Unterschrift des Betreuers

TECHNISCHE UNIVERSITÄT WIEN
Vienna University of Technology

DIPLOMARBEIT

Segmentierung großer Punktwolken mittels Region Growing

ausgeführt am

Department für Geodäsie und Geoinformation
Forschungsgruppe Photogrammetrie
Technische Universität Wien

unter Anleitung von

Univ.Prof. Dr.techn. Norbert Pfeifer
Projektass. Dr.techn. Johannes Otepka

durch

Markus Pöchtrager, BSc
Matr.-Nr.: 1026601
poechtma@gmail.com

Wien, Mai 2016

Unterschrift des Verfassers

Danksagung

Es ist mir ein persönliches Anliegen, allen Personen Danke zu sagen, die mich das Studium hindurch begleitet und bei der Erstellung dieser Arbeit unterstützt haben. Mein besonderer Dank gilt Herrn Univ. Prof. Dr. Norbert Pfeifer für die interessante Themenstellung und Herrn Dr. Johannes Otepka, der mich in ausgezeichneter Weise beim Schreiben dieser Arbeit betreut hat. Die schnellen und kompetenten Lösungsvorschläge bei fachlichen Fragen und Problemen haben mir ganz wesentlich beim Erstellen dieser Arbeit geholfen.
Dank geht aber auch an meine Eltern für die großartige Unterstützung in allen Belangen und an Lisi, die in den anstrengenden Wochen des Studiums immer für mich da war.

Kurzfassung

Die immer größer werdenden Datenmengen von 3D-Punktwolken, gewonnen durch Airborne Laserscanning, terrestrisches Laserscanning und Image Matching, ermöglichen eine Vielzahl unterschiedlichster Berechnungen und Datenanalysen. Die Anwendungsgebiete reichen von Monitoring-Aufgaben (z.B. Bauwerksüberwachung, Monitoring von Hangrutschungen, etc.) über archäologische Auswertungen und Vegetationskartierung bis hin zu 3D-Stadtmodellierung. Für Berechnungen auf Datensätzen mit vielen Millionen von Punkten wird dabei von den Algorithmen eine hohe Effizienz hinsichtlich der Laufzeit gefordert.
Das Verfahren der Segmentierung liefert für Punktwolken eine Gruppierung von gleichartigen Punkten anhand eines Homogenitätskriteriums. Diese Gruppeninformation ermöglicht einen effizienten Zugriff auf Punkte mit gleichen Eigenschaften. Die Segmentierung ist damit einer der ersten Schritte in der Prozessierungskette vieler Anwendungen.
Die vorliegende Arbeit stellt ein Konzept für eine Segmentierung von großen Punktwolken mit *Seeded Region Growing* vor. Da die Verarbeitungseinheit nicht beliebig große Datensätze in den Arbeitsspeicher einlesen kann, müssen diese in kleinere Einheiten aufgeteilt werden. Die Punktwolke wird in rechteckige Teilpunktwolken (*Kacheln*) ohne Überlappungsbereich unterteilt. Die mosaikartig zusammengesetzten Kacheln werden unabhängig voneinander segmentiert. Dadurch wird eine parallele Prozessierung der Kacheln - auf mehrere *Threads* verteilt - ermöglicht. Anschließend werden benachbarte gleichartige Segmente aus den Teilpunktwolken zusammengeführt.
Wie diese Arbeit zeigt sind die Ergebnisse der Segmentierung nicht von der Größe der Teilpunktwolke sondern hauptsächlich vom Homogenitätskriterium abhängig. Die Punktwolke kann dadurch in Kacheln mit beliebiger Größe unterteilt werden, um die Laufzeit und den Speicherplatzbedarf der Segmentierung zu optimieren.

Abstract

The growing amount of 3D point cloud data obtained by airborne laser scanning, terrestrial laser scanning and image matching, allows a variety of different calculations and data analysis. Applications range from monitoring tasks (e.g. structural health monitoring, landslides monitoring, etc.), archaeological evaluations, vegetation mapping to 3D city modeling. For processing huge data sets with millions of points highly efficient algorithms are required.

Segmentation of point cloud data provides a grouping of points based on a similarity criterion. This group information enables efficient access to points with the same properties. Thus segmentation is one of the first steps within the processing chain of many applications.

This thesis presents a concept for segmentation of large point clouds with *Seeded Region Growing*. Since the processing unit can not read huge dataset into main memory, the data must be divided into smaller parts. The point cloud is divided into rectangular non-overlapping parts (*tiles*). The tiles are then processed independently within the segmentation. This allows parallel computation by distributing tiles to multiple processing threads. Afterwards adjacent segments from different tiles are merged.

As it is shown the results of the segmentation do not depend on the tile size, but are mainly influenced by the similarity criterion. The point cloud can thus be divided into arbitrary tiles to optimize for processing speed and memory footprint.

Inhaltsverzeichnis

Kapitel 1

Einleitung

Die Aufnahme von dreidimensionalen Punktwolken für die Modellierung von Objekten in der realen Welt, mithilfe von Messgeräten aus Photogrammetrie und Laserscanning, gewinnt zunehmend an Bedeutung. Moderne Verfahren für die automatisierte Datenaufbereitung aus den Messdaten leisten dafür einen entscheidenden Beitrag. Durch die stetige technische Weiterentwicklung und die wachsende Popularität der Verfahren werden auch die gewonnenen Datenmengen an Punktwolken immer größer. Um die großen Datenmengen auch auswerten zu können, ist man deshalb in weiterer Folge an einer automatisierten Auswertung, sowie an Methoden, welche die Speicherung und Verwaltung von Punktwolken vereinfachen, interessiert. Ähnlich wie bei digitaler Bildverarbeitung und Bildkompression ist man auf der Suche nach Verfahren zur Datenkompression, um den Speicherbedarf zu reduzieren. Außerdem versucht man, zum Beispiel mit Algorithmen der Mustererkennung, zusätzliche Informationen aus den Daten zu generieren. Während Bilddateien für jedes Pixel häufig nur Farbinformation liefern, können Punktwolken eine große Liste von Attributen für jeden Bildpunkt speichern. Viele Algorithmen, die ursprünglich für die Bildverarbeitung entwickelt wurden, können durch ähnliche Datenstruktur für Punktwolken adaptiert werden.
Im Rahmen dieser Arbeit wurde ein Verfahren zur Segmentierung von großen Punktwolken entwickelt. Eine Segmentierung teilt einen Datensatz anhand eines globalen oder lokalen Kriteriums in Segmente mit gleicher Eigenschaft. Ein segmentierter Datensatz ermöglicht dadurch einen schnellen Zugriff auf gleichartige Elemente. Außerdem dienen die Segmente einem besseren Verständnis der Zusammenhänge zwischen Elementen in den Daten. Vosselman u. a. [2004] beschreiben den Vorgang der Segmentierung als *Erkennung von Struktur in Punktwolken*. Die Flächensegmentierungsverfahren können dabei grob unterteilt werden in Verfahren zur Segmentierung anhand eines Kriteriums und Verfahren mit direkter Schätzung von Oberflächenparametern.
Gerade bei der Verarbeitung von großen Datensätzen stoßen handelsübliche Rechner

schnell an die Grenzen von Prozessor-Laufzeit und Arbeitsspeicher. Aber auch Supercomputer können nicht beliebig große Datensätze in einem Stück verarbeiten. Das in der Arbeit entwickelte Konzept zielt deshalb im Besonderen auf eine Parallelisierung der Segmentierung ab. Dadurch kann die Segmentierung mit mehreren Prozessoren, oder auch auf verteilten Systemen (*Computer Cluster*) berechnet werden.

Aufbau der Arbeit Die Arbeit gliedert sich im Folgenden in ein Grundlagenkapitel (Kapitel 2), welches eine Einführung in die Grundlagen für die Aufnahme und Verwaltung von Punktwolken gibt. Außerdem beinhaltet das Kapitel aktuelle Verfahren zur Segmentierung von Punktwolken sowie Lösungsansätze für die Parallelisierung bei großen Punktwolken.
In Kapitel 3 wird ein im Rahmen dieser Arbeit entwickeltes Konzept zur Segmentierung großer Punktwolken vorgestellt und detailliert auf die Merkmale dieses Verfahrens eingegangen. Das Konzept basiert auf einem *Split-and-Merge*-Verfahren (*teilen und zusammenführen*), bei dem die Punktwolke in kleinere Teilpunktwolken (Kacheln) unterteilt und anschließend wieder zusammengeführt wird.
Die für die Tests des Segmentierungsverfahrens verwendeten Datensätze werden im Kapitel 4 beschrieben. Kapitel 5 beinhaltet die Ergebnisse aus unterschiedlichen Tests der Segmentierung. Die Laufzeitanalysen und Segmentierungsergebnisse geben dabei dabei Antworten auf folgende zwei Fragen:

- Liefert die Segmentierung unabhängig von der Größe der Teilpunktwolken (Kachelgröße) immer die gleichen Ergebnisse?
- Welchen Einfluss hat die Wahl der Kachelgröße auf die Laufzeit der Segmentierung?

Die Ergebnisse der Arbeit werden in Kapitel 6 zusammengefasst und ein Ausblick auf weitere Entwicklungsmöglichkeiten gegeben.

Kapitel 2

State of the art

In diesem Kapitel wird zum besseren Verständnis der Arbeit ein kurzer Einblick in die theoretischen und technischen Grundlagen sowie den aktuellen Stand der Technik gegeben. Zunächst wird erklärt, wie mit Laserscanning und Photogrammetrie aus den Objekten in der realen Welt dreidimensionale Punktwolken mit Attributinformationen aufgenommen werden können. Weiters werden die theoretischen Grundlagen zur Segmentierung von großen Punktwolken geliefert. Dabei werden verschiedene *State of the art* Segmentierungsverfahren, wie etwa das in der Arbeit verwendete *Seeded Region Growing*, kurz beschrieben. Das Kapitel wird abgeschlossen mit aktuellen Ansätzen zur Parallelisierung der Segmentierungsverfahren. Als Grundlage dafür wird auch auf die Datenstruktur der Punktwolke, sowie auf die Methoden im Datenmanagement eingegangen.

2.1 Punktwolken und Aufnahmemethoden

Die fortschreitende Entwicklung von Laserscannern hat in den vergangenen Jahren die Arbeit im Vermessungswesen stark mitgeprägt. Großflächige 3D-Punktwolken von Gelände und Gebäuden können von wenigen Standpunkten aus terrestrisch (TLS), oder durch Airborne Laserscanning (ALS) kontinuierlich aufgenommen werden. Der aktuelle Trend der Drohnenvermessung (UAV-Vermessung) bringt neue Alternativen zu klassischem ALS und macht es kostengünstiger und flexibler einsetzbar. Im Folgenden werden die Messprinzipien von terrestrischem und Airborne Laserscanning erklärt. [vgl. Kraus, 2004, S. 449-477]

2.1.1 Messprinzip - Airborne Laserscanning

Bei ALS-Aufnahmen wird ein gebündelter Laserstrahl von einem Flugzeug oder einer Drohne (unmanned aerial vehicle - UAV) ausgesendet. Der Scanner tastet dabei im Flug die darunter liegende Ebene ab (Abbildung 2.1). Je nach Bauart des Laserscanners wird

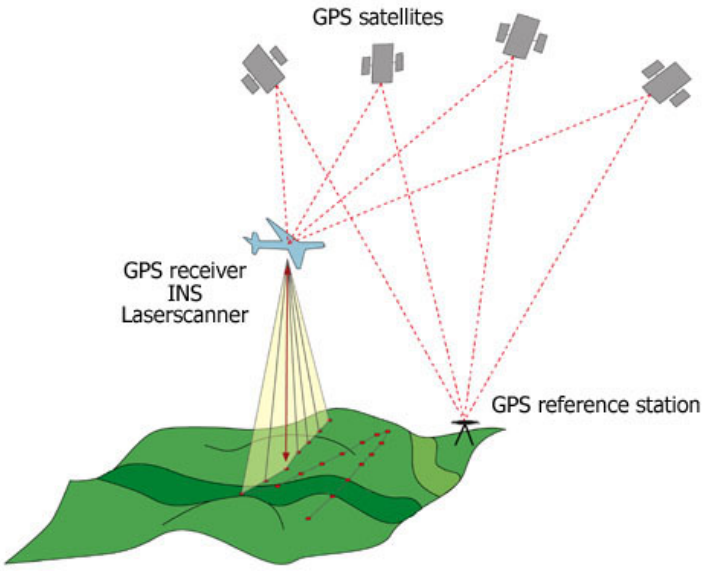

Abbildung 2.1: Messprinzip ALS [TerraImaging, 2016]

die Ebene unterschiedlich gescannt. Die in Abbildung 2.2 dargestellten Scan-Muster werden mit folgenden Spiegel-Systemen aufgenommen:

a Oszillierender Spiegel

b Rotierender Polygonspiegel

c Palmer Scanner (Nutierender Spiegel)

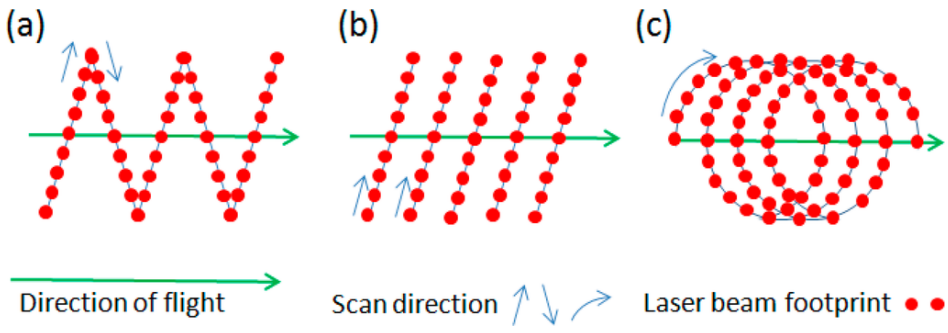

Abbildung 2.2: Scan-Muster von verschiedenen Mess-Mechanismen [Fernandez-Diaz u. a., 2014]

Aus der Laufzeit eines ausgesendeten Impulses zwischen Laserscanner und reflektierenden Objekt lässt sich die doppelte Entfernung als $2D = c * t$ berechnen, wobei die Ausbreitungsgeschwindigkeit c vom durchlaufenen Medium abhängt. Um von der Entfernung auf die Position des Objektes schließen zu können, muss die Laserscanner-Position, Orientierung und der Stellwinkel zu jedem Zeitpunkt genau bestimmt sein. Position und Orientierung werden dafür mittels GNSS (Satellitennavigationsverfahren, engl. *Global Navigation Satellite System*) und IMU (Inertiales Messsystem, engl. *Inertial Measurement Unit*) ermittelt. GNSS, IMU und die Laserscanner-Messeinheit werden dabei im Mikrosekundenbereich miteinander synchronisiert.
Da ein ausgesendeter Impuls an unterschiedlichen Objekten (nur teilweise) reflektiert oder absorbiert werden kann, kommen reflektierte Impulse zu unterschiedlichen Zeiten wieder am Laserscanner an. Dabei liefert das erste Echo im Allgemeinen den höchstgelegenen Punkt des Objekts und das letzte Echo den tiefsten Punkt des Objekts. Wenn man Mehrwegeffekte des Laserstrahls vernachlässigt, liefern die letzten Echos Geländepunkte, sofern der Laserstrahl bis zum Boden durchdringen kann (Abbildung 2.3). Airborne Laserscanning eignet sich dadurch sehr gut, um digitale Geländemodelle (DGM) abzuleiten.

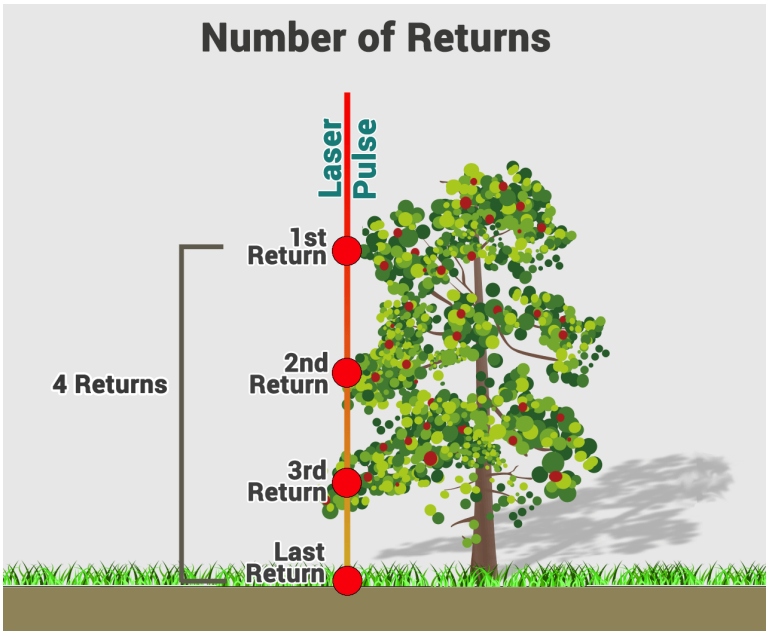

Abbildung 2.3: Anzahl der Echos [GISGeography, 2016]

2.1.2 Messprinzip - Terrestrisches Laserscanning

Im Unterschied zu ALS wird bei der Messung mit terrestrischen Laserscannern der Scanner während des Abtastvorgangs nicht bewegt. Der Scanner muss also in zwei Richtungen schwenkbar sein, um den Raum rundherum aufnehmen zu können. Terrestrische Laserscanner rotieren üblicherweise um die Stechachse und besitzen einen rotierenden Umlenkspiegel, der den Laserstrahl in Zielachsrichtung ablenkt.
Um eine dreidimensionale Punktwolke eines Objekts zu erhalten, wird das Objekt von mehreren Standpunkten aus so abgetastet, dass sich die Punktwolken der einzelnen Scans überlappen. Die einzelnen Scans werden anschließend in ein globales Koordinatensystem georeferenziert oder in einem lokalen Koordinatensystem relativ zueinander orientiert.

