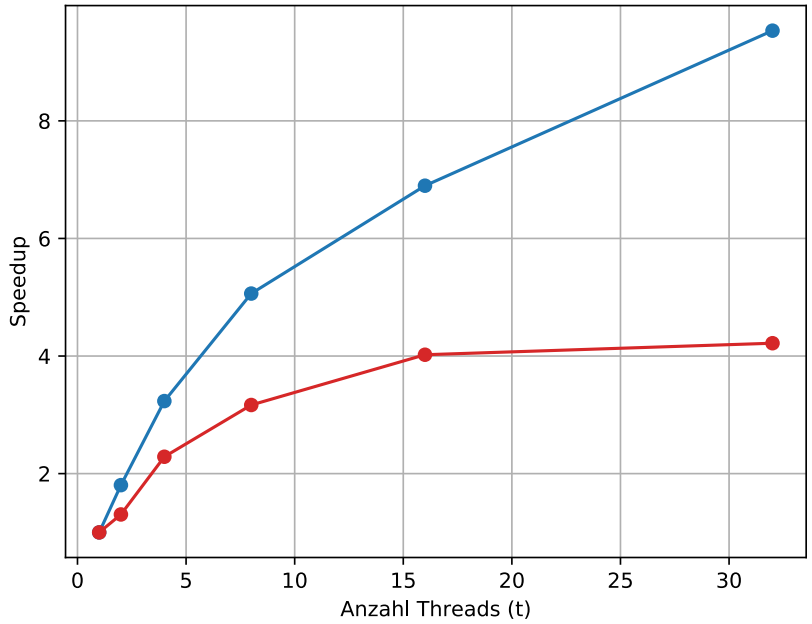
Abbildung B.2.



-	●	●	●	●
p	C	Z	B	S
d	3			
n	6666666			
t	x			
s	-	$\frac{5}{100}n$		
w	-	konstant		
c	-	1		

C: Cycle-Partitioner; **Z**: Zuweisung zu Samplezentrum; **B**: Zuweisung zu Samplebegrenzung; **S**: Zuweisung zu nächstem Samplepunkt

Abbildung B.3.

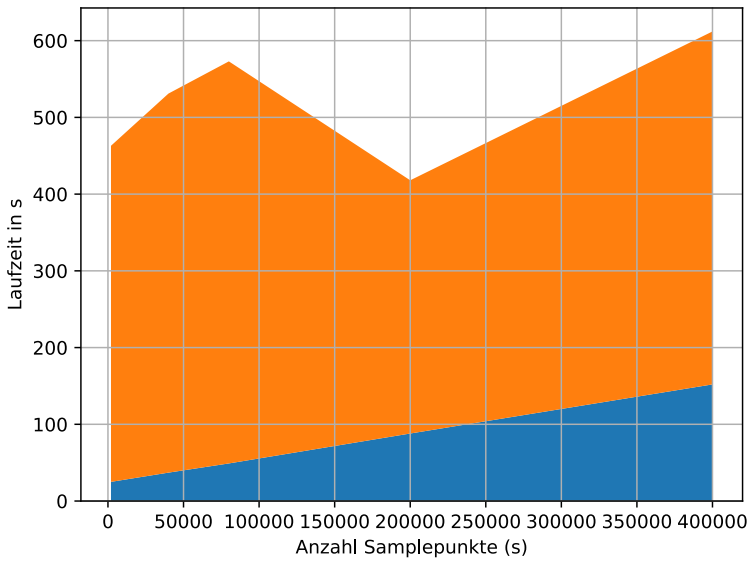


-	●	●	●	●
p	C	Z	B	S
d	3			
n	6666666			
t	x			
s	-	$\frac{10}{100}n$		
w	-	konstant		
c	-	1		

C: Cycle-Partitioner; **Z**: Zuweisung zu Samplezentrum; **B**: Zuweisung zu Samplebegrenzung; **S**: Zuweisung zu nächstem Samplepunkt

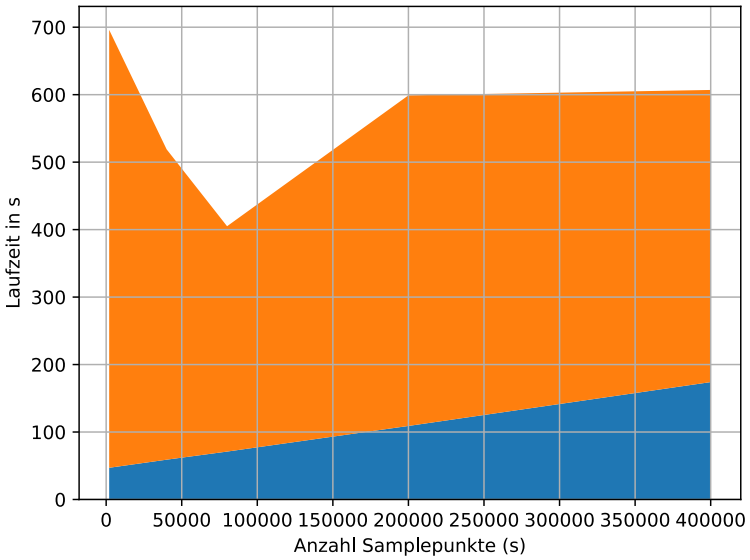
Abbildung B.4.