2.1.3 Punktwolken mit Photogrammetrie

Die Entwicklungen in *Structure from Motion* (SfM) und *Dense Image Matching* (DIM) ermöglichen heutzutage eine beinahe vollständig automatisierte 3D-Objekterkennung und 3D-Rekonstruktion aus 2D-Bildern. Bei den Verfahren werden korrespondierende Punkte (Feature-Punkte) in verschiedenen Bildern gesucht. Die Verknüpfung von Punkten aus mindestens zwei Bildern liefert, ähnlich wie beim stereoskopischen Sehen, räumliche Information. Dafür ermöglichen Methoden wie etwa SIFT (Scale Invariant Feature Transform) automatisiert eine relative Orientierung zwischen den Bildern zu berechnen. Durch *Dense Image Matching*-Methoden, wie etwa *Semi-global matching*, können die Bilder Pixel für Pixel verglichen werden und eine große Dichte an Verknüpfungspunkten geschaffen werden. *Semi-global matching* berechnet dabei *Disparity Maps*, also Karten mit der Verschiebung von Objekten (Disparation) zwischen zwei Aufnahmen. Diese *Disparity Maps* können als 2.5D Repräsentation der Geometrie im Raum gesehen werden. Die Vereinigung mehrerer *Disparity Maps* liefert ein 3D-Modell. Allgemein kann man sagen, dass durch eine große Dichte an Verknüpfungspunkten eine 3D-Punktwolke mit hoher Qualität aus den Bildern abgeleitet werden. [Brown und Lowe, 2005, vgl.] [Hirschmüller, 2008, vgl.]

2.1.4 Unterschiede in Punktwolken

Durch die unterschiedlichen Aufnahmemethoden ergeben sich natürlich auch charakteristische Unterschiede in den Punktwolken. Ein Vorteil der Rekonstruktion mittels *Dense Image Matching* ist, dass die gewonnenen Punktwolken üblicherweise eine höhere Punktdichte aufweisen. Zusätzlich liefern die 2D-Bilder Farbinformation für die einzelnen Punkte in der Punktwolke (siehe Abbildung 2.4). Gerade die Farbinformation

kann auch für Segmentierungsanwendungen von großem Interesse sein.

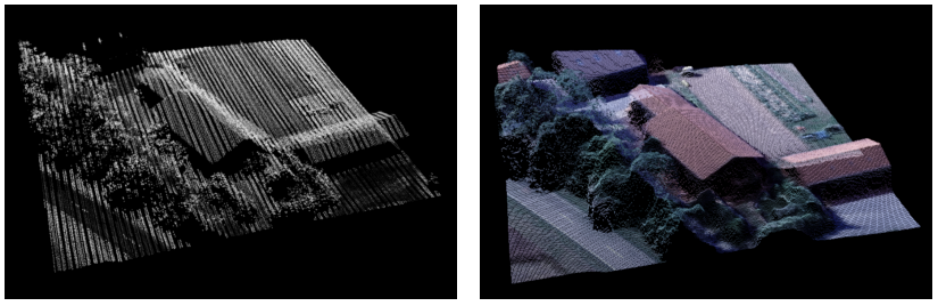

Abbildung 2.4: Vergleich von Punktwolken; Intensitätswerte aus Laserscan und Farbwerte aus DIM [Otepka u. a., 2013]

Die Verteilung der Punkte in der Punktwolke liefert ein weiteres Unterscheidungsmerkmal. Während bei ALS- und TLS-Aufnahmen in der Vegetation die ausgesendeten Laserpulse teilweise durch die Pflanzen dringen und dadurch Informationen aus dem Inneren sowie Geländeinformation gewinnen können, liefern Punktwolken aus *Dense image matching* i.A. ein Oberflächenmodell. Ein Profil der Punktwolken, inklusive der geschätzten Normalvektoren, ist in Abbildung 2.5 zu sehen.

Bei terrestrischen Aufnahmen ist zu beachten, dass die Punktdichte aufgrund von

Abbildung 2.5: Profil mit geschätzten Normalvektoren; Laserscan (li) und DIM (re)[Otepka u. a., 2013]

großen Unterschieden in der Objektentfernung stark variieren kann. Objekte, die vom Messgerät weit entfernt sind, weisen eine niedrigere Punktdichte auf als Objekte die nahe sind. Durch eine möglichst gleichbleibende Flughöhe besitzen die Punktwolken bei ALS homogene Punktdichten.

2.2 Segmentierungsverfahren

Als Segmentierung bezeichnet man das Unterteilen von Punktwolken in Teilbereiche, mit dem Ziel, gleichartige Objekte zu Verbänden zusammenzufügen. Segmentierung ist dabei häufig ein wichtiger Schritt in der Datenkompression und unterstützt Anwendungen zur Objekterkennung und Klassifizierung.

In dieser Arbeit wird ein Segment S_i definiert als Teilmenge der Punktwolke $S_i \subseteq PC$. Die Segmente der Punktwolke erfüllen dabei folgende Eigenschaften:

- Disjunkte Segmente: $S_i \cap S_j = \emptyset$, für alle $i = 1, ..., n$
- Ähnliche Punkte in gleichem Segment:
  $p_s \in S_i, p_t \in S_j : (P(S_i, S_j) == TRUE) \Leftrightarrow i = j$

Das Ähnlichkeits- bzw. Homogenitäts-Kriterium $P(S_i, S_j)$ kann dabei je nach Anwendungsfall unterschiedlich gewählt werden.

Im Folgenden werden die Segmentierungsverfahren *Seeded Region Growing* (2.2.1), *k-means Clustering* (2.2.2) und *Graph Cuts* (2.2.3) kurz beschrieben und verglichen.

2.2.1 Seeded Region Growing

Das *Seeded Region Growing* [Adams und Bischof, 1994, vgl.] beginnt den Segmentierungsvorgang an einem beliebigen Saatpunkt in der Punktmenge. Der Saatpunkt kann dabei als eigenes Segment S_i angesehen werden. Ein Segment S_i wächst nach und nach, wenn benachbarte Pixel das Segmentierungskriterium erfüllen. Das Segment kann jedoch nur durch Punkte wachsen, die noch keinem anderen Segment zugewiesen sind. T ist die Menge aller nicht-zugeordneten Punkte, die die Nachbarschaftsbedingung zu zumindest einem Punkt im Segment erfüllen.

$$T = \{x \notin \bigcup_{i=1}^{n} S_i | N(x) \cap \bigcup_{i=1}^{n} S_i \neq \emptyset\}$$

Ist ein Punkt p_t ähnlich zu einem benachbarten Punkt p_s aus dem Segment S_i - erfüllt er also das Homogenitätskriterium - so wird er zum Segment hinzugefügt. Die Ähnlichkeit wird überprüft mit dem Prädikat P.

$$p_s \in S_i, p_t \in T : (P(p_s, p_t) == TRUE) \wedge p_s \in N(p_t) \Rightarrow S_i = S_i \cup \{p_t\}$$

Jedes Segment kann nur solange wachsen, bis $T = \emptyset$. Danach gilt ein Segment als abgeschlossen. Um den ganzen Datensatz zu Segmentieren, gibt es zwei Ansätze:

A **Parallel** - n Saatpunkte zu Beginn

B **Sequentiell** - Neuer Saatpunkt, wenn anderes Segment abgeschlossen

Bei Ansatz A werden zu Beginn n Saatpunkte zufällig in der Punktwolke verteilt. Alle Saatpunkte führen gleichzeitig und eigenständig eine Segmentierung aus. Erfüllt ein Punkt $p_t \in T$ dabei die Nachbarschaftsbedingung zu mehr als einem Segment, wird

der Punkt zu jenem Segment hinzugefügt, zu dem er die größte Ähnlichkeit besitzt. Ein Vorteil dieses Ansatzes ist es, dass bereits zu Beginn die Anzahl der Segmente (n) bekannt ist und die Segmentierung für alle Segmente parallel ausgeführt werden kann. Der große Nachteil ist jedoch, dass abhängig von den gewählten Saatpunkten große Unterschiede im Segmentierungsergebnis auftreten können. Dabei können in der Segmentierung auch Löcher, also unsegmentierte Bereiche, entstehen, wenn in einem potentiellen Segment zu Beginn kein Saatpunkt liegt.
Der zweite Ansatz (B) wählt einen zufälligen Saatpunkt in der Punktwolke aus. Das Segment wächst dann so lange bis $T = \emptyset$. Danach wird aus allen noch nicht prozessierten Punkten ein neuer zufälliger Saatpunkt für ein neues Segment gewählt. Dadurch werden alle Punkte in der Punktwolke segmentiert. Die Segmente weisen jedoch große Unterschiede in Form und Größe auf. Im Extremfall liefert die Segmentierung Segmente mit nur einem Punkt. Um dem entgegenzuwirken wird häufig eine minimale Segmentgröße definiert. Abbildung 2.6 zeigt die Funktionsweise von sequentiellem Region Growing schematisch.

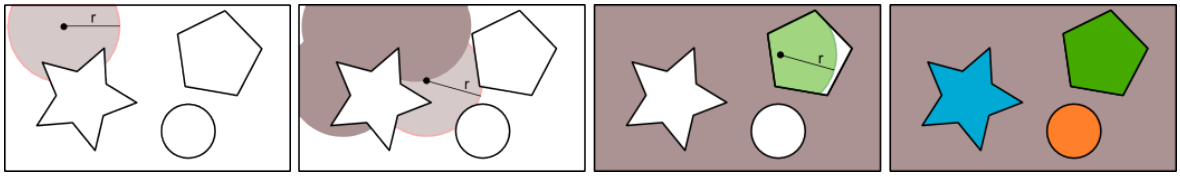

Abbildung 2.6: Seeded Region Growing

2.2.2 k-means Clustering

k-means Clustering [MacQueen, 1967] ist ein Verfahren, das genutzt wird um einen n-dimensionalen Raum in k Segmente aufzuteilen. Das Verfahren liefert dabei Segmente mit minimaler Varianz innerhalb eines Segmentes.
Das k-means Clustering besteht im Wesentlichen aus drei Schritten [Achanta u. a., 2012, vgl.]:

- Initialisierung
- Zuweisung
- Update

Im *Initialisierungsschritt* werden k Cluster gleichmäßig in der Punktwolke verteilt. Um zu verhindern, dass ein initiales Cluster-Zentrum am Rand eines Segmentes liegt, wird häufig dessen Nachbarschaft analysiert und gegebenenfalls ein geeigneterer Startpunkt in der Nachbarschaft gewählt.

Im *Zuweisungsschritt* wird für jeden Punkt im Raum anhand eines Kriteriums (z.B. minimale Distanz zum Cluster-Zentrum; *Euclidean-Distance Clustering*) das zugehörige Cluster-Zentrum bestimmt.
Nach der Zuweisung aller Punkte zu einem der k Cluster-Zentren, werden im *Update-Schritt* alle Cluster-Zentren neu berechnet. Die neuen Cluster-Zentren liegen danach wieder in den Mittelpunkten (Schwerpunkten) aller Punkte eines Clusters. Zusätzlich zu den neuen Cluster-Zentren werden auch Residuen zwischen den Positionen der alten und neuen Cluster-Zentren berechnet.
Zuweisungs- und Update-Schritt werden so lange wiederholt, bis die Residuen konvergieren. In Abbildung 2.7 ist ein k-means Clustering Vorgang mit 3 Wiederholungen abgebildet. Die initialen Cluster-Zentren wurden dabei zufällig gewählt.

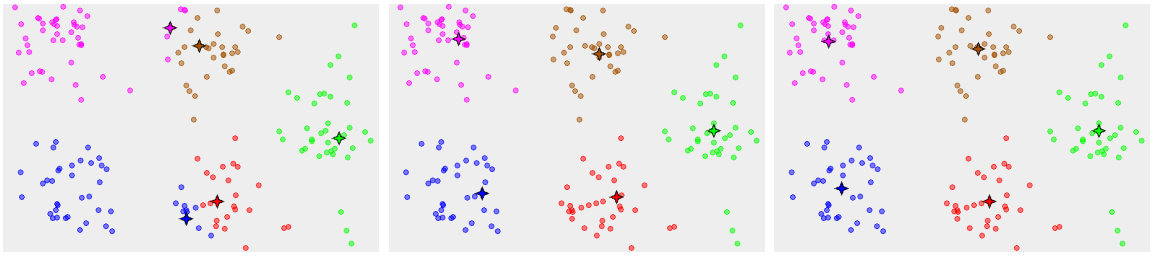

Abbildung 2.7: k-means Clustering - 3 Wiederholungen [E.M. Mirkes, K-means and K-medoids applet. University of Leicester, 2011]

2.2.3 Graph Cuts

Bei graphenbasierter Segmentierung (*Graph cuts*) wird jedes Segment S_i durch einen ungerichteten Graphen $G_i = (V, E)$ repräsentiert. Jeder Punkt der Segmentierung entspricht einem Knoten $v_i \in V$ im Graph. Benachbarte Punkte werden durch gewichtete Kanten $(v_i, v_j) \in E$ verbunden. Das Gewicht der Kante $w(v_i, v_j)$ beschreibt dabei, wie stark sich die verbundenen Punkte bezüglich des Homogenitätskriteriums unterscheiden (*dissimilarity*). Das Ziel des graphenbasierten Ansatzes der Segmentierung ist es, anhand eines Kriteriums gleichartige Punkte in einem Graph zu vereinen und eine Grenze (*cut*) zwischen verschiedenartigen Segmenten zu finden. [Felzenszwalb und Huttenlocher, 2004, vgl.]
Für die Berechnung des *cut* gibt es eine Vielzahl an Verfahren. Die Größe des *cut* kann dabei berechnet werden als die Summe der Gewichte aller Kanten, die durch den *cut* gelöscht worden sind:

$$cut(S_1, S_2) = \sum_{v_1 \in S_1, v_2 \in S_2} w(v_1, v_2)$$

Dabei gilt wieder, dass $S_1 \cap S_2 = \emptyset$. Die Wahl des *cut* ist üblicherweise ein globales Kriterium. Neben dem *Minimum Cut*-Kriterium, bei dem der *cut* global minimiert wird, wurden weitere Verfahren (wie z.B. *Normalized Cut*) entwickelt. [Shi und Malik, 2000, vgl.]
Bei der graphenbasierten Segmentierung werden die Segmente häufig mit einem minimalen Spannbaum (engl. *Minimum Spanning Tree*) dargestellt und verwaltet. Der *Minimum Spanning Tree* ist jener Teilgraph eines Graphen, der alle Knoten im Graphen mit einer minimalen Summe an Kantengewichten verbindet. In Abbildung 2.8 sind zwei Segmente durch deren *Minimum Spanning Tree* repräsentiert.

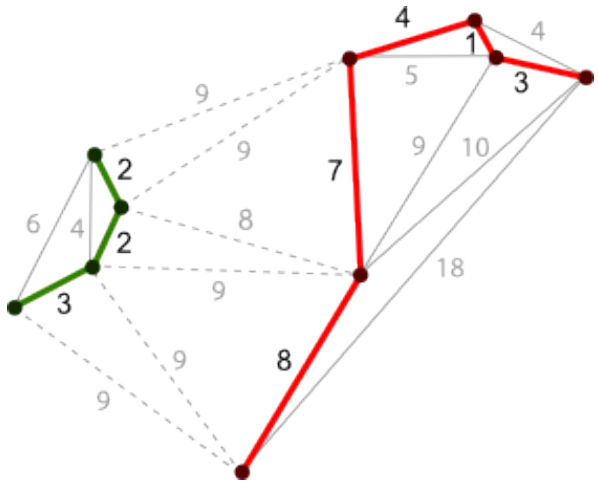

Abbildung 2.8: Graph cuts - Minimum Spanning Tree der Segmente

2.3 Datenmanagement

Die große Anzahl der mit terrestrischen Laserscanning oder ALS gemessenen Punkte stellt die verarbeitende Software vor einige Herausforderungen. Die Objektpunkte sollen möglichst schnell verarbeitet werden, wofür aber nur begrenzt Arbeitsspeicher und CPU-Rechenleistung zur Verfügung steht. Für eine möglichst effiziente Verarbeitung von Punktwolken werden daher eine geeignete Datenstruktur für die Speicherung und effiziente Verarbeitungs-Algorithmen benötigt. Bevor ich auf aktuelle Methoden zur Parallelisierung der Segmentierung eingehe, möchte ich daher auch einen kurzen Blick auf die Datenstruktur von Punktwolken machen. [Otepka u. a., 2013, vgl.]

2.3.1 Datenstruktur der Punktwolke

In dieser Arbeit wird eine Punktwolke PC als Menge von Punkten $p_i \in PC$, mit $i = 1, ...n$, im dreidimensionalen kartesischen Raum definiert. Jeder Punkt p_i wird

durch drei Koordinaten, $(x, y, z)^T \in \mathbb{R}^3$ repräsentiert und besitzt meist zusätzliche Attribute $a_{j,i}$, mit $j = 1, ..., m$ als Attributnummer des Punktes i. Häufige Attribute sind Normalvektorkomponenten und Farbwerte in den Kanälen R, G, B und nIR. Bei Full Waveform (FWF) Laserscannern können auch radiometrische Parameter wie der *Rückstreukoeffizient* und geometrische Parameter wie etwa die *Echobreite* abgeleitet werden. Durch die Attribute erweitert sich die Dimension der Punkte auf $3 + m$ und jeder Punkt p_i auf einen Vektor $(x, y, z, a_{1,i}, ..., a_{m,i})^T$.
Häufig betreffen Abfragen und Berechnungen in Punktwolken eine Menge von benachbarten Punkten des Punktes p_i. Die Nachbarschaft ist immer eine Teilmenge der Punktwolke und daher definiert als $NH(p_i) \subseteq PC$. Bei einer Punktwolke mit mehreren Millionen von Punkten ist es notwendig die Nachbarschaftsabfrage möglichst effizient durchzuführen. Dadurch sollen auch darauf aufbauende Berechnungen effizient ausführbar bleiben. Für die Zugriffe auf geometrische Objekte werden dafür räumliche Indizes verwendet. Da nicht jeder räumliche Index für alle Geometrietypen gleich gut geeignet ist, wurden zahlreiche Strategien (z.B. R-Tree, Quadtree, kd-Tree, etc.) für unterschiedliche Geometrietypen und Anwendungen entwickelt. Während sich der R-Tree sehr gut für die Speicherung von Polygonen und Linien eignet, besitzt der kd-Tree deutliche Vorteile bei der Verwaltung von Einzelpunkten.

Räumlicher Index - kd-tree Eine Speicherstruktur, die einen schnellen Index für Punktwolken liefert und damit eine effiziente Nachbarschaftssuche ermöglicht, ist der kd-tree (k-dimensionaler Baum) [Bentley, 1975, vgl.]. Der kd-tree ist ein Binärbaum, der Punkte im k-dimensionalen Raum verwalten kann. Jeder Knoten im Baum repräsentiert einen k-dimensionalen Bereich des Raumes und teilt diesen Raum in zwei achsparallele Teilbereiche (siehe Abbildung 2.9 - Anmerkung: Die Strichstärke der Trennlinien nimmt mit der Tiefe des Knotens im Baum ab). Der Wurzelknoten (*root*) repräsentiert also die gesamte Datei. Ist der Knoten kein Blatt, so besitzt er einen Kinderknoten in einem oder beiden Teilbereichen. Alle Punkte in den beiden Kindbäumen liegen somit in den bekannten Teilbereichen des Knotens. Der Knoten selbst speichert dabei nur zwei Pointer auf die Kinderknoten. Für den ursprünglichen Aufbau des kd-trees Bentley [1975] galt zusätzlich, dass alle Knoten, die auf dem gleichen Level im Baum sind, den Raum in der gleichen Dimension teilen. Die Dimension wird dabei bestimmt durch den Diskriminator k (siehe Abbildung 2.10). Diese Forderung wird heute jedoch in vielen Implementierungen nicht mehr angewendet. Abhängig von der Verteilung der Punkte im Raum kann dann an einem Knoten eine Dimension, ein- oder mehrmalig, von der Teilung ausgenommen werden (z.B. wenn alle Punkte in der Region die gleiche Z-Koordinate haben).

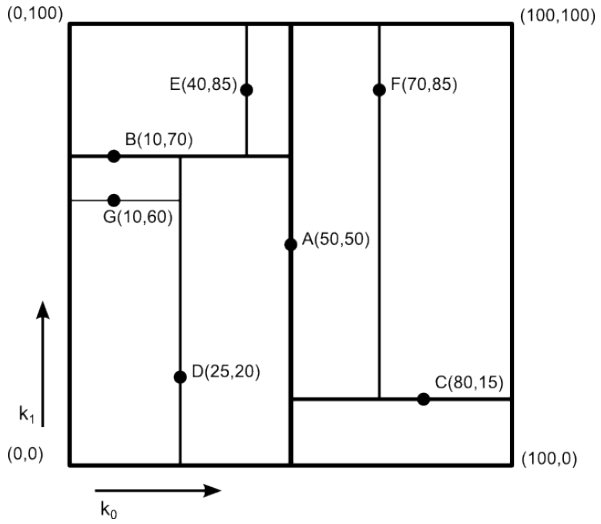

Abbildung 2.9: 2D - Teilbereiche [Bentley, 1975]

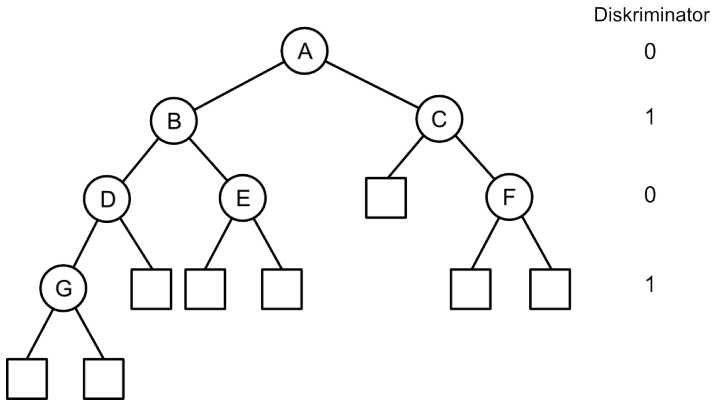

Abbildung 2.10: 2D - Baum [Bentley, 1975]

Punktsuche im Raum Der kd-tree ermöglicht eine effiziente Suche von Punkten im Raum. Der Suchalgorithmus wird dabei rekursiv auf den Knoten ausgeführt. Für jeden Knoten können die geometrischen Grenzen (*boundaries*) der beiden Teilbereiche bestimmt werden. Sind beide Teilbereiche vollständig im Suchraum enthalten, so wird der Knoten markiert. Alle markierten Knoten liefern damit die Punkte im Suchraum. [Teutsch u. a., 2011, vgl.]

2.3.2 Datenprozessierung

Für eine Prozessierung von großen Punktwolken ist es einerseits wichtig, die Anzahl der laufzeitintensiven Lese- und Schreibvorgänge auf der Festplatte möglichst gering zu halten. Auf der anderen Seite ist es aber nicht möglich die gesamte Punktwolke dauerhaft im Arbeitsspeicher zu behalten. Im Rahmen der Arbeit wird daher mit einem zweistufigen Index gearbeitet. Die Punkte werden mit einem Kachel-basierten Index (LEVEL-0) persistent auf der Festplatte gespeichert. Werden Punkte innerhalb einer Kachel für die Prozessierung benötigt und verwendet, so wird mit ihnen *on-the-fly* ein kd-tree (LEVEL-1 Index) im Arbeitsspeicher aufgebaut. Die auf diesem Konzept basierende Datenverarbeitung übernimmt die an der TU Wien entwickelte Software OPALS (Orientation and Processing of Airborne Laser Scanning). [Otepka u. a., 2012, vgl.]

2.3.3 OPALS Data Manager

OPALS ist eine modulbasierte Software für ALS-Daten, bei der der Anwender die einzelnen Module zu flexiblen Prozessierungsketten zusammenfügen kann. Jedes OPALS Modul kann dabei über die Windows Kommandozeile, ein Python Modul oder eine C++ API aufgerufen werden. Das Kernstück vieler OPALS Module ist der OPALS Data Manager (ODM). Der ODM ist für die effiziente und dynamische Verwaltung von Punktwolken zuständig und liefert dabei die Methoden für räumliche Abfragen (Nachbarschaft-Suche). Wie bereits weiter oben erwähnt, wird dafür ein zweistufiger Index verwendet. Der Vorteil des zweistufigen Index ist, dass nur der Kachel-basierte Index auf der Festplatte gespeichert wird und dadurch Speicherplatz eingespart werden kann. Der on-the-fly Index (kd-tree) kann je nach Anwendungsfall flexibel erstellt werden.
Ein weiterer wichtiger Bestandteil von ODM ist die Attributverwaltung. Mit *opalsAddInfo* können eine Reihe von vordefinierten Attributen, aber auch benutzerdefinierte Attribute, zu einer Liste von Punktattributen dynamisch hinzugefügt werden. Folgen-

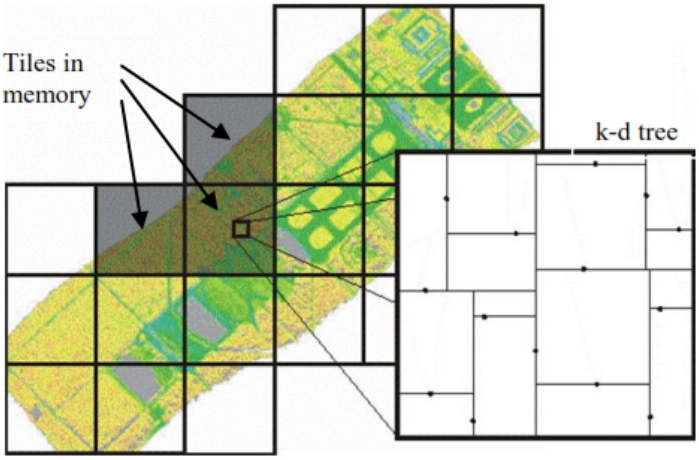

Abbildung 2.11: OPALS Data Manager - Zweistufiger Index [Otepka u. a., 2012]

de Attribute sind, neben vielen weiteren, in der OPALS Programm-Dokumentation festgelegt: [1]

- Amplitude (float)
- Farb-Kanäle
  - Blue (unsigned short)
  - Green (unsigned short)
  - Red (unsigned short)
  - InfraRed (unsigned short)
  - UltraViolet (unsigned short)
- EchoRatio (float)
- EchoWidth (float)
- Normalvektorkomponenten
  - NormalX (float)
  - NormalY (float)
  - NormalZ (float)
- SegmentID (unsigned short)

[1] `http://geo.tuwien.ac.at/opals/html/ref_odm.html`

Die mit den Punkten im *on-the-fly* aufgebauten kd-tree verknüpften Attribute können während den Berechnungen dynamisch geändert und gelesen werden, ohne dass Information auf die Festplatte geschrieben wird. Der Schreibvorgang für persistente Informationen erfolgt üblicherweise nach Abschluss aller Berechnungen.

2.4 Parallelisierung von Segmentierungsverfahren

Große Punktwolken können häufig nicht vollständig in den Arbeitsspeicher geladen werden. Die Punktwolke kann aber, wie im vorherigen Abschnitt beschrieben, in kleinere Teilpunktwolken (Kacheln, engl. Tiles) aufgeteilt werden. Die Kachel-Größe kann so gewählt werden, dass der Arbeitsspeicher möglichst gut genutzt wird. Für die Segmentierung der Punktwolke bedeutet das, dass jede Kachel unabhängig von den anderen geladen und verarbeitet wird.
Das Kacheln (engl. *Tiling*) kann aber nicht nur genutzt werden um Arbeitsspeicher zu sparen, sondern auch auf Systemen mit Multicore-Prozessoren ausgenützt werden. Dabei werden die unabhängigen Berechnungen der einzelnen Kacheln auf mehrere Prozessoren aufgeteilt und *parallelisiert* ausgeführt.

Um aus den Kacheln wieder einen zusammenhängenden Datensatz zu erhalten, gibt es verschiedene Strategien. Für Chen und Pavlidis [1990] stellen sich dabei zwei grundsätzliche Fragen:

A Wenn wir alle Ergebnisse aus den unabhängigen Berechnungen wieder direkt zusammenfügen, ist dann das so ermittelte Endergebnis gleich dem Ergebnis, das erzielt wird, wenn die gleichen Berechnungen auf dem ganzen Datensatz erfolgen?

B Gibt es, wenn A nicht erfüllt ist, einen Weg die Tiling- oder Zusammenführungsstrategien so zu ändern, dass die Ergebnisse annähernd gleich sind?

Für einen Großteil an Berechnungen werden sich die Ergebnisse in A unterscheiden. Vor allem an den Grenzen der Kacheln wird es größere Unterscheidungen geben. Um auf die in B gestellte Frage eine positive Antwort zu erhalten, gibt es verschiedene Verfahren die Teilergebnisse wieder zusammenzufügen. Häufig wird von *vernähen (engl. seaming)* oder *verschmelzen (engl. merge)* gesprochen.

Kapitel 3

Konzept

Im Rahmen dieser Arbeit wurde ein Konzept für eine effiziente Segmentierung von großen Punktwolken entwickelt. Das Konzept basiert auf einem *Split-and-Merge* Verfahren. Wie im vorherigen Kapitel bereits erwähnt verwaltet der OPALS Data Manager (ODM) die Daten in Kacheln (engl. Tiles). Im ersten Schritt wird die Punktwolke in den ODM importiert. Der verwendete LEVEL-0 Index unterteilt die Punktwolke dabei in Kacheln. Im nächsten Schritt erfolgt eine unabhängige Segmentierung in allen Kacheln. Abschließend werden die Punkte aus den segmentierten Kacheln wieder zu einer Punktwolke vereint und dabei werden gleichartige Segmente aus benachbarten Kacheln zusammengeführt.
Im diesem Kapitel wird erklärt, wie die Segmentierung mit *Seeded Region Growing* auf den einzelnen Kacheln erfolgt (Abschnitt 3.1). Um gleichartige Segmente aus benachbarten Kacheln anschließend wieder zu vereinen, müssen Nachbarschaftsbeziehungen zwischen den Kacheln definiert werden. Dazu wurde in dieser Arbeit ein Konzept für die Definition von Nachbarschaftsbeziehungen zwischen Kachel-Regionen entwickelt, welches in Abschnitt 3.2 beschrieben ist.
Außerdem beinhaltet dieses Kapitel ein Verfahren für das Zusammenführen (*Merge*) der Ergebnisse aus den einzelnen Kacheln. *Das dabei angestrebte Ziel ist, dass diese Ergebnisse den Ergebnissen einer Segmentierung auf der gesamten Punktwolke entsprechen.*
Das Kapitel wird abgeschlossen mit Anmerkungen zur Implementierung des Konzeptes.

3.1 Segmentierung auf Kachel-Ebene

Die unabhängige Segmentierung der Punkte in den Kacheln erfolgt mit *Seeded Region Growing* (siehe Kapitel 2.2.1). Dafür wird zuerst mit allen Punkten, die in einer Kachel liegen, ein kd-tree im Arbeitsspeicher aufgebaut. Die Eigenschaften und Vorteile

des kd-trees wurden in Abschnitt 2.3.1 bereits erwähnt. Anschließend beginnt das *Seeded Region Growing* an einem beliebigen Punkt im kd-tree. Bei dem in dieser Arbeit verwendeten Algorithmus des *Region Growing* wächst zu jeder Zeit nur ein Segment pro Kachel. Ist dieses Segment abgeschlossen, wird ein neuer Saatpunkt für ein neues Segment gewählt.
Es werden also alle Punkte p_i mit $i = 1, ..., m$ in der Kachel in beliebiger Reihenfolge durchlaufen. Ist der Punkt p_i noch keinem Segment zugewiesen, wird ein neues Segment S_i mit p_i als Saatpunkt angelegt. Für jeden Saatpunkt p_i wird eine Nachbarschaftssuche durchgeführt. Die Nachbarschaftssuche (siehe weiter unten), liefert dabei eine Liste von Punkten p_n mit $n = 1, ...m$, die ein definiertes Nachbarschaftskriterium erfüllen. Ein benachbarter Punkt p_n wird dann zum Segment S_i hinzugefügt, wenn ein Homogenitätskriterium

$$P(p_i, p_n) == TRUE$$

erfüllt ist. Jeder Punkt p_n, der das Kriterium erfüllt und dadurch zum Segment hinzugefügt wird, wird später ebenfalls Ausgangspunkt (Saatpunkt) des Segments. Damit wächst die Region immer weiter, bis keine neuen Punkte mehr das Homogenitätskriterium erfüllen. Das Segment wird dann abgeschlossen und ein neues Segment mit einem noch nicht prozessierten Saatpunkt p_i begonnen.
Ein detaillierteres Ablaufdiagramm der Segmentierung ist in Abbildung 3.1 zu sehen.