B.2. Zu Abbildung 4.3



d	3
n	4000000
t	16
s	x
w	konstant
c	1

Abbildung B.5.: Zuweisung zu Samplepartitionszentren

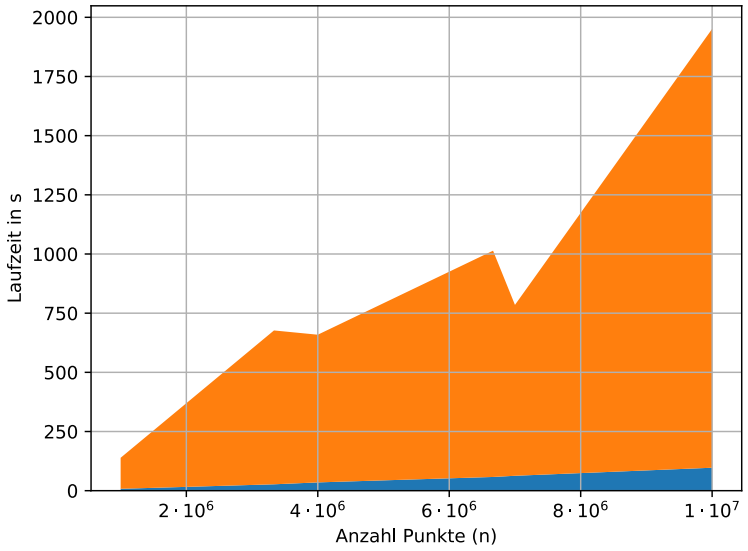


●	Partitionierungszeit
●	Triangulierungszeit

d	3
n	4000000
t	16
s	x
w	konstant
c	1

Abbildung B.6.: Zuweisung zu Samplepartitionsboxen

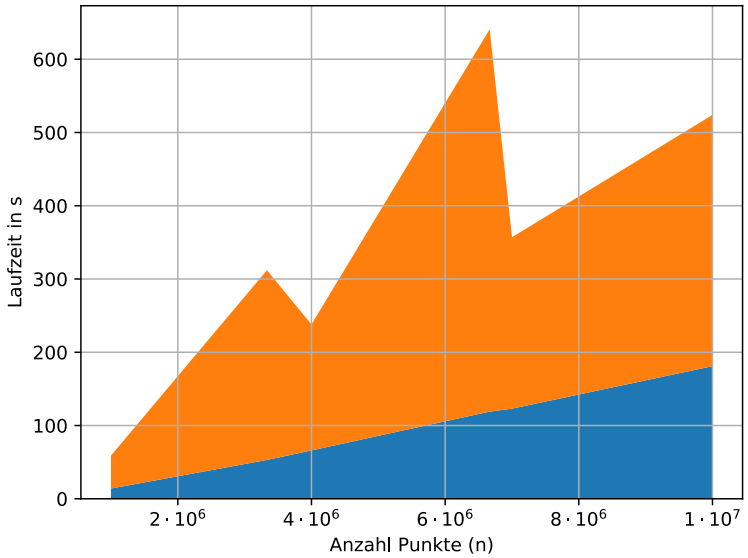
B.3. Zu Abbildungen 4.9 und 4.10



●	Partitionierungszeit
●	Triangulierungszeit

d	3
n	x
t	16
s	$\sqrt{n}$
w	konstant
c	1

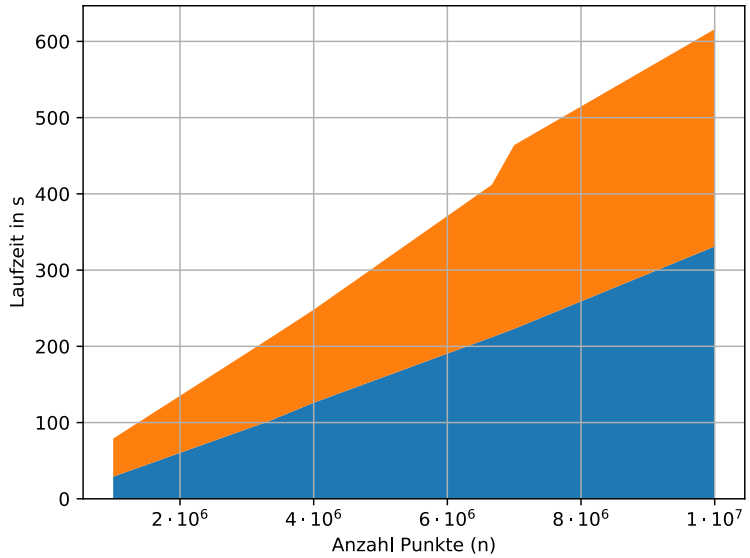
Abbildung B.7.: Zuweisung zum nächsten Samplepunkt



●	Partitionierungszeit
●	Triangulierungszeit

d	3
n	x
t	16
s	$\frac{1}{100}n$
w	konstant
c	1

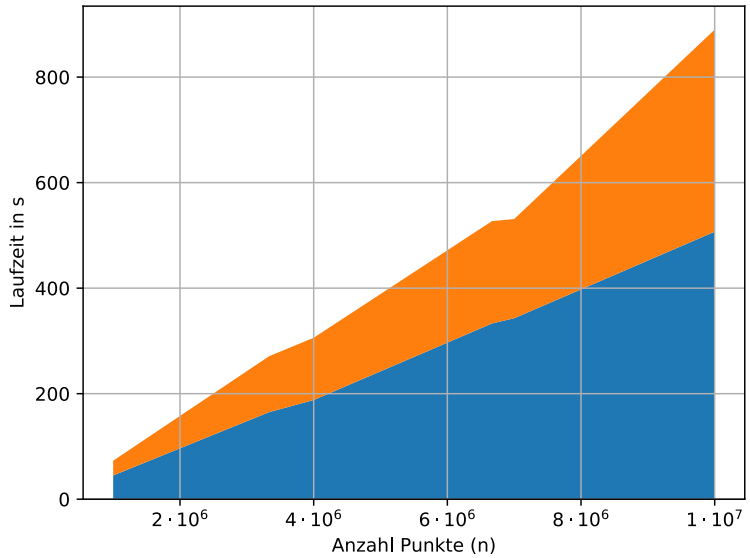
Abbildung B.8.: Zuweisung zum nächsten Samplepunkt



●	Partitionierungszeit
●	Triangulierungszeit

d	3
n	x
t	16
s	$\frac{5}{100}n$
w	konstant
c	1

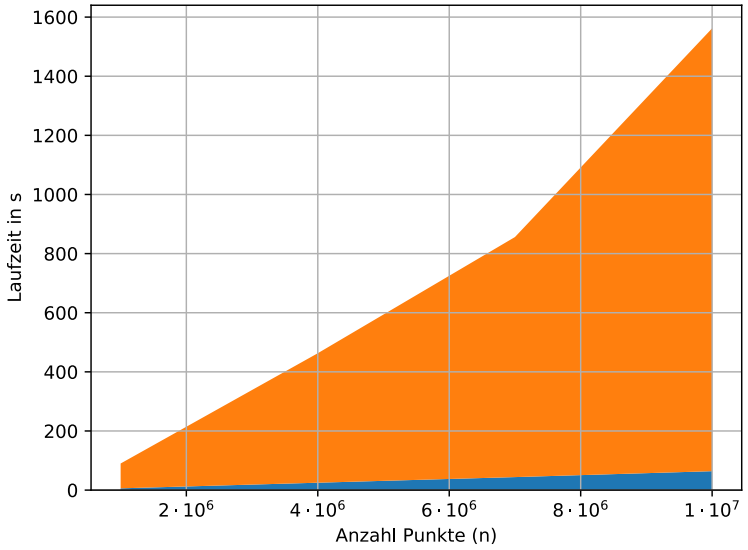
Abbildung B.9.: Zuweisung zum nächsten Samplepunkt



●	Partitionierungszeit
●	Triangulierungszeit

d	3
n	x
t	16
s	$\frac{10}{100}n$
w	konstant
c	1

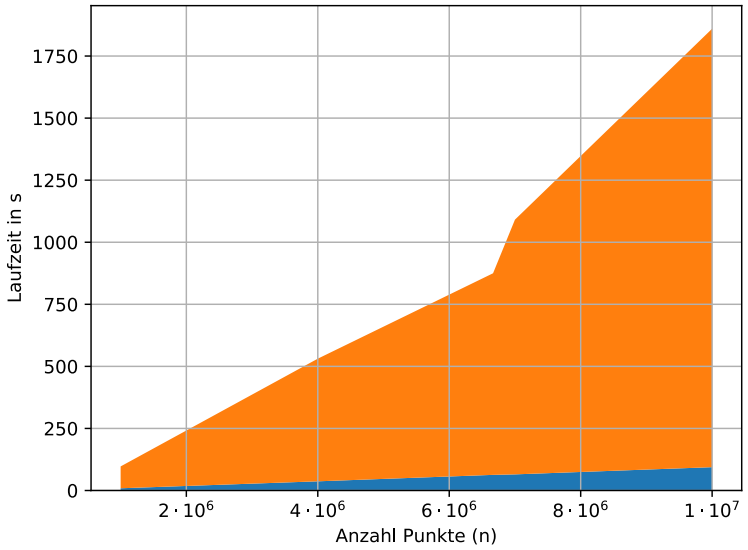
Abbildung B.10.: Zuweisung zum nächsten Samplepunkt



●	Partitionierungszeit
●	Triangulierungszeit

d	3
n	x
t	16
s	$\sqrt{n}$
w	konstant
c	1

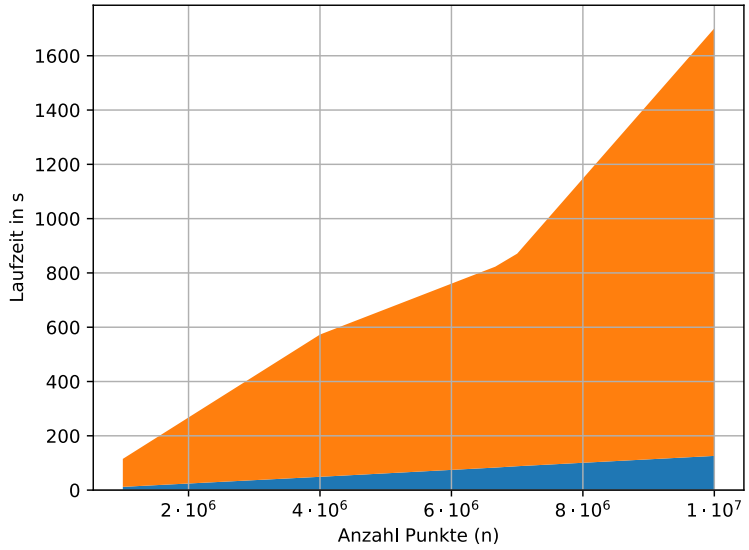
Abbildung B.11.: Zuweisung zu Samplepartitionszentren



●	Partitionierungszeit
●	Triangulierungszeit

d	3
n	x
t	16
s	$\frac{1}{100}n$
w	konstant
c	1

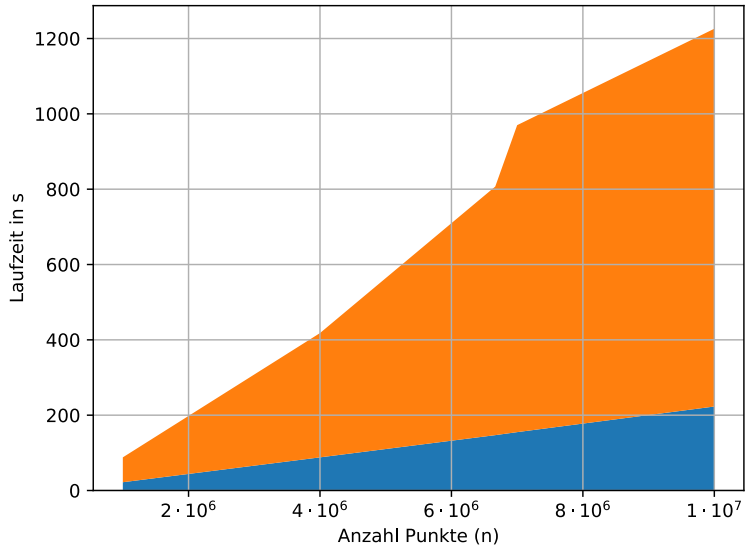
Abbildung B.12.: Zuweisung zu Samplepartitionszentren



●	Partitionierungszeit
●	Triangulierungszeit

d	3
n	x
t	16
s	$\frac{2}{100}n$
w	konstant
c	1

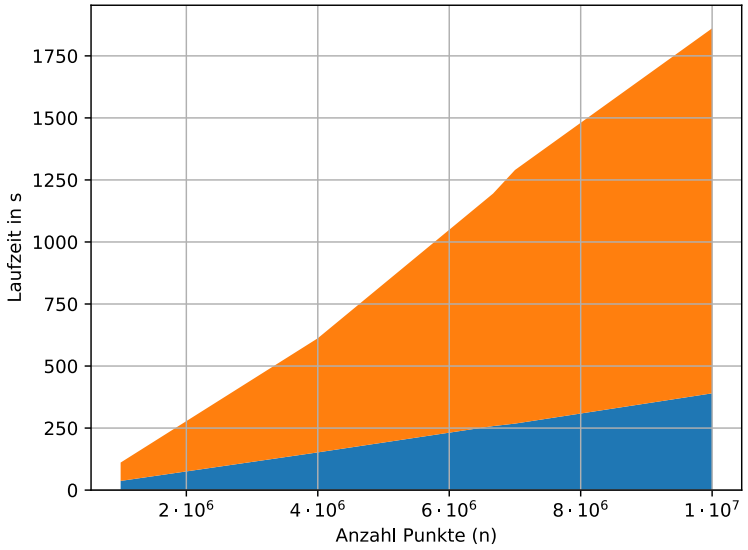
Abbildung B.13.: Zuweisung zu Samplepartitionszentren