Homogenitätskriterium Zwei benachbarte Punkte werden in das gleiche Segment zusammengefasst, wenn sie ein Kriterium erfüllen. Dieses Kriterium $P(p_i, p_n)$ wird häufig als Ähnlichkeits-, Homogenitäts- oder auch Segmentierungskriterium bezeichnet, da alle Punkte eines Segmentes eine Homogenität (bzw. Ähnlichkeit) bezüglich dieses Kriteriums aufweisen. Damit die Segmentierung mit *Region Growing* für einen Datensatz bei n Durchläufen immer das gleiche Ergebnis liefert, muss dieses Homogenitätskriterium ein symmetrisches Kriterium sein. Das heißt, wenn ein benachbarter Punkt p_n von p_i das Kriterium erfüllt, also:

$$P(p_i, p_n) == TRUE$$

Dann erfüllt der Punkt p_i auch das Kriterium für die Segmentierung mit p_n:

$$P(p_n, p_i) == TRUE$$

Im Rahmen dieser Arbeit wird mit einem einfachen Segmentierungskriterium für beliebige Attribute gearbeitet, bei dem der Absolutbetrag der Differenz des Attributwertes beider Punkte kleiner (oder gleich) einem definierten Grenzwert ϵ ist. Die Ho-

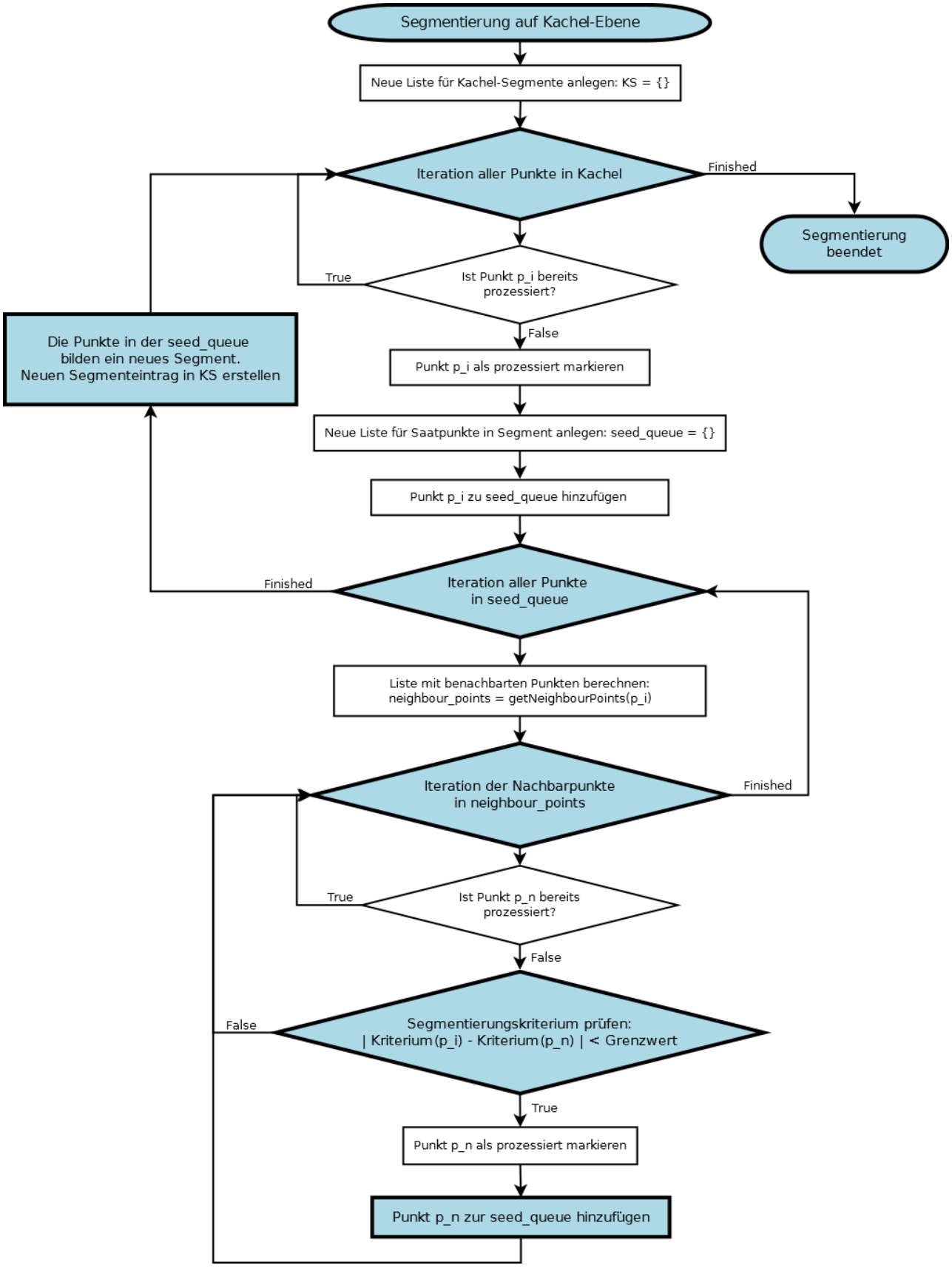

Abbildung 3.1: Berechnen der Segmente für jede Kachel

mogenität wird also bestimmt durch:

$$|Kriterium(p_i) - Kriterium(p_n)| \leq \epsilon \tag{3.1}$$

Dieses Kriterium könnte jedoch auch einfach gegen jedes beliebige andere symmetrische Kriterium ausgetauscht werden.

Nachbarschaftssuche Die Nachbarschaftssuche ist ein wesentlicher Bestandteil von *Seeded Region Growing*. Das wachsende Segment kann dabei immer nur um Punkte, die innerhalb des definierten Suchraumes der Nachbarschaftssuche liegen, erweitert werden. Für eine effiziente Nachbarschaftssuche wird mit den Punkten in der Punktwolke ein kd-tree aufgebaut. Im Rahmen dieser Arbeit werden folgende Nachbarschaftssuchen berücksichtigt und in Kapitel 5 hinsichtlich Laufzeit (5.2) und Segmentierungsergebnissen (5.1) analysiert:

- kNN (k-nearest neighbours)
- 3D Radius-Suche (Kugel)
- 2D Radius-Suche (Zylinder)
- Bounding-Box (Suche innerhalb eines Quaders)

Auch für die Nachbarschaft $NH(p_i)$ gilt, ähnlich wie beim Homogenitätskriterium, dass die Suche symmetrische Ergebnisse liefern muss. Ist ein Punkt p_n in der Nachbarschaft von p_i, also $p_n \in NH(p_i)$, dann muss auch p_i in der Nachbarschaft $NH(p_n)$ sein. Diese Bedingung ist bei der Suche nach kNN nicht unbedingt gegeben, daher kann auch nicht davon ausgegangen werden, dass eine Segmentierung mit kNN-Suche immer die gleichen Ergebnisse liefert.

Parallelisierungsmöglichkeiten Da bei der Segmentierung innerhalb einer Kachel immer nur ein Segment aktiv wächst, ist keine Parallelisierung der Berechnungen pro Kachel möglich. Die Idee dieser Arbeit ist es, die Parallelisierung über eine größere Anzahl an Kacheln, also das Aufteilen der der Punktwolke, zu ermöglichen. Jede Kachel kann dabei unabhängig von den anderen segmentiert werden. Die Anzahl der parallelen Prozesse, auf die eine Segmentierung aufgeteilt werden kann, wird also schon im *Tiling*-Schritt bestimmt. Der Einfluss von unterschiedlichen Kachel-Größen auf die Laufzeit der Segmentierung wird im Kapitel 5.2 untersucht.

3.2 Kachel-Nachbarschaft

Damit die *Split-and-Merge* Segmentierung die gleichen Ergebnisse liefern kann wie die Segmentierung auf dem gesamten Datensatz, müssen benachbarte und gleichartige Segmente aus unterschiedlichen Kacheln im Vereinigungs-Schritt zu einem Segment zusammengeführt werden. Um die benachbarten Segmente als solche erkennen zu können, wird bei diesem Konzept jede Kachel in Regionen unterteilt. Zusätzlich zu den Regionen in den Kacheln wird eine Nachbarschaftsbeziehung zwischen benachbarten Kacheln definiert. Dadurch ergeben sich Nachbarschaftsbereiche, in denen für die Zusammenführung relevante Segmente liegen können.

3.2.1 Kachel-Regionen

Jede Kachel wird im Zuge dieser Arbeit in neun Teilbereiche unterteilt. Neben der inneren Region (0), in der alle Punkte liegen, die für die Nachbarschaftssuche des Vereinigungs-Vorgangs keine Relevanz haben, gibt es acht Randregionen (1)-(8). Die Regionen sind in Abbildung 3.2 zu sehen. Die Größe der Randregionen ergibt sich

Abbildung 3.2: Kachel-Regionen

aus dem Suchradius r der Nachbarschaftssuche in der Segmentierung. Alle Punkte in den Randregionen haben damit höchstens eine Entfernung r zum Kachelrand. Wird die Segmentierung mit *k-nearest neighbours* Suche ausgeführt, so muss die Größe des Randbereiches so abgeschätzt werden, dass in jedem Fall die kNN in den Randbereichen enthalten sind. Das ist aber nur bei gleichmäßigen Punktdichten möglich. Bei unregelmäßigen Punktverteilungen und kNN Nachbarschaft (=allgemeiner kNN Fall) ist die oben beschriebene Einteilung in Kachel-Regionen nicht mehr möglich. Zwar kann auch in diesem Fall zweistufig segmentiert werden, aber diverse - in dieser Arbeit entwickelte - Optimierungen können nicht mehr angewendet werden. Nachdem für praktische

Anwendungen die räumliche Nähe meist eine fundamentale Rolle spielt, wird dieser allgemeine kNN Fall im weiteren Verlauf der Arbeit nicht weiter ausgeführt.
Für ein Mapping von benachbarten Segmenten (Zusammenführung) kommen also nur Punkte in Frage, die auch bei einer Segmentierung auf der gesamten Punktwolke als benachbarte Punkte segmentiert werden können.

3.2.2 Nachbarschaft für die Zusammenführung

Um für eine Kachel-Region alle benachbarten Regionen zu erhalten, muss zuerst bekannt sein, welche benachbarten Kacheln an die Region angrenzen. Dafür wird die Nachbarschaftsbeziehung aller Kacheln bestimmt. Jede Kachel kennt seine Nachbarkacheln und kann über den in Abbildung 3.3 festgelegten Index (schwarzes Gitter) darauf zugreifen. Diese Achter-Nachbarschaft wird im Weiteren dazu verwendet, um die benachbarten Kachel-Regionen zu finden. Die jeweiligen benachbarten Kachel-Regionen sind in der Abbildung mit gleichen Farben abgebildet. Im Wesentlichen gibt es drei verschiedene Nachbarschaften:

- Eck-Nachbarschaft – **ROT**
- Links-Rechts-Nachbarschaft – **GRÜN**
- Oben-Unten-Nachbarschaft – **BLAU**

Die Definition der Nachbarschaft von Kachel-Regionen wird anhand eines Beispieles erläutert.

Beispiel 1. Werden für die erste Randregion (1) - TOP_LEFT einer Kachel die benachbarten Regionen gesucht, können sie über folgenden Index abgerufen werden:

$$[0, 1] \longrightarrow [[8, 3], [1, 5], [2, 7]]$$

Anders geschrieben hat [Kachel INNER, Region TOP_LEFT] die Nachbarn:

- [Kachel LEFT, Region TOP_RIGHT]
- [Kachel TOP_LEFT, Region BOTTOM_RIGHT]
- [Kachel TOP, Region BOTTOM_LEFT]

Es gibt also ein Mapping zwischen einer Kachel-Region und den benachbarten Kacheln und Kachel-Regionen. Diese Nachbarschaft ist damit die Basis für die Zusammenführung der Segmentierungsergebnisse aus den einzelnen Kacheln. Der Algorithmus für die Zusammenführung ist im folgenden Kapitel beschrieben.

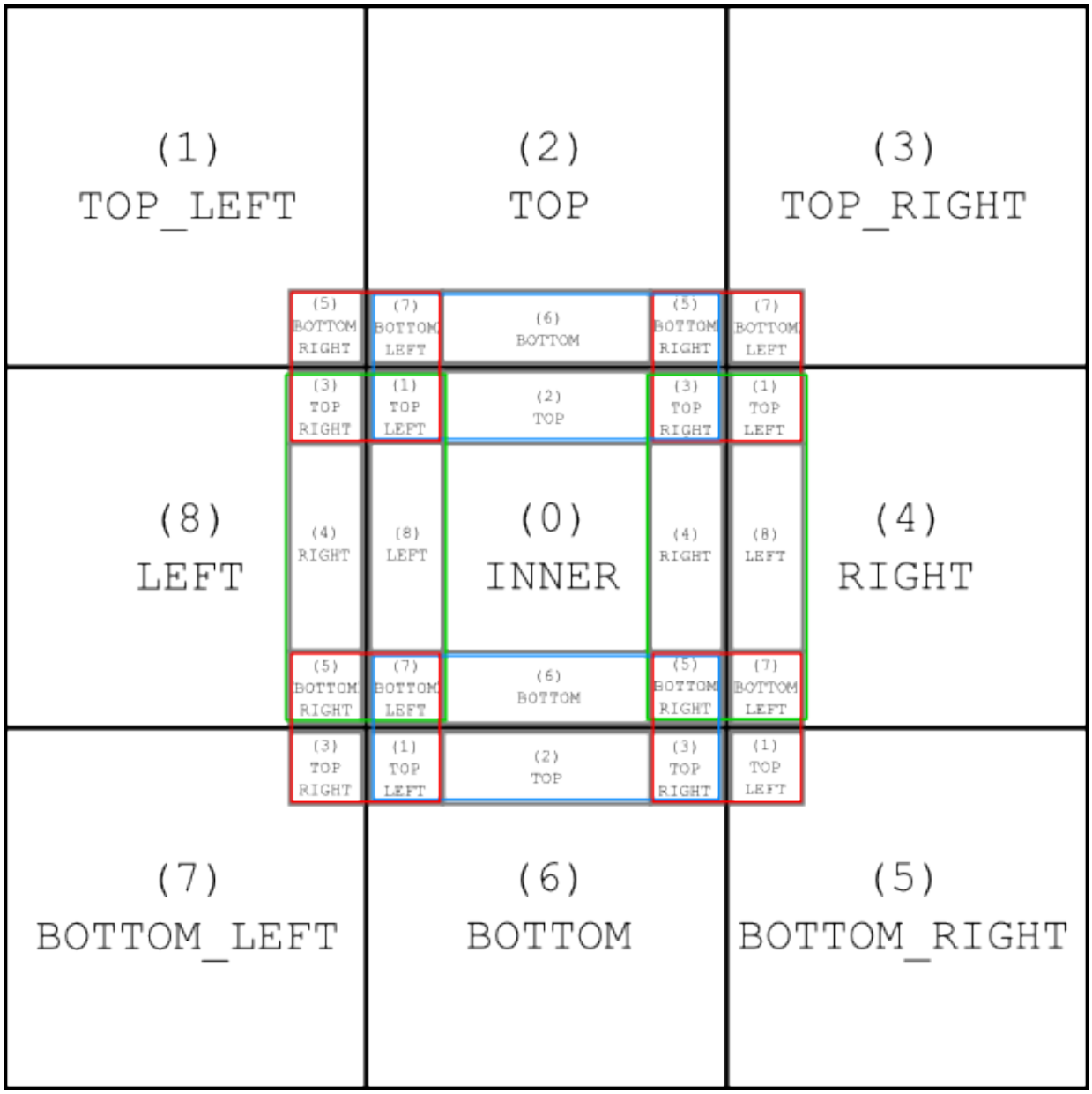

Abbildung 3.3: Nachbarschaft

3.3 Zusammenführung der Kacheln

Die Zusammenführung (engl. Merge) der einzelnen Kacheln ist für das Endergebnis einer auf mehreren Kacheln parallel ausgeführten Segmentierung essentiell. Das Ergebnis der Segmentierung soll nach dem Zusammenführen der Segmente gleich dem Ergebnis sein, das bei der Segmentierung des gesamten Datensatzes errechnet worden wäre. Dazu wurde im Rahmen dieser Arbeit ein Verfahren entwickelt, welches die benachbarten Segmente aus benachbarten Kacheln vereint, wenn sie das gleiche Segmentierungskriterium, wie die Segmente aus der Segmentierung auf Kachel-Ebene erfüllen.
Im Zusammenführungs-Schritt werden dazu alle Kacheln der Punktwolke durchlaufen. In jeder Kachel werden die Randbereiche (Kachel-Region 1-8) unabhängig von einander - sequentiell oder parallel - für die Zusammenführung der Segmente geladen. Für jede Kachel-Region werden zusätzlich die relevanten Nachbarregionen (siehe 3.3) geladen und mit allen Punkten in dieser Nachbarschaft (*Merge-Neighbourhood*) $MNH \subseteq PC$ ein lokaler kd-tree für die Zusammenführung aufgebaut (siehe Abbildung 3.4 - roter Bereich).
Im Prinzip wird für diese Punkte eine zweite Segmentierung mit Punkten aus benach-

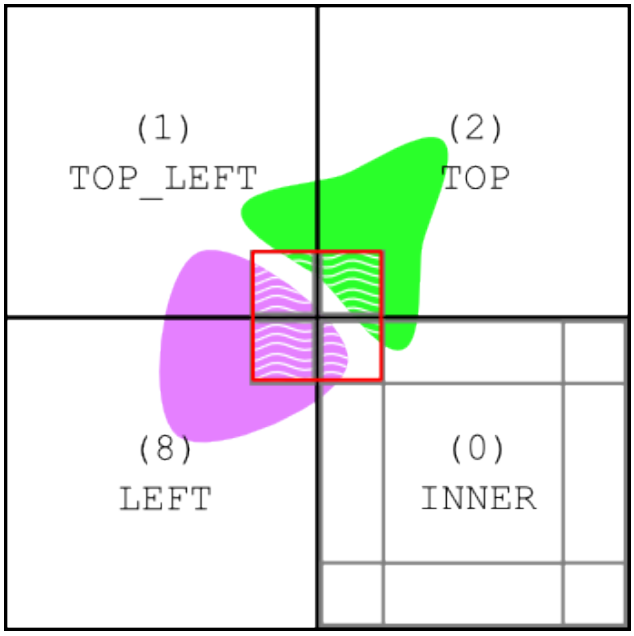

Abbildung 3.4: Relevante Punkte für das Zusammenführen liegen innerhalb der Nachbarschaft

barten Kacheln durchgeführt. Für jeden, im Zusammenführungs-Schritt noch nicht prozessierten, Punkt p_i mit $i = 1, ..., m$ im lokalen kd-tree wird dafür wieder eine Nachbarschaftssuche ausgeführt. Alle Punkte p_n mit $n = 1, ..., m$ in der Nachbarschaft werden mit dem Ausgangspunkt p_i verglichen, wenn folgende Bedingungen erfüllt wer-

den:

- Der Punkt p_n ist noch nicht prozessiert.
- Die Punkte p_i und p_n sind nicht im gleichen Segment.
- Die Segmente der Punkte p_i und p_n besitzen noch kein Mapping zueinander.

Sind diese drei Bedingungen erfüllt, wird ein neues Mapping zwischen den beiden Segmenten erstellt, wenn zusätzlich das Segmentierungskriterium zwischen den Punkten p_i und p_n erfüllt ist. Der genaue Ablauf der Zusammenführung ist in Abbildung 3.5 zu sehen.
Für die Zusammenführung werden somit nur Punkte in den Arbeitsspeicher geladen und prozessiert, die für das lokale Segmentierungskriterium der Nachbarschaft zweier oder mehrerer Kacheln relevant sind. Liegen Punkte in den inneren Kachel-Regionen (0), kann aufgrund der Nachbarschaftsbedingung (z.B. Suchradius) kein direkter Segmentpartner in einer der benachbarten Kacheln liegen. Diese Punkte haben also keine Relevanz für eine Zusammenführung von benachbarten Kacheln. Das Gleiche gilt natürlich auch für Punkte in Randbereichen (1-8), wenn diese am Rand der Punktwolke liegen und dabei keine benachbarten Kacheln haben.
Wird im Rahmen der Segmentierung eine minimale Segmentgröße gefordert, können alle Segmente, die ausschließlich Punkte in einer inneren Kachel-Region besitzen und die minimale Segmentgröße nicht erfüllen, bereits in der Segmentierung verworfen werden. Zu kleine Segmente, die zumindest teilweise in Randbereichen einer Kachel liegen und dadurch Punkte (Kandidaten) für eine Zusammenführung mit anderen Segmenten beinhalten, müssen gespeichert werden. Nach der Zusammenführung können alle restlichen Segmente, welche die minimale Segmentgröße nicht erfüllen können, verworfen.
Im Folgenden wird erklärt, wie das Mapping zwischen zusammengeführten Segmenten verwaltet wird. Zum besseren Verständnis wird es zudem anhand eines Beispiels demonstriert.