●	Partitionierungszeit
●	Triangulierungszeit

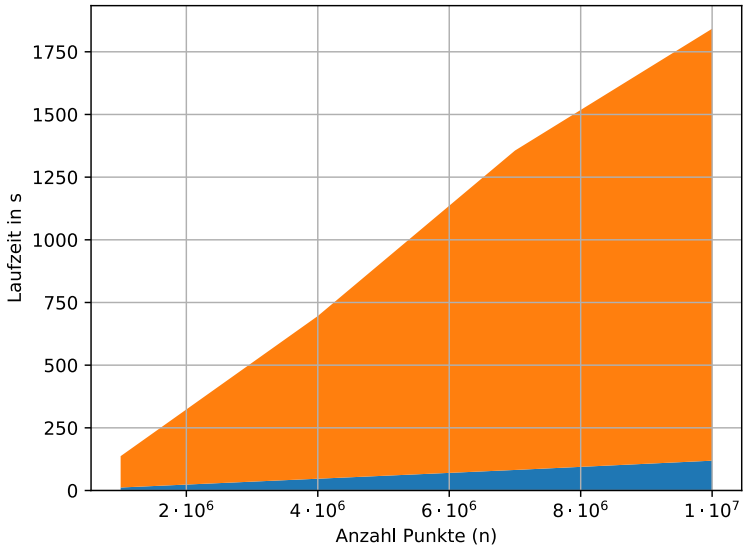
d	3
n	x
t	16
s	$\frac{5}{100}n$
w	konstant
c	1

Abbildung B.14.: Zuweisung zu Samplepartitionszentren



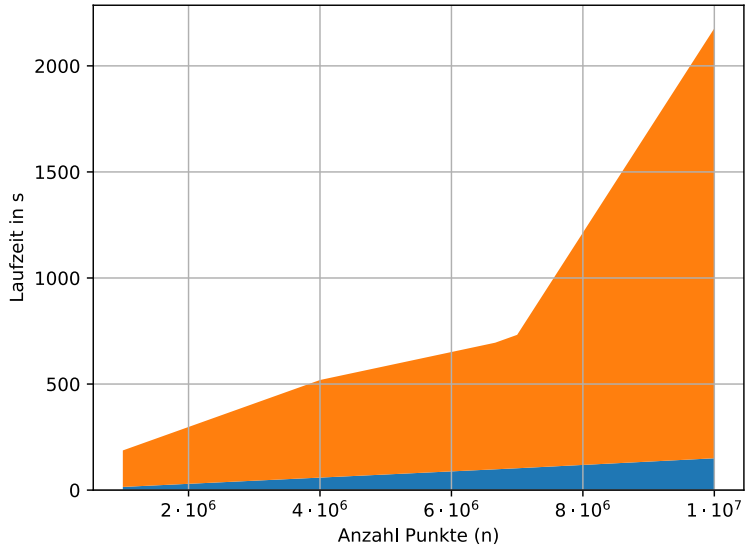
d	3
n	x
t	16
s	$\frac{10}{100}n$
w	konstant
c	1

Abbildung B.15.: Zuweisung zu Samplepartitionszentren



d	3
n	x
t	16
s	$\sqrt{n}$
w	konstant
c	1

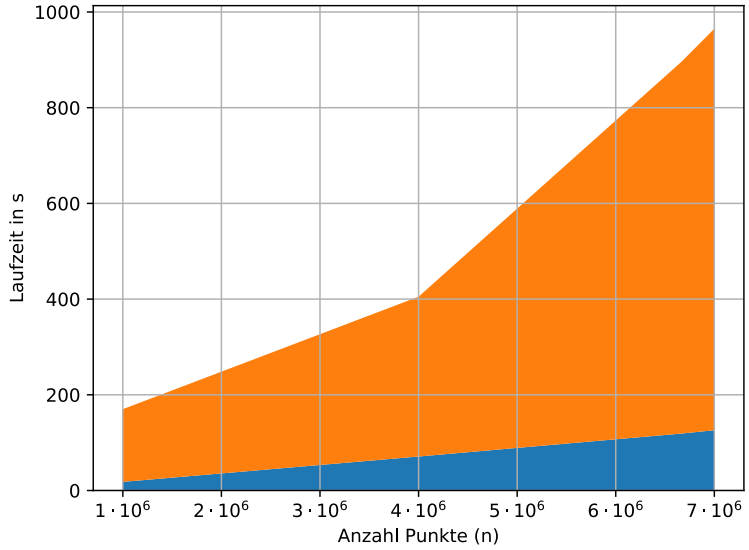
Abbildung B.16.: Zuweisung zu Samplepartitionsboxen



●	Partitionierungszeit
●	Triangulierungszeit

d	3
n	x
t	16
s	$\frac{1}{100}n$
w	konstant
c	1

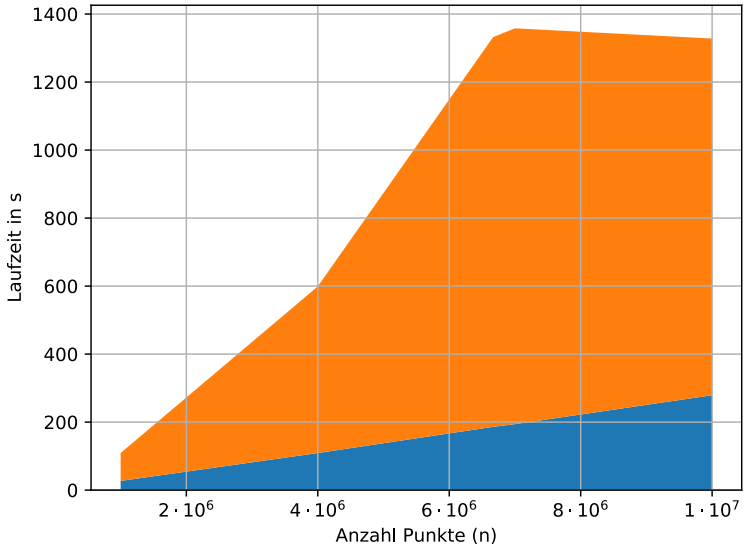
Abbildung B.17.: Zuweisung zu Samplepartitionsboxen



●	Partitionierungszeit
●	Triangulierungszeit

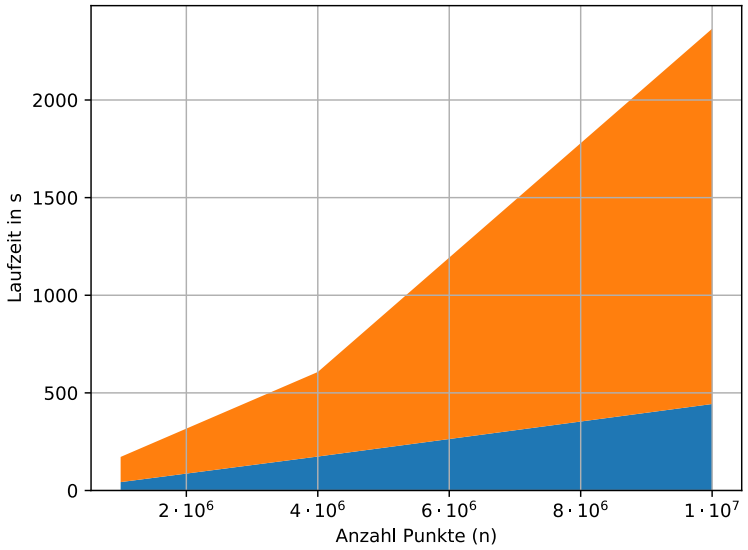
d	3
n	x
t	16
s	$\frac{2}{100}n$
w	konstant
c	1

Abbildung B.18.: Zuweisung zu Samplepartitionsboxen



d	3
n	x
t	16
s	$\frac{5}{100}n$
w	konstant
c	1

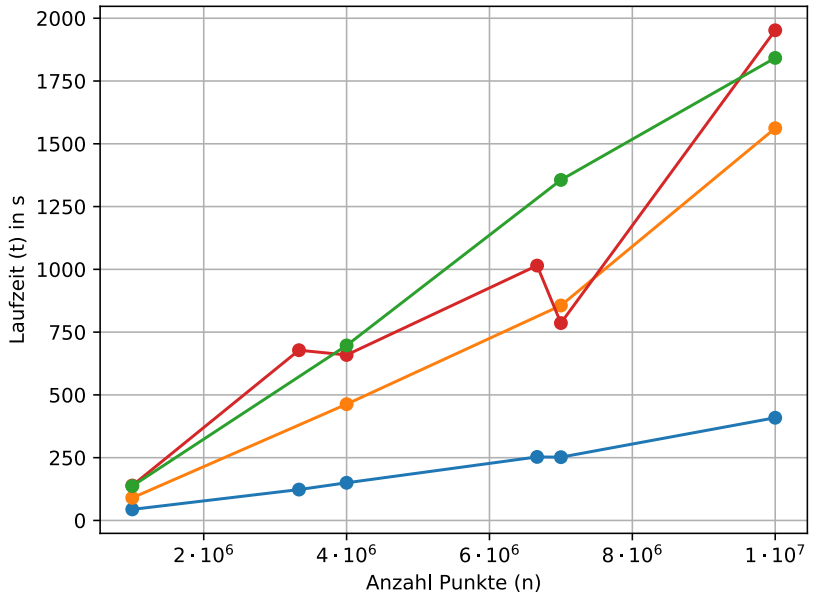
Abbildung B.19.: Zuweisung zu Samplepartitionsboxen



d	3
n	x
t	16
s	$\frac{10}{100}n$
w	konstant
c	1

Abbildung B.20.: Zuweisung zu Samplepartitionsboxen

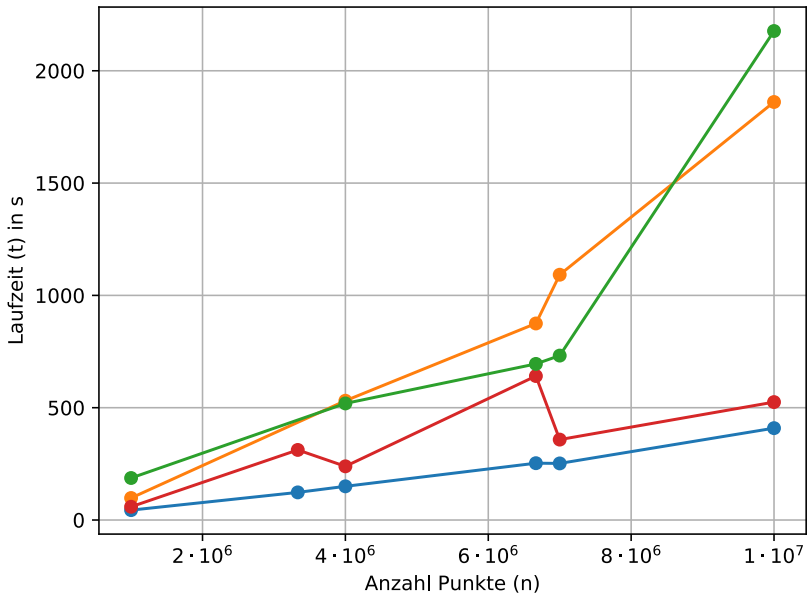
B.4. Zu Abbildung 4.11



-	●	●	●	●
p	C	Z	B	S
d	3			
n	x			
t	16			
s	-	$\sqrt{n}$		
w	-	konstant		
c	-	1		

C: Cycle-Partitioner; **Z**: Zuweisung zu Samplezentrum; **B**: Zuweisung zu Samplebegrenzung; **S**: Zuweisung zu nächstem Samplepunkt

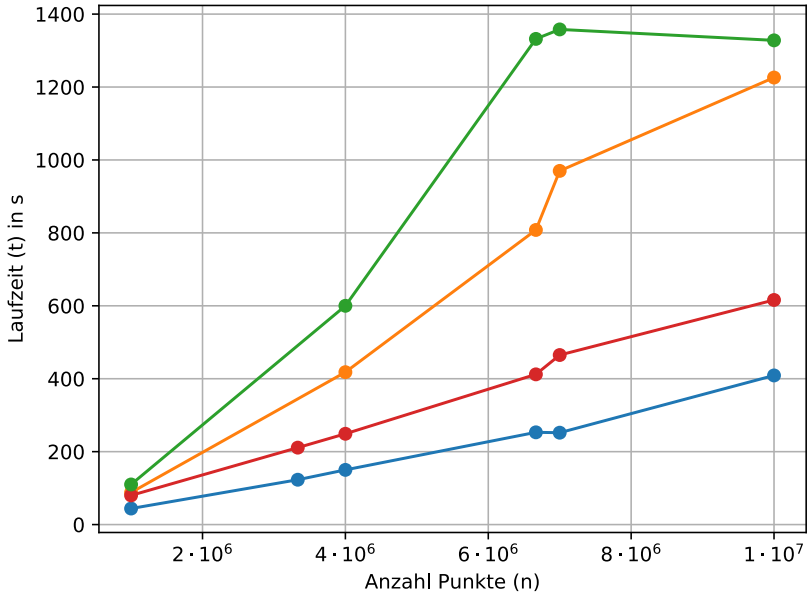
Abbildung B.21.