Mapping zwischen Segmenten Alle Segmente, die durch die Zusammenführung zu einem Segment vereint werden, sind in einem Mapping-Eintrag zusammengefasst. Jeder Mapping-Eintrag in der Mapping-Liste beinhaltet also zwei oder mehr Segment-IDs. Werden zwei Segmente zusammengeführt, ergeben sich drei mögliche Situationen:

a Hinzufügen eines neuen Mapping-Eintrages

b Hinzufügen einer Segment-ID zu einem bestehenden Mapping-Eintrag

c Zwei vorhandene Mapping-Einträge zu einem verbinden

Diese drei Fälle werden im folgenden Beispiel erläutert.

Beispiel 2. In diesem Beispiel gehen wir davon aus, dass folgendes Mapping zwischen Segmenten bereits existiert:

$[35, 17, 49, 38]$

$[19, 25, 1, 9]$

$[3, 4, 15]$

Die Zahlen in den Mapping-Einträgen stehen dabei für die Segment-IDs. Wir betrachten in diesem Beispiel die oben erwähnten Fälle einer Zusammenführung von zwei Segmenten. Fall a tritt beispielsweise ein, wenn ein neues Mapping zwischen Segment 7 und Segment 13 erstellt werden soll. Weder für Segment 7, noch für Segment 13 sind Einträge in der Liste vorhanden, deshalb wird ein neuer Eintrag hinzugefügt:

$[\mathbf{7}, \mathbf{13}]$

Wird ein neues Mapping zwischen Segment 25 und Segment 5 erstellt (Fall b), wird das Segment 5 einfach zu dem bestehenden Mapping-Eintrag von Segment 25 hinzugefügt:

$[19, \mathbf{25}, 1, 9, \mathbf{5}]$

Wenn beide Segment-IDs bereits in unterschiedlichen Mapping-Einträgen vorhanden sind (Fall c), wie etwa bei einem Mapping von Segment 17 und Segment 3, werden beide Einträge zu einem verbunden:

$[35, \mathbf{17}, 49, 38, \mathbf{3}, 4, 15]$

Nach dem Zusammenführen der Segmente in den Fällen a-c lautet das Mapping also:

$[35, 17, 49, 38, 3, 4, 15]$

$[19, 25, 1, 9, 5]$

$[7, 13]$

Parallelisierungsmöglichkeiten Das *Split-and-Merge* Segmentierungskonzept ermöglicht neben der parallelen Segmentierung auf den Kacheln auch im Vereinigungsvorgang eine Parallelisierung. Jede für die Zusammenführung gewählte Nachbarschaft MNH kann hierbei unabhängig von den anderen bearbeitet werden. Man könnte, zum Beispiel, jeweils für eine Kachel alle noch nicht prozessierten Merge-Nachbarschaften (maximal 8) parallel laden und verarbeiten. Wichtig ist dabei, dass das Mapping zwischen gleichen Segmenten abschließend auf eine eindeutige Segment-ID übergeführt wird.

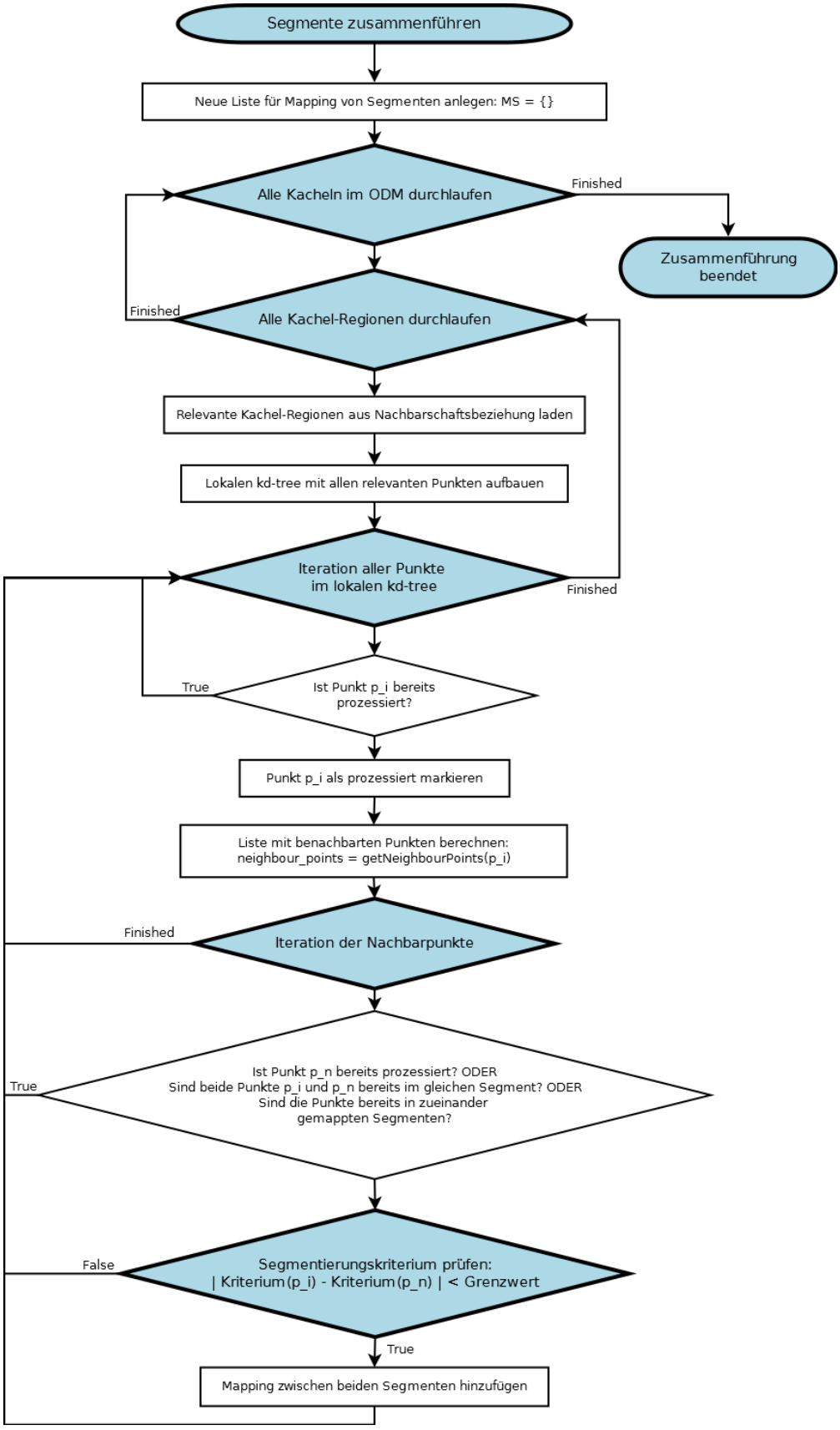

Abbildung 3.5: Zusammenführen der Segmente über Kachelgrenzen hinweg

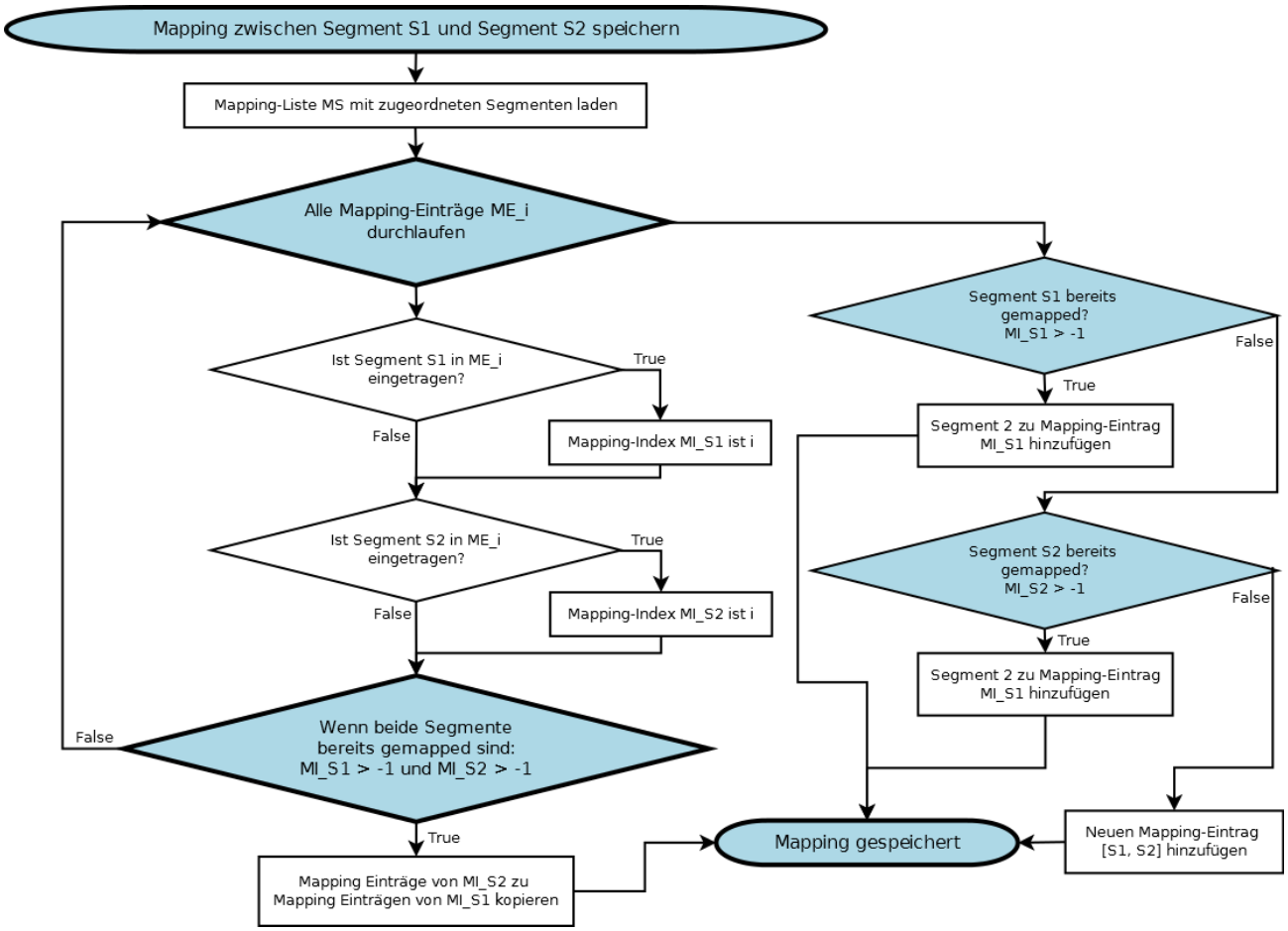

Abbildung 3.6: Ablaufdiagramm für die Verwaltung der zusammengeführten Segmente

3.4 Implementierung

Das Konzept wurde im Zuge dieser Arbeit in einer C++ Anwendung implementiert. Das Programm setzt dabei in der Datenverwaltung auf die C++ API von OPALS. In der Anwendung werden die Punkte mit OPALS Data Manager geladen. Die durch die Segmentierung auf den Kacheln erhaltenen Segment-Objekte beinhalten jeweils eine Liste von zugehörigen Punkten. Dabei ist auch bekannt, in welcher Kachel-Region ein Punkt eines Segmentes liegt. Die Punkte werden über ihre Punkt-ID aus dem ODM identifiziert und verwaltet. Für jedes Segment kann dadurch einfach bestimmt werden, ob und welche Punkte in einer Kachel-Region liegen. Dadurch kann bei der Zusammenführung schnell auf alle Segmentpunkte innerhalb einer Kachel-Region zugegriffen werden. Erst nach der Zusammenführung der Segmente aus den verschiedenen Kacheln wird durch das Segment-Mapping für jeden Punkt eine endgültige Segment-ID bestimmt und diese vom ODM gespeichert. Dieser Schritt beinhaltet den einzigen Schreibvorgang der Anwendung auf die Punktwolke.

Die Implementierung wird in der Arbeit auf Funktion und Laufzeit untersucht. Die verwendeten Datensätze (Kapitel 4) und erzielten Ergebnisse (Kapitel 5) werden in den folgenden Kapiteln der Arbeit erläutert. Für den zahlenmäßigen Vergleich der Ergebnisse werden folgende Statistiken berechnet:

- Anzahl der Segmente
- Mittlere Segmentgröße
- Punkte in größtem Segment
- Anzahl der zu kleinen Segmente
- Anteil der Punkte in Segmenten

Die Laufzeit wird für folgende Berechnungsschritte gemessen:

- Segmentierung auf den Kacheln
- Anteil der Nachbarschaftssuche an der Segmentierung
- Zusammenführung der Segmente aus benachbarten Kacheln

Für die visuelle Überprüfung werden die Punktwolken in ein LAS-Dateiformat exportiert. Für jede Segment-ID wird dabei ein zufälliger RGB-Farbwert bestimmt und gespeichert.

Kapitel 4

Test - Datensätze

Der *Split-and-Merge*-Ansatz für die Segmentierung von großen Punktwolken wurde mit unterschiedlichen Datensätzen auf Funktion und Laufzeit getestet. Die drei verwendeten Datensätze *ring.las*, *Wien_ aoi_ 05.laz* und *Niederrhein_ gebiet3.laz* unterscheiden sich wesentlich in der Punktanzahl und Punktdichte. Während die künstlich generierte Punktwolke in *ring.laz* nur 10000 Punkte besitzt und für den allgemeinen Funktionstest der Segmentierung gedacht ist, besitzen die beiden anderen Datensätze eine viel größere Anzahl an Punkten und sind damit für Laufzeittests geeignet. Die Testdatensätze werden verwendet, um die Segmentierung mit verschiedenen Kachelgrößen, Nachbarschaftssuchen und Homogenitätskriterien zu testen.
In diesem Kapitel werden die verwendeten Datensätze und deren Eigenschaften genauer beschrieben.

LAS - Dateiformat Im der vorliegenden Arbeit werden ALS-Daten im LAS- bzw. LAZ-Dateiformat in den OPALS Data Manager importiert. Dort wird die Punktwolke verarbeitet und später wieder in ein LAS-Format exportiert. LAS ist ein binäres Dateiformat zum Austausch von 3D Punktwolken. Im Gegensatz zu vielen proprietären Dateiformaten kann LAS einfach von unterschiedlichen Programmen geöffnet und verarbeitet werden. Gegenüber ASCII-Formaten bringt das binäre LAS-Dateiformat einen Vorteil in Verarbeitungsgeschwindigkeit und Dateigröße. Dadurch hat sich LAS zu einem Standardformat für Punktwolken entwickelt. Die Open Source Anwendung LASzip generiert aus der LAS-Datei eine verlustfrei komprimierte LAZ-Datei. Für Details bezüglich dem LAS-Dateiformat wird auf die LAS-Spezifikation verwiesen. [1]

[1] `http://asprs.org/a/society/committees/standards/LAS_1_4_r13.pdf`

4.1 Testpunktwolke - Ring

Für die Überprüfung des Segmentierungsverfahrens mit mehreren Kacheln wurde ein einfacher Testdatensatz generiert. Die Test-Punktwolke *ring.las* besteht aus 3 Teilbereichen - einem inneren kreisförmigen Bereich, einem ringförmigen und dem Randbereich. Das Ringsegment ($z = 1$) liegt dabei einen Meter oberhalb der anderen Bereiche ($z = 0$). Die Punkte in diesem Datensatz besitzen keine zusätzlichen Attribute. Eine Segmentierung dieser Punktwolke kann deshalb nur anhand von räumlichen Homogenitätskriterien erfolgen.

Dateiname	ring.las
Punktanzahl	10000
Punktdichte $[pt/m^2]$	1.02
X-Ausdehnung $[m]$	0 – 99
Y-Ausdehnung $[m]$	0 – 99
Z-Ausdehnung $[m]$	0 – 1

Tabelle 4.1: Testdatensatz ring.las

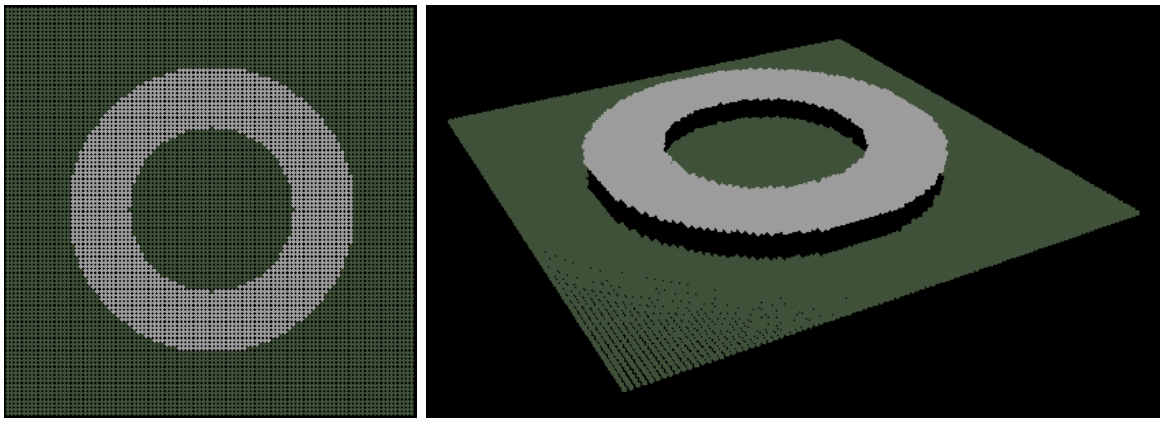

Abbildung 4.1: Testdatensatz ring.las, Farbwerte höhenkodiert

4.2 Testgebiet Wien

Um die Segmentierung neben unterschiedlichen Kachelgrößen auch mit unterschiedlichen Homogenitätskriterien zu testen, wird ein ALS-Datensatz im Wiener Stadtgebiet verwendet. Der Datensatz *Wien_aoi_05.laz* wurde im Rahmen einer im Winter und Frühjahr 2006/07 durchgeführten Befliegung aufgenommen. Dabei wurde ein RIEGL LMS-Q560 Full-Waveform Laserscanner verwendet. Die Aufnahme-Flughöhe betrug etwa 500 m und lieferte eine Punktwolke mit einer mittleren Punktdichte von 15-20 Punkten pro m^2. In Tabelle 4.2 sind die Eigenschaften des verwendeten Testdatensatzes zu sehen. [Rutzinger u. a., 2008, vgl.]