-	●	●	●	●
p	C	Z	B	S
d	3			
n	x			
t	16			
s	-	$\frac{1}{100}n$		
w	-	konstant		
c	-	1		

C: Cycle-Partitioner; **Z**: Zuweisung zu Samplezentrum; **B**: Zuweisung zu Samplebegrenzung; **S**: Zuweisung zu nächstem Samplepunkt

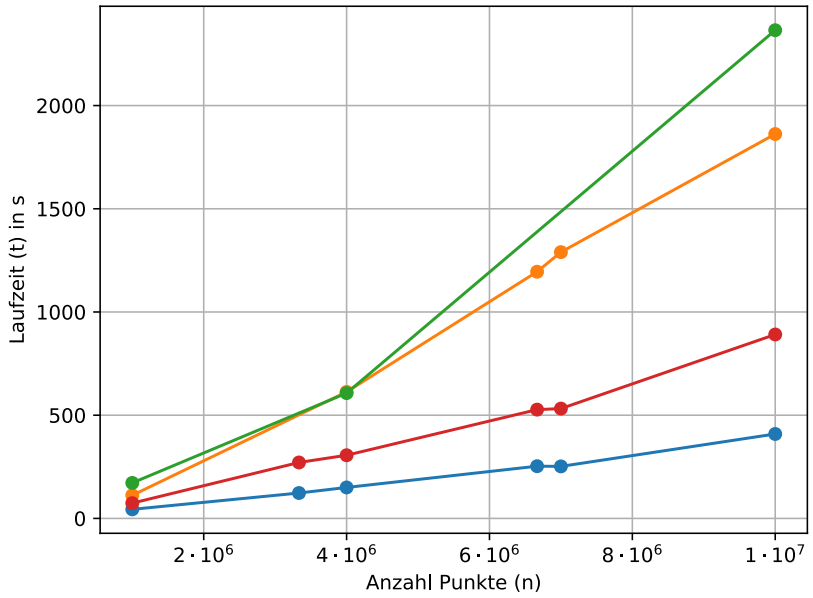
Abbildung B.22.



-	●	●	●	●
p	C	Z	B	S
d	3			
n	x			
t	16			
s	-	$\frac{5}{100}n$		
w	-	konstant		
c	-	1		

C: Cycle-Partitioner; **Z**: Zuweisung zu Samplezentrum; **B**: Zuweisung zu Samplebegrenzung; **S**: Zuweisung zu nächstem Samplepunkt

Abbildung B.23.

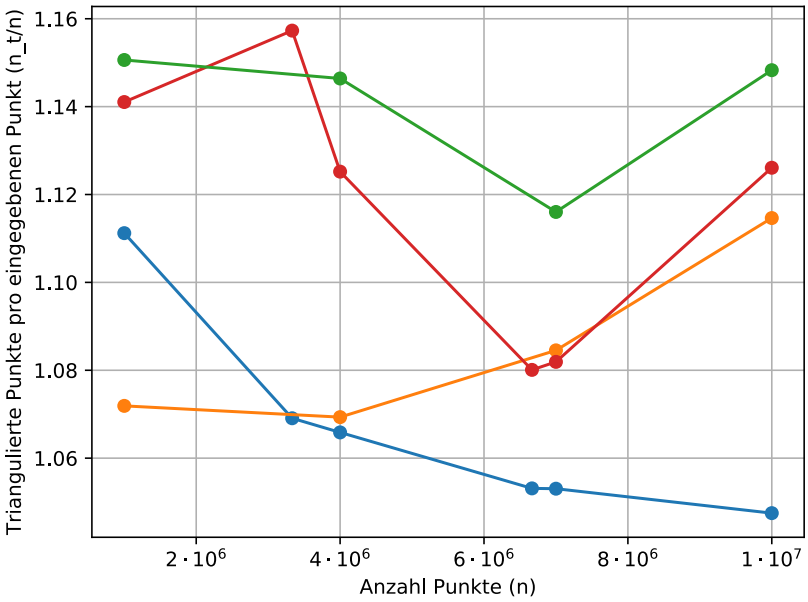


-	●	●	●	●
p	C	Z	B	S
d	3			
n	x			
t	16			
s	-	$\frac{10}{100}n$		
w	-	konstant		
c	-	1		

C: Cycle-Partitioner; **Z**: Zuweisung zu Samplezentrum; **B**: Zuweisung zu Samplebegrenzung; **S**: Zuweisung zu nächstem Samplepunkt

Abbildung B.24.

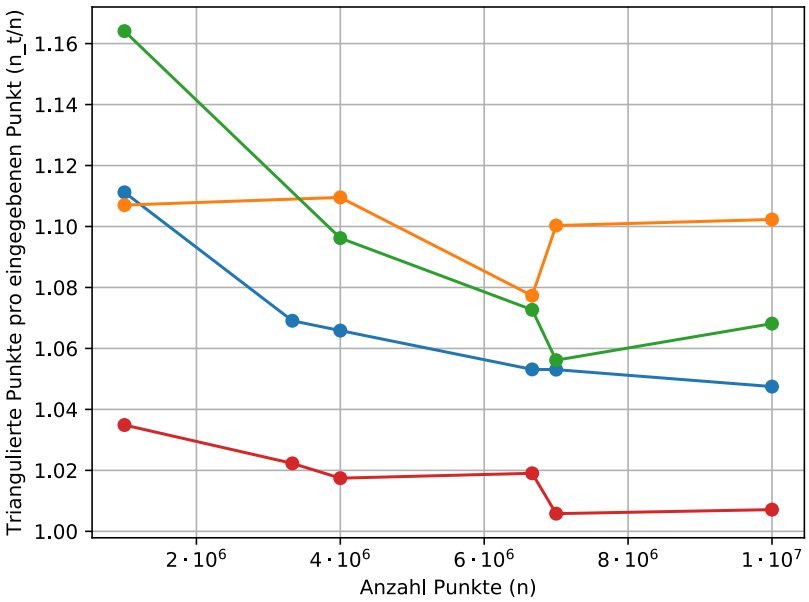
B.5. Zu Abbildung 4.12



-	●	●	●	●
p	C	Z	B	S
d	3			
n	x			
t	16			
s	-	$\sqrt{n}$		
w	-	konstant		
c	-	1		

C: Cycle-Partitioner; **Z**: Zuweisung zu Samplezentrum; **B**: Zuweisung zu Samplebegrenzung; **S**: Zuweisung zu nächstem Samplepunkt

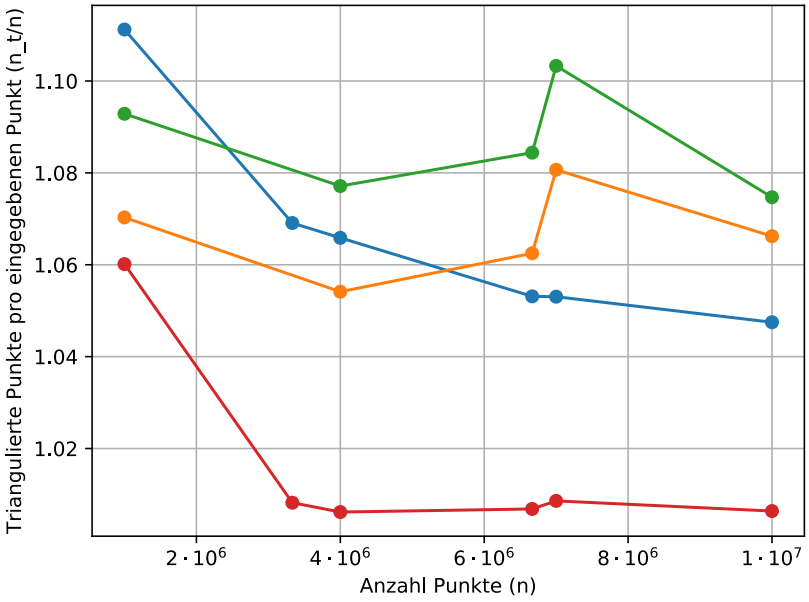
Abbildung B.25.



-	●	●	●	●
p	C	Z	B	S
d	3			
n	x			
t	16			
s	-	$\frac{1}{100}n$		
w	-	konstant		
c	-	1		

C: Cycle-Partitioner; **Z**: Zuweisung zu Samplezentrum; **B**: Zuweisung zu Samplebegrenzung; **S**: Zuweisung zu nächstem Samplepunkt

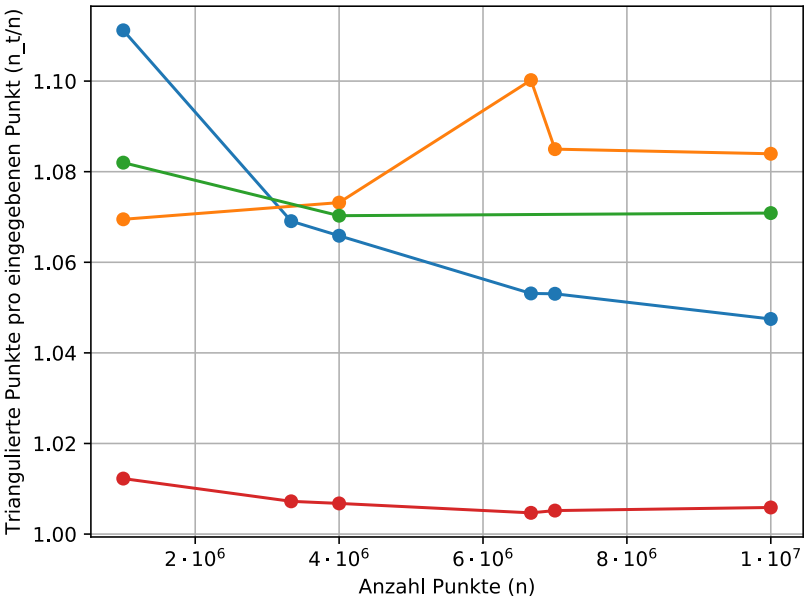
Abbildung B.26.



-	●	●	●	●
p	C	Z	B	S
d	3			
n	x			
t	16			
s	-	$\frac{5}{100}n$		
w	-	konstant		
c	-	1		

C: Cycle-Partitioner; **Z**: Zuweisung zu Samplezentrum; **B**: Zuweisung zu Samplebegrenzung; **S**: Zuweisung zu nächstem Samplepunkt

Abbildung B.27.

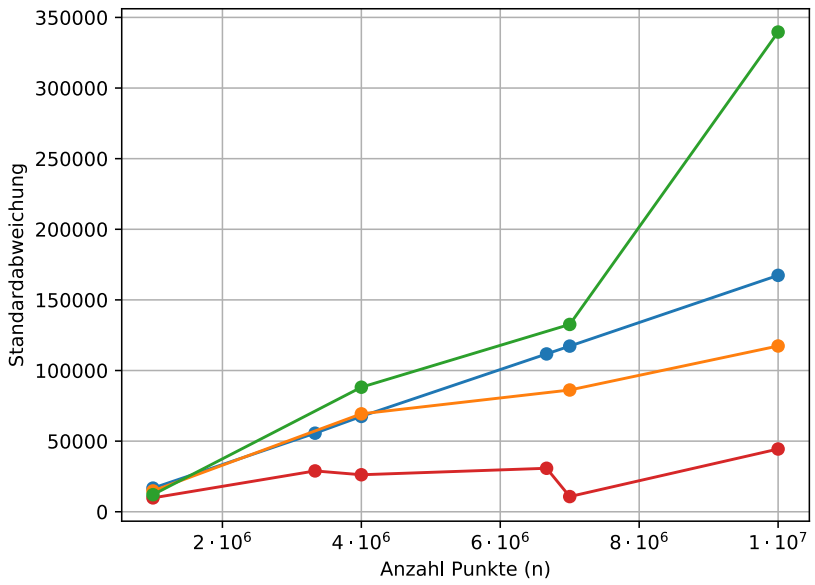


-				
p	C	Z	B	S
d	3			
n	x			
t	16			
s	-	$\frac{10}{100}n$		
w	-	konstant		
c	-	1		

C: Cycle-Partitioner; **Z**: Zuweisung zu Samplezentrum; **B**: Zuweisung zu Samplebegrenzung; **S**: Zuweisung zu nächstem Samplepunkt

Abbildung B.28.