Dateiname	Wien_aoi_05.laz
Punktanzahl	354883
Punktdichte $[pt/m^2]$	35.49
Koordinatensystem	MGI / Austria GK East (EPSG:31256)
X-Ausdehnung $[m]$	-2800.00 – -2700.00
Y-Ausdehnung $[m]$	340950.00 – 341050.00
Z-Ausdehnung $[m]$	78.25 – 135.37

Tabelle 4.2: Testdatensatz Wien_aoi_05.laz

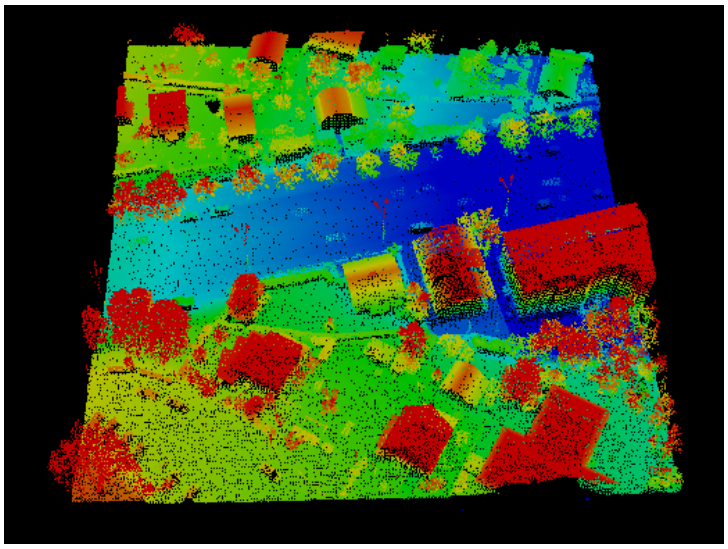

Abbildung 4.2: Testdatensatz Wien_aoi_05.laz, Farbwerte höhenkodiert

4.3 Testgebiet Niederrhein

Der dritte verwendete Testdatensatz wurde in der Nähe von Düsseldorf aufgenommen. Das Testgebiet liegt im Naturschutzgebiet Zonser Grind am Rhein und beinhaltet neben Wasserflächen überwiegend Vegetation und nur wenige kleine Gebäude. Der Datensatz wurde im Rahmen einer Befliegung aus etwa 600 m Flughöhe mit einem Riegl LMS-Q560 Laserscanner aufgenommen. Die Punktdichte dieser Aufnahme liegt bei etwa 6 Punkten pro m^2 und ist damit deutlich niedriger als jene vom Datensatz *Wien_ aoi_ 05.laz*. Durch die viel geringere Punktdichte im Bereich der Wasserfläche des Rheins ergibt sich eine mittlere Punktdichte von 0.31 Punkten pro m^2 über den gesamten Datensatz. Die geringe Punktdichte muss bei der Wahl der Segmentierung beachtet werden. [Waldhauser u. a., 2014, vgl.]

Dateiname	Niederrhein_gebiet3.laz
Punktanzahl	2433807
Punktdichte $[pt/m^2]$	0.31
Koordinatensystem	WGS 84 / UTM zone 32N (EPSG:32632)
X-Ausdehnung $[m]$	349496.00 – 350068.99
Y-Ausdehnung $[m]$	5668773.00 – 5669427.00
Z-Ausdehnung $[m]$	-294.734 – 529.992

Tabelle 4.3: Testdatensatz Niederrhein_gebiet3.laz

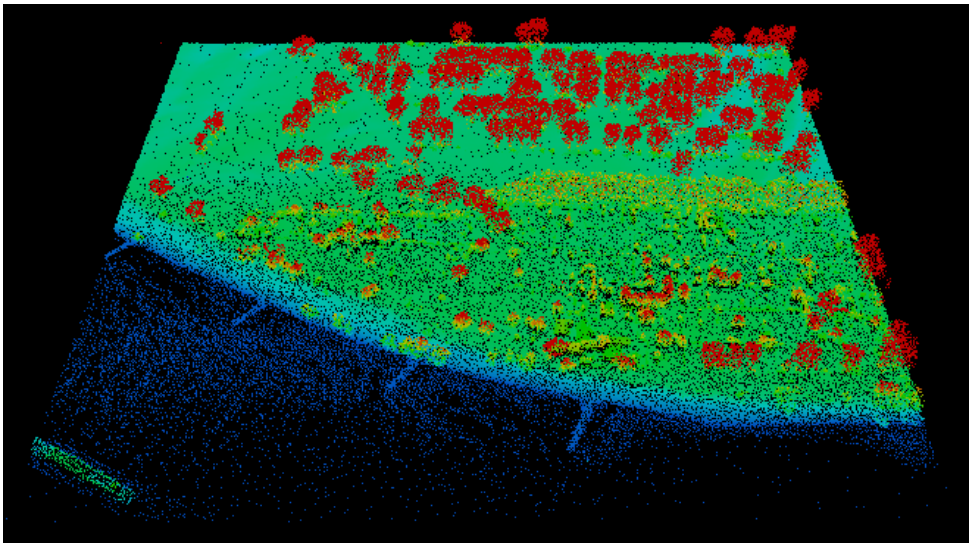

Abbildung 4.3: Testdatensatz Niederrhein_gebiet3.laz, Farbwerte höhenkodiert

Kapitel 5

Test - Ergebnisse

In diesem Kapitel werden Laufzeit und Segmentierungsergebnis bei unterschiedlichen Anwendungsfällen untersucht. Wir erwarten dabei, dass unterschiedliche Kachelgrößen keinen Einfluss auf die Ergebnisse haben und deshalb die gleichen Ergebnisse hinsichtlich Segmentierung und Laufzeit erzielt werden können. Nur dann kann dieses Konzept auch in der Praxis angewendet werden. Zusätzlich wird die Laufzeit bei unterschiedlichen Nachbarschaftssuchen untersucht und der Einfluss verschiedener Segmentierungskriterien auf die Segmentierungsdauer bestimmt. Diese Arbeit beschäftigt sich jedoch nicht mit der Suche nach optimalen Segmentierungskriterien im Bezug auf das Segmentierungsergebnis.
Für alle Berechnungen in diesem Kapitel wird eine minimale Segmentgröße von 50 Punkten definiert. Alle Segmente, die weniger als 50 Punkte beinhalten, werden nicht gespeichert und deshalb in den Abbildungen nicht angezeigt.

5.1 Segmentierungsergebnisse

5.1.1 Einfluss - Kachelgrößen

Testdatensatz ring.las Auf dem künstlich generierten Testdatensatz *ring.las* wird eine Segmentierung mit folgenden Einstellungen getestet:

- Nachbarschafts-Suche: Kugel mit 1 m Radius
- Kriterium: Segmentierung nach z-Koordinaten, Grenzwert 0.01 m

Abbildung 5.1 zeigt die Segmente vor der Zusammenführung der Kacheln. In dieser Abbildung sind die Kachelgrenzen sehr gut zu sehen, da die Segmente aus den einzelnen Kacheln an den Kachelgrenzen abgeschnitten sind. Man sieht sehr gut, dass dadurch

bei abnehmender Kachelgröße die Anzahl der Segmente zunimmt.

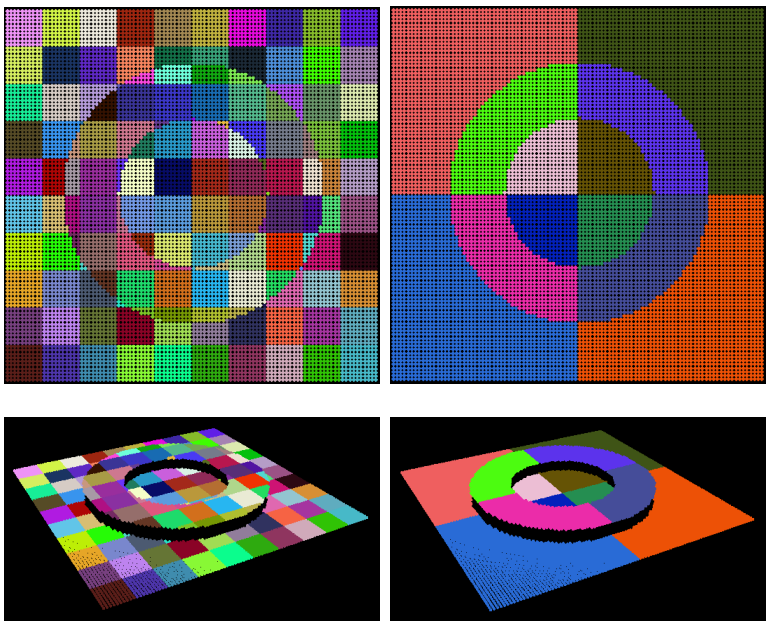

Abbildung 5.1: Segmentierter Testdatensatz ring.las, Kachelgrößen 10m, 50m, vor Zusammenführung

In Abbildung 5.2 sind die zusammengeführten Segmente bei unterschiedlichen Kachelgrößen zu sehen. Die Ergebnisse sind dabei auf allen Datensätzen gleich. Die Segmentierung liefert also, unabhängig von der Kachelgröße, die erwarteten, gleichen Segmente. Die **unterschiedlichen Farben für die Segmente** entstehen durch eine zufällige Berechnung der Farbpalette. Diese führt auch in den weiteren Abbildungen zu unterschiedlichen Farbwerten für die Segmente. In Tabelle 5.1 sind einige Eigenschaften der Segmentierungsergebnisse bei unterschiedlichen Kachelgrößen zu sehen.

Dateiname	Anz. Punkte	Kachel-größe [m]	Anz. Kacheln	Segmente vor Merge	Segmente Final	Mittlere Segmentgr.	Größtes Segment
ring_10m.odm	10000	10	100	140	3	3333.33	6159
ring_50m.odm	10000	50	4	12	3	3333.33	6159
ring_noTiling.odm	10000	-	1	3	3	3333.33	6159

Tabelle 5.1: Einfluss der Kachelgröße auf Segmente in ring.las

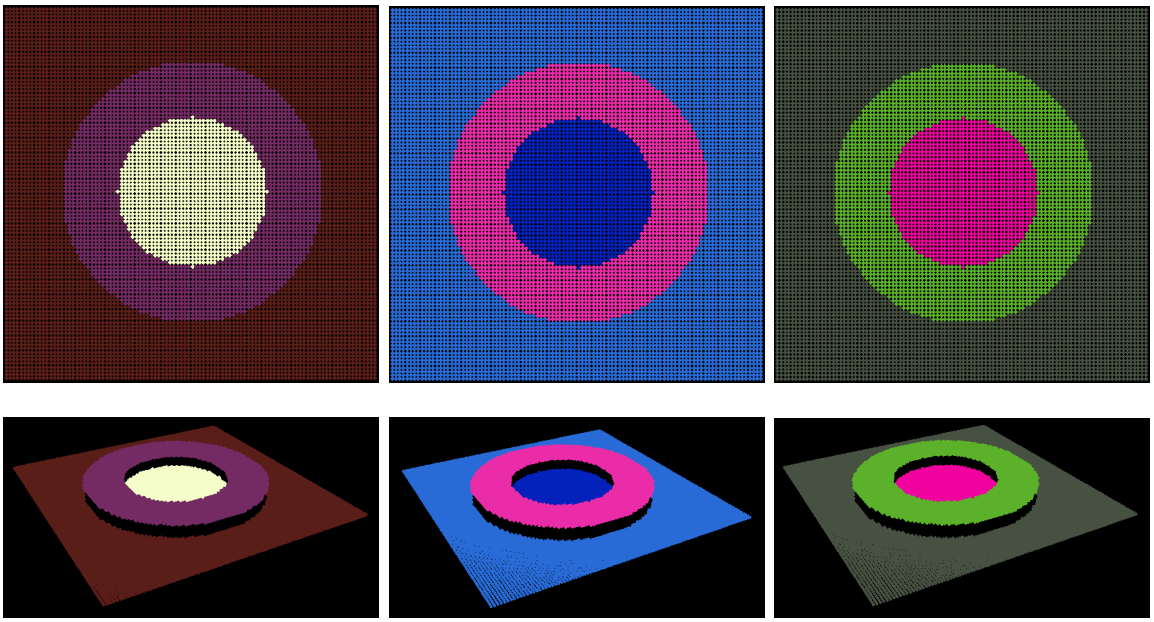

Abbildung 5.2: Segmentierter Testdatensatz ring.las, Kachelgrößen 10m, 50m und Segmentierung ohne Tiling - nach Zusammenführung

Testdatensatz Wien_aoi_05.laz Da die synthetischen Daten in *ring.las* den tatsächlichen Punktwolken von Laserscannern nicht sehr ähnlich sind (z. B. große Unterschiede in Punktmenge und Punktdichte), werden die Segmentierungsergebnisse auch anhand eines größeren ALS-Datensatzes auf Gleichheit überprüft. Die Ergebnisse in Abbildung 5.3 und Tabelle 5.2 zeigen eine Segmentierung mit folgendem Kriterium:

$$|p_i.NormalZ - p_j.NormalZ| \leq 0.01$$

Das heißt, dass zwei benachbarte Punkte, die nicht im gleichen Segment liegen, sich in der z-Komponente des Normalvektors um mindestens 1 cm unterscheiden. Die Nachbarschaftssuche erfolgt, wie auch zuvor, innerhalb einer Kugel mit einem Suchradius von 1 Meter.

Dateiname	Anz. Punkte	Kachel-größe [m]	Anz. Kacheln	Segmente vor Merge	Segmente Final	Mittlere Segmentgr.	Größtes Segment
Wien_10m.odm	354883	10	120	27446	79	3247.32	212012
Wien_25m.odm	354883	25	24	14002	79	3247.32	212012
Wien_50m.odm	354883	50	8	6028	79	3247.32	212012
Wien_75m.odm	354883	75	6	5973	79	3247.32	212012
Wien_100m.odm	354883	100	4	2746	79	3247.32	212012
Wien_noTiling.odm	354883	-	1	79	79	3247.32	212012

Tabelle 5.2: Einfluss der Kachelgröße auf Segmente in Wien_aoi_05.laz

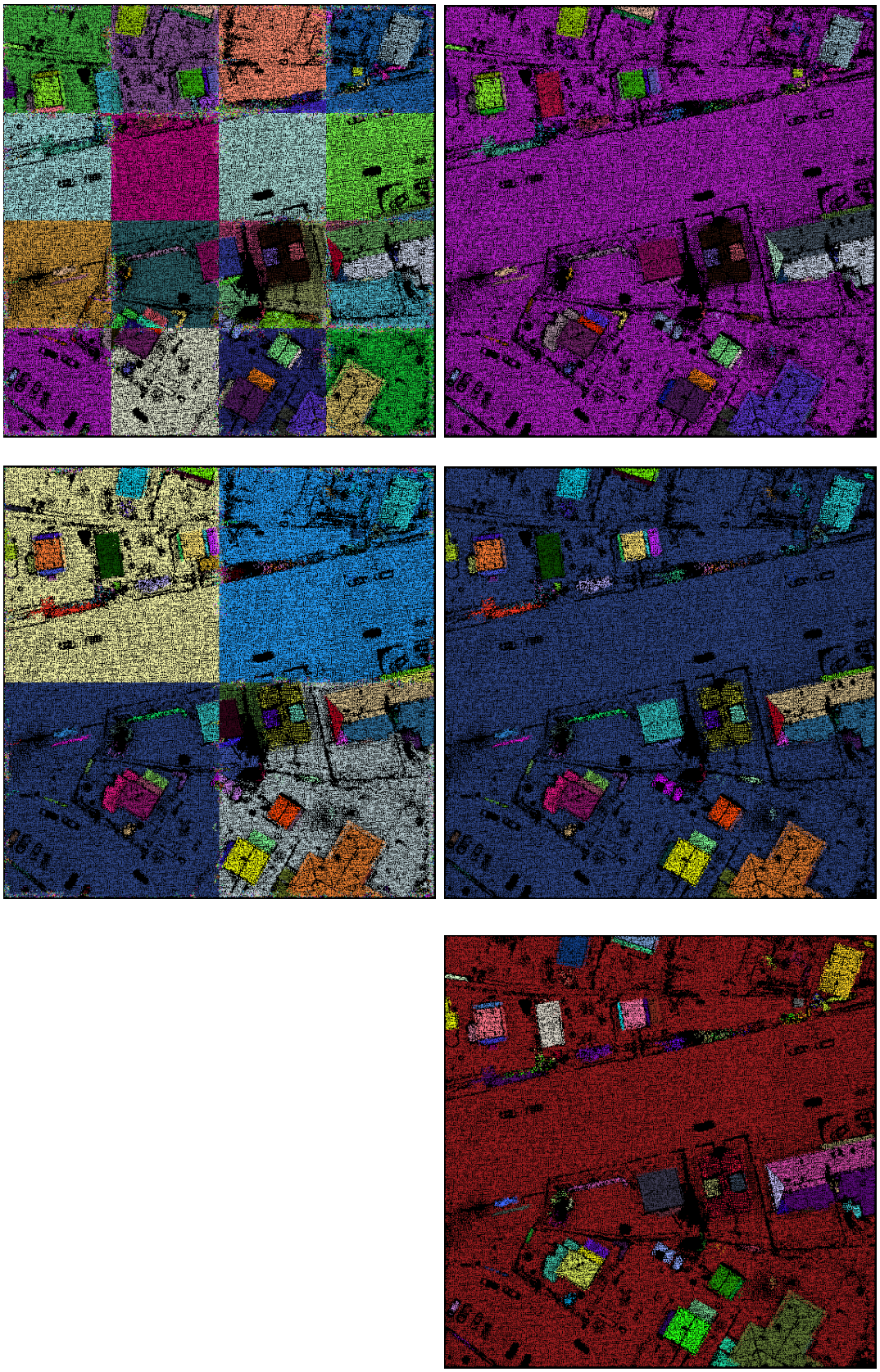

Abbildung 5.3: Vergleich der Segmentierungsergebnisse bei unterschiedlichen Kachelgrößen (25 m, 50 m und ohne Tiling) vor und nach dem Zusammenführen