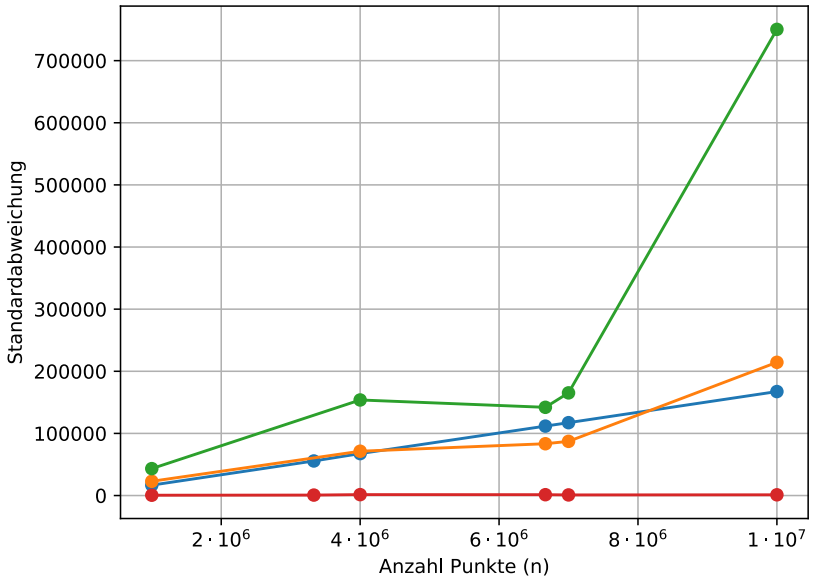
B.6. Zu Abbildung 4.13



-	●	●	●	●
p	C	Z	B	S
d	3			
n	x			
t	16			
s	-	$\sqrt{n}$		
w	-	konstant		
c	-	1		

C: Cycle-Partitioner; **Z**: Zuweisung zu Samplezentrum; **B**: Zuweisung zu Samplebegrenzung; **S**: Zuweisung zu nächstem Samplepunkt

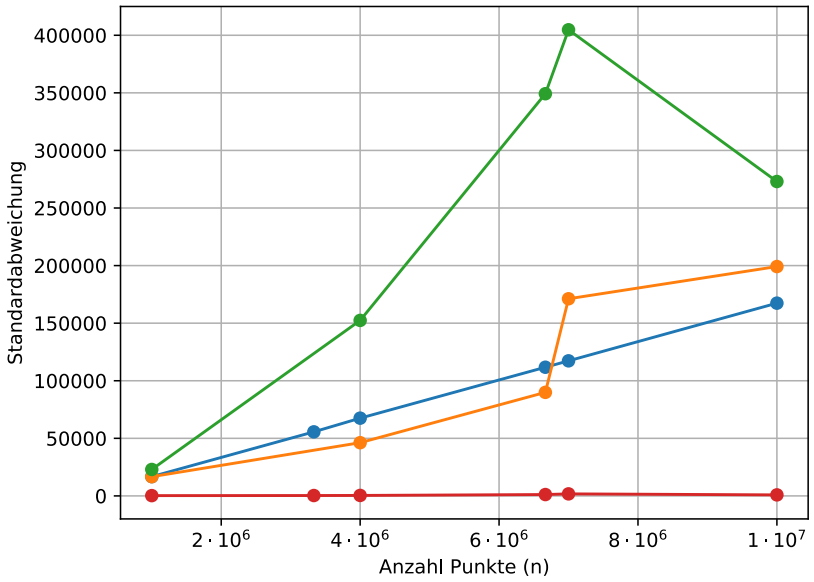
Abbildung B.29.



-	●	●	●	●
p	C	Z	B	S
d	3			
n	x			
t	16			
s	-	$\frac{1}{100}n$		
w	-	konstant		
c	-	1		

C: Cycle-Partitioner; **Z**: Zuweisung zu Samplezentrum; **B**: Zuweisung zu Samplebegrenzung; **S**: Zuweisung zu nächstem Samplepunkt

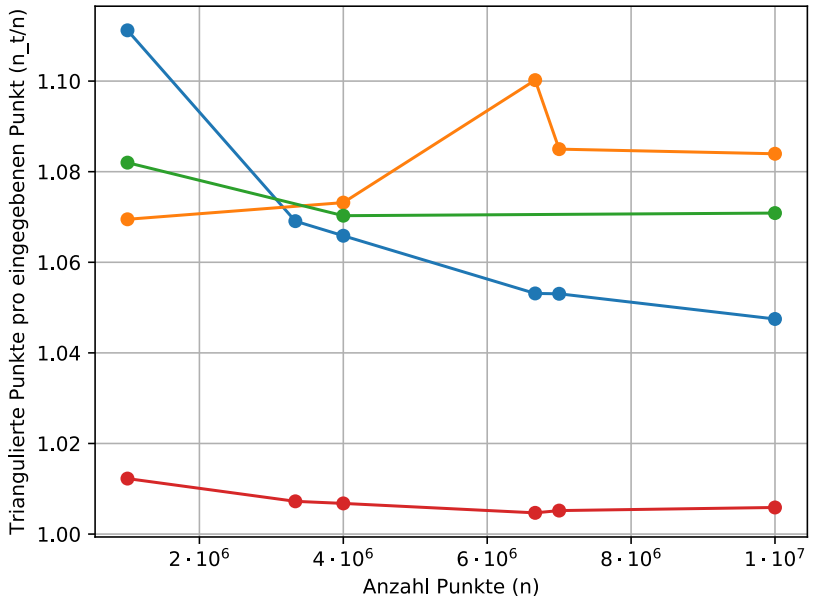
Abbildung B.30.



-	●	●	●	●
p	C	Z	B	S
d	3			
n	x			
t	16			
s	-	$\frac{5}{100}n$		
w	-	konstant		
c	-	1		

C: Cycle-Partitioner; **Z**: Zuweisung zu Samplezentrum; **B**: Zuweisung zu Samplebegrenzung; **S**: Zuweisung zu nächstem Samplepunkt

Abbildung B.31.



-	●	●	●	●
p	C	Z	B	S
d	3			
n	x			
t	16			
s	-	$\frac{10}{100}n$		
w	-	konstant		
c	-	1		

C: Cycle-Partitioner; **Z**: Zuweisung zu Samplezentrum; **B**: Zuweisung zu Samplebegrenzung; **S**: Zuweisung zu nächstem Samplepunkt

Abbildung B.32.