5.1.2 Einfluss - Wahl der Nachbarschaftssuche

Die Nachbarschaftssuche sollte für die Segmentierung auf jeden Fall so gewählt werden, dass in allen Bereichen des Datensatzes genügend Punkte innerhalb der Nachbarschaft liegen. Wird die Nachbarschaft jedoch zu groß gewählt, werden auch Segmente verknüpft, die weit von einander entfernt liegen. Wenn für den Datensatz *ring.las* beispielsweise ein zu großer Suchradius gewählt wird, werden das innere Kreissegment und das Randsegment aufgrund der gleichen Höhe ($z = 0$) zu einem Segment zusammengeführt (Abbildung 5.4), obwohl sie räumlich durch das Ringsegment getrennt sind. Allgemein kann man sagen, dass jede unterschiedliche Nachbarschaftssuche unterschiedliche Punkte für die Segmentierung zurückliefert und dadurch auch das Segmentierungsergebnis verändern kann. Die nicht-symmetrische *k-nearest-neighbours*-Suche wird dabei auch bei unterschiedlichen Ausgangspunkten der Segmentierung unterschiedliche Ergebnisse liefern (siehe Kapitel 3.1 - Nachbarschaftssuche).
Für das entworfene Konzept für die Berechnung in Kacheln gilt zusätzlich, dass der Suchradius maximal die halbe Größe der Kachel haben kann. Dann würden die Randbereiche die gesamte Kachel abdecken und keine innere Region existieren. In den üblichen Fällen macht es aber ohnehin keinen Sinn, so große Suchradien bzw. so kleine Kachelgrößen zu wählen.

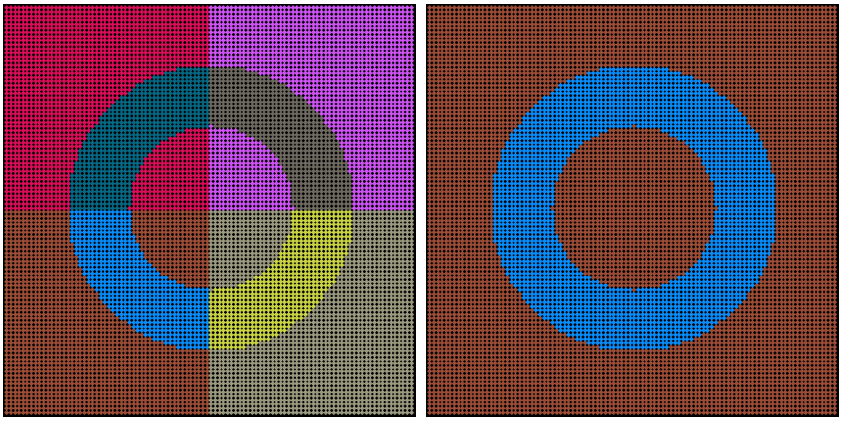

Abbildung 5.4: Radius für Nachbarschaftssuche R=30m, Links vor Zusammenführung, Rechts nach Zusammenführung

5.1.3 Einfluss - Homogenitätskriterium

Schon aus der Definition der Segmentierung geht klar hervor, dass das Homogenitätskriterium der entscheidende Parameter für das Ergebnis der Segmentierung ist. Das Homogenitätskriterium bestimmt, anhand welcher Eigenschaften die Segmente getrennt

oder vereint werden. Bei einem großen Grenzwert ϵ im Homogenitätskriterium wird ein Punkt aus der Nachbarschaft eher zum Segment hinzugefügt, als bei einem kleinen Grenzwert. Die Ergebnisse in Tabelle 5.3 bestätigen dies und zeigen, dass ein großer Grenzwert ϵ zu größeren Segmenten und ein kleiner Grenzwert zu kleineren Segmenten führt. Die Segmentierung wurde durchgeführt auf dem Datensatz *Wien_aoi_05.laz* mit einer Kachelgröße von 50 Metern. Die gewählten Grenzwerte sind 0.001 m, 0.01 m, 0.1 m und 1.0 m für das bereits zuvor verwendete Attribut NormalZ.

Homogenitätskriterium	Segmente Final	Punkte in Segmenten	Mittlere Segmentgröße	Größtes Segment
Differenz in NormalZ ≤ 0.001	57	62.9%	3914.35	188386
Differenz in NormalZ ≤ 0.01	71	85.3%	4261.82	279452
Differenz in NormalZ ≤ 0.1	4	97.5%	86483.8	340387
Differenz in NormalZ ≤ 1.0	1	97.8%	347050	347050

Tabelle 5.3: Einfluss des Grenzwertes im Homogenitätskriterium auf die Segmentierungsergebnisse in Wien_aoi_05.laz, bei einer Kachelgröße von 50 m

Man sieht, dass die mittlere Segmentgröße, sowie die Punktanzahl im größten Segment, mit der Größe des Grenzwertes stark zunimmt. Auf der anderen Seite, nimmt die Anzahl der Segmente mit größer werdenden Grenzwerten ab. Dabei ist auffällig, dass bei ganz kleinen Grenzwerten durch eine festgelegte minimale Segmentgröße (z.B. mindestens 50 Punkte) die Anzahl der Segmente auch wieder abnimmt. Je kleiner die Grenzwerte gewählt werden, umso mehr Segmente können die minimale Segmentgröße nicht erfüllen. Deshalb nimmt die Anzahl der Segmente ab und infolgedessen auch der Anteil der Punkte, die in Segmenten liegen.

Die folgenden Abbildungen zeigen Vergleiche von Ergebnissen auf einem kleinen Ausschnitt aus *Wien_aoi_05.laz* bei unterschiedlichen Grenzwerten mit dem Homogenitätskriterium

$$|p_i.Normalvektor \bullet p_n.Normalvektor| > \epsilon$$

Als Grenzwerte werden 0.9 ($\hat{=}$26 deg), 0.95 ($\hat{=}$18 deg) und 0.99 ($\hat{=}$8 deg) verwendet. Der verwendete Suchradius ist 1 m bei 3D-Suche.

Die Ergebnisse zeigen dabei, dass bei einem niedrigeren Grenzwert (= größerer Winkel zwischen Normalvektoren) der Anteil der Punkte in Segmenten größer ist. Die Punkte werden dabei eher zu größeren Segmenten zusammengefasst. Viele Objekte fallen dadurch beispielsweise in das Bodensegment. Bei 8 deg Differenz zwischen Normal-

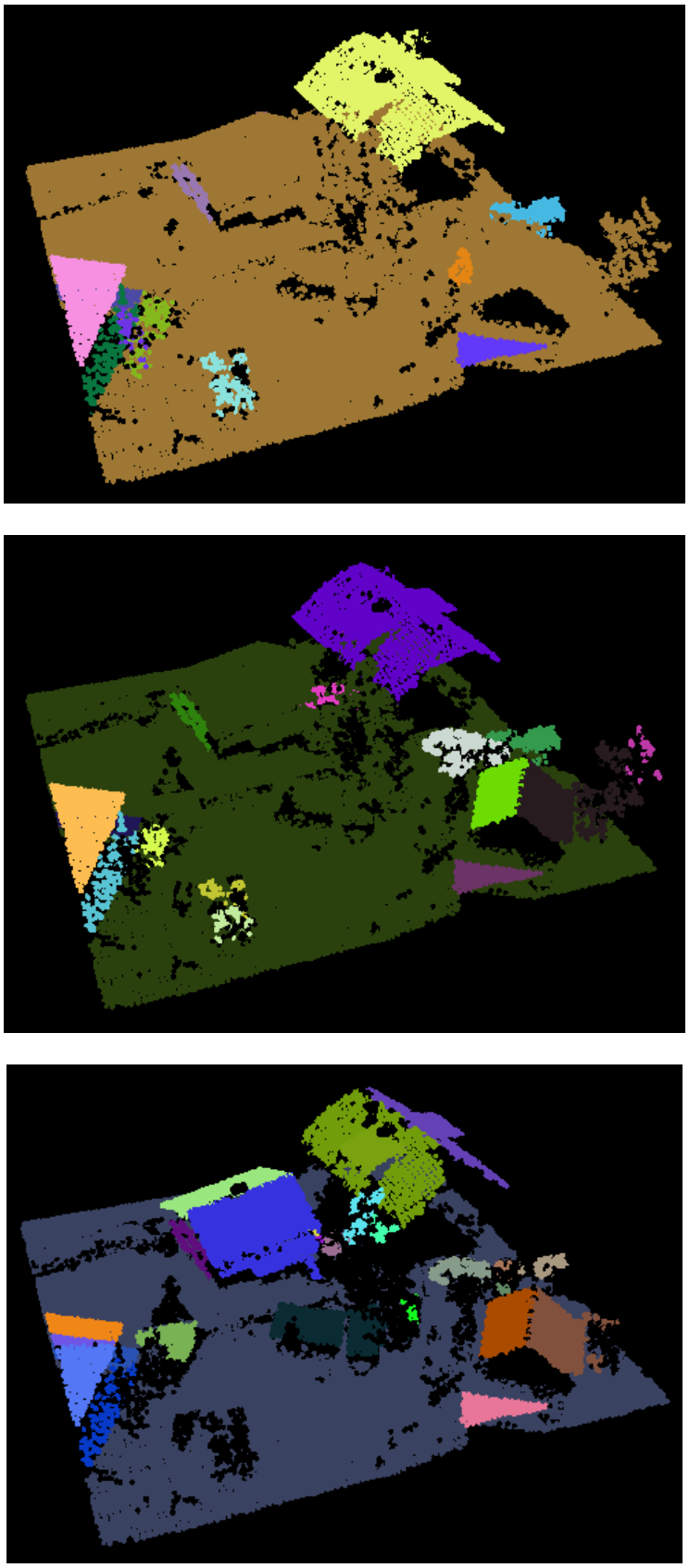

Abbildung 5.5: Vergleich der Segmentierungsergebnisse bei unterschiedlichen Grenzwerten (0.9, 0.95, 0.99) für das Homogenitätskriterium

vektoren benachbarter Punkte (unteres Bild) werden die Objekte (z.B. Dächer) eher in eigene Segmente unterteilt. Dabei können einige Objekte (z.B. Bäume und Sträucher) jedoch die minimale Segmentgröße nicht erfüllen, da sie in viele kleine Segmente aufgeteilt werden. Der Baum am rechten Rand beispielsweise fällt dadurch aus der segmentierten Punktwolke.
Ein Problem des lokalen Homogenitätskriteriums kann ebenfalls am Beispiel des Baumes am rechten Rand erkannt werden. Wird das Homogenitätskriterium nur zwischen zwei Punkten aus benachbarten Segmenten erfüllt, so werden diese Segmente bereits zu einem Segment zusammengefasst. Zwei Segmente können also bereits durch eine kleine *Brücke* zu einem Segment verbunden werden. Der Baum am rechten Rand wird im mittleren Bild dadurch mit dem Dachsegment verbunden. Ein zusätzliches Kriterium könnte diese Segmente auftrennen.

5.2 Laufzeitanalyse

Das Konzept für die Segmentierung von großen Punktwolken basiert auf der Annahme, dass keine signifikanten Laufzeitunterschiede zwischen der Segmentierung des gesamten Datensatzes und der verteilten Segmentierung auftreten. Die Laufzeit-Tests wurden auf einem herkömmlichen Business-Notebook mit folgenden Spezifikationen durchgeführt:

- Prozessor: Intel® Core™ i5 CPU (M560 @ 2.67 GHz)
- Arbeitsspeicher: 4 GB RAM
- Betriebssystem: Windows 8.1 Pro, 64-Bit

Obwohl der Prozessor eine parallele Ausführung der Segmentierung auf 4 Threads ermöglichen würde, wurden die Tests in einem Thread ausgeführt. Eine Parallelisierung könnte sowohl die Segmentierung, als auch die Zusammenführung erheblich beschleunigen. In dieser Arbeit sind wir jedoch am Verhältnis der Laufzeiten von verschiedenen Konfiguration interessiert. Da die Segmentierung mit dem entwickelten Verfahren ohne *Tiling* nicht parallelisiert werden kann, wurde die Möglichkeit zur Parallelisierung auch bei allen weiteren Laufzeittests nicht genutzt, um den direkten Vergleich zu erreichen.

5.2.1 Einfluss - Kachelgrößen

In diesem Abschnitt werden die Laufzeiten der Segmentierung für unterschiedliche Kachelgrößen miteinander verglichen.
Die Ergebnisse für den Datensatz *Wien_ aoi_ 05.laz* sind in Tabelle 5.5, sowie in den Abbildungen 5.6 und 5.7 zu sehen. Jene für den deutlich größeren Datensatz *Niederrhein_ gebiet3.laz* sind in Tabelle 5.6 und in den Abbildungen 5.8 und 5.9. Die Segmentierung wurde mit folgenden Einstellungen durchgeführt:

	Wien_ aoi_ 05.laz	*Niederrhein_ gebiet3.laz*
Homogenitätskriterium	NormalZ	z-Koordinate
Grenzwert	0.01	0.1
Nachbarschaftssuche	Kugel (3D)	Kugel (3D)
Suchradius	1 m	1 m

Tabelle 5.4: Segmentierungseinstellungen für Laufzeittests

Dateiname	Anz. Kacheln	ø Punkte pro Kachel	Laufzeit Segmentierung [s]	Anteil Nachbarschaftssuche [s]	Laufzeit Zusammenführung [s]	Laufzeit Gesamt [s]
Wien_10m.odm	120	2957.36	77	30	93	170
Wien_25m.odm	24	14786.79	84	46	27	111
Wien_50m.odm	8	44360.38	87	44	10	97
Wien_75m.odm	6	59147.17	90	37	14	104
Wien_100m.odm	4	88720.75	91	42	5	96
Wien_noTiling.odm	1	354883.00	90	41	-	90

Tabelle 5.5: Einfluss der Kachelgröße auf die Laufzeit der Segmentierung in Wien_aoi_05.laz

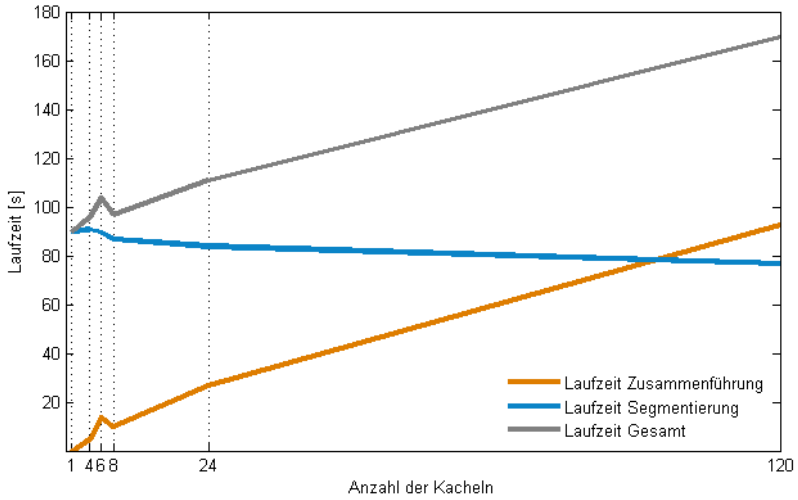

Abbildung 5.6: Laufzeitvergleich, Segmentierung bei unterschiedlichen Kachelgrößen, Wien_aoi_05.laz

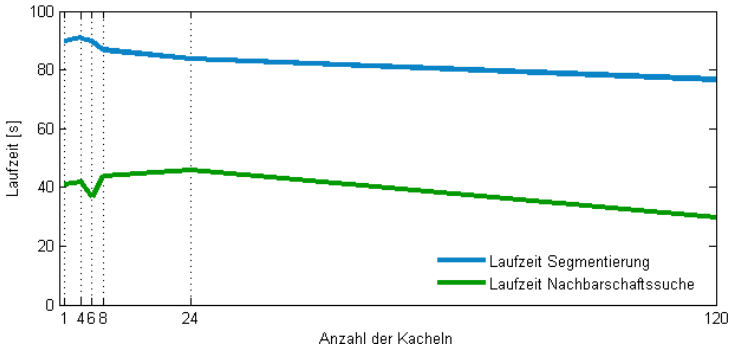

Abbildung 5.7: Anteil der Nachbarschaftssuche an der Laufzeit der Segmentierung, Wien_aoi_05.laz

Dateiname	Anz. Kacheln	ø Punkte pro Kachel	Laufzeit Segmentierung [s]	Anteil Nachbarschaftssuche [s]	Laufzeit Zusammenführung [s]	Laufzeit Gesamt [s]
Niederrhein_75m.odm	90	27042.30	1309	576	1075	2384
Niederrhein_100m.odm	56	43460.84	1234	534	775	2009
Niederrhein_125m.odm	36	67605.75	1310	614	506	1816
Niederrhein_150m.odm	30	81126.90	1317	594	559	1876
Niederrhein_200m.odm	20	121690.35	1259	578	402	1661
Niederrhein_250m.odm	12	202817.25	1316	561	212	1528

Tabelle 5.6: Einfluss der Kachelgröße auf die Laufzeit der Segmentierung in Niederrhein_gebiet3.laz

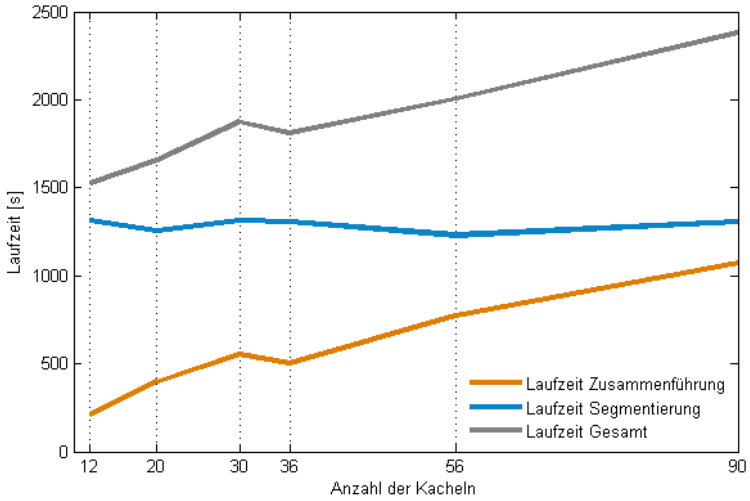

Abbildung 5.8: Laufzeitvergleich, Segmentierung bei unterschiedlichen Kachelgrößen, Niederrhein_gebiet3.laz

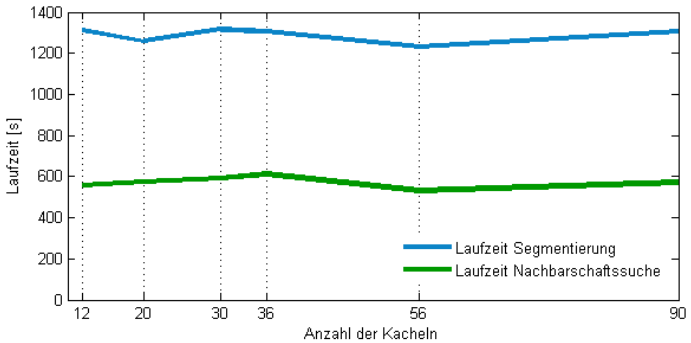

Abbildung 5.9: Anteil der Nachbarschaftssuche an der Laufzeit der Segmentierung, Niederrhein_gebiet3.laz

Laufzeit - Segmentierung Wie in den vorherigen Abbildungen zu sehen ist, haben unterschiedliche Kachelgrößen bei sinnvoller Wahl keinen großen Einfluss auf die Segmentierung der Punktwolke. Die reine Segmentierungslaufzeit auf den Kacheln nimmt sogar leicht ab, je kleiner die Teilpunktwolke ist. Der Grund dafür ist, dass die Dimensionen der Variablen (z.B. *Seed Queue*), welche die Segmente im *Region Growing* verwalten, kleiner bleiben, da weniger Punkte in der Teilpunktwolke liegen.
Wie zu erwarten war, haben die Kachelgrößen bei sinnvoller Wahl auch keinen entscheidenden Einfluss auf die gesamte Dauer der Nachbarschaftssuchen im kd-tree. Die Punktdichte und die Anzahl der Punkte in einer Nachbarschaft bleibt ja für den Großteil der Punkte unverändert. In dieser Laufzeitanalyse zeigt sich, dass im Allgemeinen der Anteil der Nachbarschaftssuche an der Segmentierungsdauer bei 40 - 50% liegt. Was jedoch auffällt ist, dass bei sehr kleinen Kacheln (z.B. 10m) die Nachbarschaftssuche schneller ist. Durch die kleineren Kacheln erhöht sich die Anzahl der Punkte, die an einer Kachelgrenze liegen. Für diese Punkte liefert die Nachbarschaftssuche - weil die Nachbarn abgeschnitten sind - weniger Punkte zurück und benötigt dafür weniger Laufzeit. Dieser für die Segmentierung positive Effekt beeinflusst jedoch die Zusammenführung negativ.

Laufzeit - Zusammenführung In den Abbildungen 5.6 und 5.8 ist ein deutlicher Laufzeitunterschied bei der Zusammenführung der Segmente mit unterschiedlichen Kachelgrößen zu sehen. Die Laufzeit nimmt dabei linear mit der Kachelgröße ab. Der Ausreißer bei $Wien\_75m.odm$ kann durch eine schlechte Wahl der Kachelung begründet werden. Der Grund für die längere Dauer der Zusammenführung bei kleineren Kacheln ist, dass die Anzahl der Segmente und die Anzahl der Punkte, die für den Zusammenführungsschritt in Frage kommen, größer werden. Zum einen liegt es daran, dass Segmente durch die kleineren Kacheln in kleinere Teilsegmente getrennt werden, zum anderen aber hauptsächlich daran, dass bei kleineren Kacheln die relevante Nachbarschaftsfläche für die Zusammenführung größer wird. Eine größere Nachbarschaftsfläche bedeutet wiederum, dass mehr Punkte an der Zusammenführung beteiligt sind. Logischerweise dauert die Berechnung für eine größere Anzahl von Punkten länger.

Die Abschätzung des Einflusses der Kachelgröße auf die Zusammenführung erfolgt anhand eines kurzen Beispieles.

Beispiel 3. Für das Beispiel nehmen wir folgende Konfigurationen für den Vergleich an:

a Segmentierung mit 10 m Kacheln und 1 m Radius für Nachbarschaftssuche

b Segmentierung mit 50 m Kacheln und 1 m Radius für Nachbarschaftssuche

Bei beiden Segmentierungen liegen jeweils jene Punkte, die bis zu einem Meter von den jeweiligen Kachelrändern entfernt liegen, in den für die Zusammenführung relevanten Bereichen. Bei einer 10x10 Meter Kachel liegen somit 36 m^2, also 36% der Kachel in der Nachbarschaftsregion. Bei einer 50x50 Meter Kachel liegen hingegen nur 7.8% der Kachel (196 m^2) in der Nachbarschaftsregion. Es sind also bei Konfiguration a etwa 5 mal so viele Punkte in der Zusammenführung involviert.

Die Nachbarschaftsbereiche für die in diesem Beispiel verwendeten Einstellungen werden in Abbildung 5.10 dargestellt. Die Farbwerte in den Abbildungen sind Resultate aus unterschiedlichen Segmentierungen.

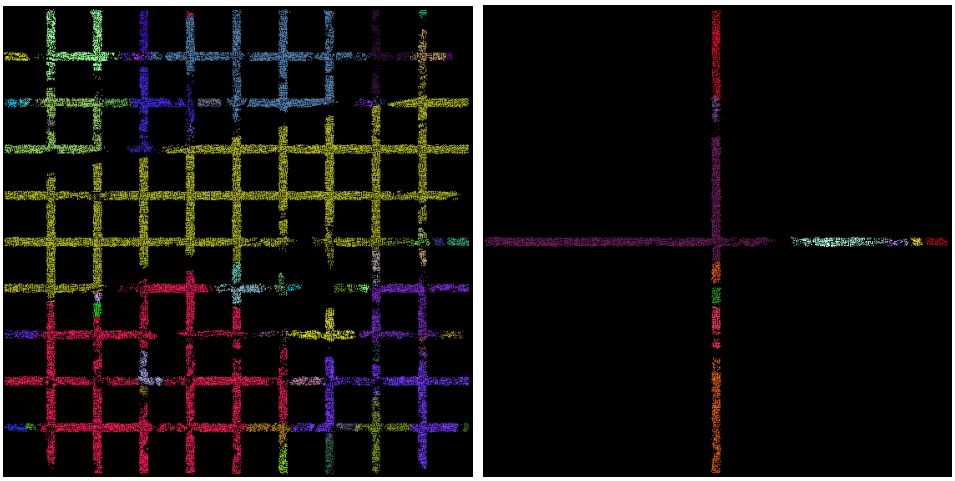

Abbildung 5.10: Vergleich der Nachbarschaftsregionen für die Zusammenführung bei 10 m (re) und 50 m (li) Kacheln mit 1 m Radius

5.2.2 Einfluss - Wahl der Nachbarschaftssuche

Es gibt bei der Segmentierung verschiedene Möglichkeiten, wie die Nachbarschaftssuche durchgeführt werden soll. In diesem Abschnitt werden drei implementierte Suchmethoden - kNN (k-nearest Neighbours), Radius 3D (Kugel), Radius 2D (Zylinder) - auf Laufzeit getestet. Die kNN-Suche wird mit 10, 20, 40 und 60 Nachbarn ausgeführt. Die anderen beiden Suchen werden mit verschiedenen Suchradien (1 m und 2 m) getestet. Die Segmentierung wird auf dem Datensatz *Wien_aoi_05_50m.odm* mit 50 m Kacheln mit dem bereits zuvor verwendeten Homogenitätskriterium

$$|p_i.NormalZ - p_j.NormalZ| \leq 0.01$$

ausgeführt.

Nachbarschaftssuche	Segmente Final	Laufzeit Segmentierung [s]	Anteil Nachbarschaftssuche [s]	Laufzeit Zusammenführung [s]	Laufzeit Gesamt [s]
kNN mit 10 Nachbarn *)	117	18	8	16	34
kNN mit 20 Nachbarn *)	56	30	17	30	60
kNN mit 40 Nachbarn *)	49	57	31	25	82
kNN mit 60 Nachbarn *)	34	84	42	35	119
Kugel mit r = 1 m	79	87	44	10	97
Zylinder mit r = 1 m	48	144	63	14	158
Kugel mit r = 2 m	71	350	142	90	440
Zylinder mit r = 2 m	31	502	200	107	609

Tabelle 5.7: Einfluss der Nachbarschaftssuche auf die Laufzeit der Segmentierung in Wien_aoi_05.laz, bei einer Kachelgröße von 50 m

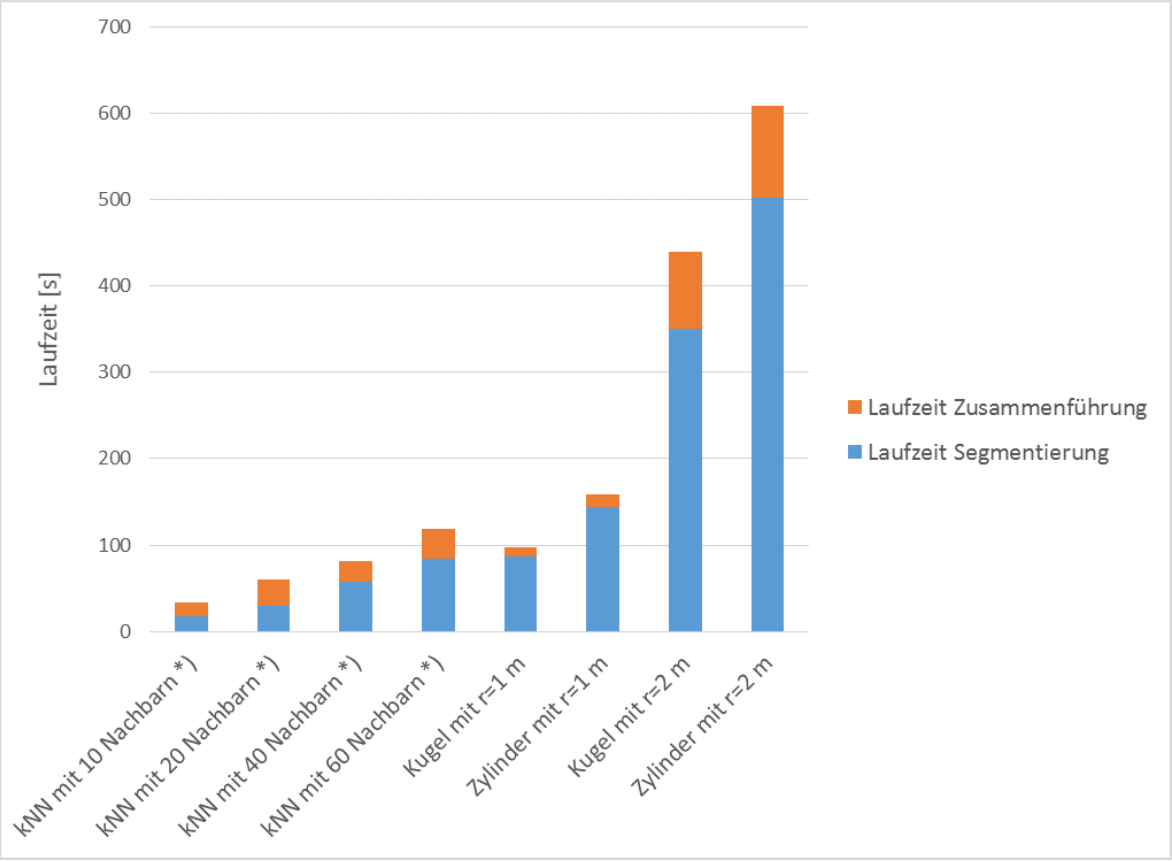

Abbildung 5.11: Vergleich der Laufzeit bei unterschiedlichen Nachbarschaftssuchen, Wien_aoi_05.laz, Kachelgröße 50 m

Die Ergebnisse in Tabelle 5.7 und Abbildung 5.11 zeigen, dass die Laufzeit sehr stark von der gewählten Nachbarschaftssuche abhängt. Dabei kann festgehalten werden, dass die kNN-Suche die schnellste Segmentierung ermöglicht, da die k nächsten Punkte im kd-tree schnell gefunden werden können. (*) Die kNN-Suche ist jedoch keine symmetrische Nachbarschaftssuche. Das heißt, dass eine mehrmalige Segmentierung mit kNN-Suche auf einem Datensatz nicht immer die gleichen Ergebnisse liefern muss. Ein weiterer Nachteil der kNN-Suche ist, dass für die Zusammenführung die Größe der Nachbarschaftsregion abgeschätzt werden muss. Wird diese Region zu klein abgeschätzt, kann es zu Problemen bei der Zusammenführung kommen (z.B. Segmente werden nicht zusammengeführt). Wird diese Region zu groß abgeschätzt, erhöht sich die Laufzeit der Zusammenführung.
Weiters sieht man in den Ergebnissen, dass die Segmentierungsdauer mit der Anzahl der Punkte, die bei der Nachbarschaftssuche zurückgeliefert werden, zunimmt. Um die Geschwindigkeit der Segmentierung zu optimieren, soll also eine möglichst kleine Nachbarschaft gewählt werden. Damit die Segmentierung jedoch sinnvolle Ergebnisse liefern kann, ist eine gewisse Größe für die Nachbarschaft notwendig. Die 2D-Radius-Suche sucht ausgehend von einem Punkt alle Punkte, die in einem Zylinder mit Radius r und Höhe z liegen. Die Höhe des Zylinders entspricht dabei der gesamten Höhe der Punktwolke (z_min - z_max). Es werden also alle Punkte zurückgegeben, die in die XY-Ebene projiziert einen Abstand von maximal r haben. Die 2D-Radius-Nachbarschaftssuche liefert also immer eine Übermenge der 3D-Radius-Suche, welche die Punkte in einer Kugel im Raum zurückliefert. Wenn man den Anteil der Nachbarschaftssuche an der Segmentierungslaufzeit auf den Kacheln betrachtet, sieht man, dass dieser unabhängig von der gewählten Suchmethode etwa 50% der Laufzeit beträgt.

5.2.3 Einfluss - Homogenitätskriterium

Da das Homogenitätskriterium keinen direkten Bezug zu einer laufzeitintensiven Berechnung aufweist, kann der Einfluss auf die Laufzeit der Segmentierung schwer quantifiziert werden. Im Folgenden wird zur Überprüfung des Einflusses die Laufzeit der Segmentierung mit verschiedenen Grenzwerten für das Homogenitätskritierum bestimmt. Wie bereits zuvor wird wieder der Datensatz $Wien\_aoi\_05\_50m.odm$ verwendet. Der Datensatz ist in $50x50m$ Kacheln geteilt. Als Homogenitätskriterium wird wieder

$$|p_i.NormalZ - p_j.NormalZ| \leq \epsilon$$

verwendet. Auch für die Laufzeittests werden für den Grenzwert ϵ die Werte 0.001 m, 0.01 m, 0.1 m und 1.0 m verwendet werden.

Homogenitätskriterium	Segmente Final	Punkte in Segmenten	Segmente vor Merge	Laufzeit Segmentierung [s]	Laufzeit Zusammenführung [s]	Laufzeit Gesamt [s]
Diff. in NormalZ $\leq$ 0.001	57	62.9%	15554	329	128	457
Diff. in NormalZ $\leq$ 0.01	71	85.3%	5748	343	80	423
Diff. in NormalZ $\leq$ 0.1	4	97.5%	1719	340	64	404
Diff. in NormalZ $\leq$ 1.0	1	97.8%	1509	311	64	375

Tabelle 5.8: Einfluss von Homogenitätskriterien auf die Laufzeit der Segmentierung in Wien_aoi_05.laz, bei einer Kachelgröße von 50 m

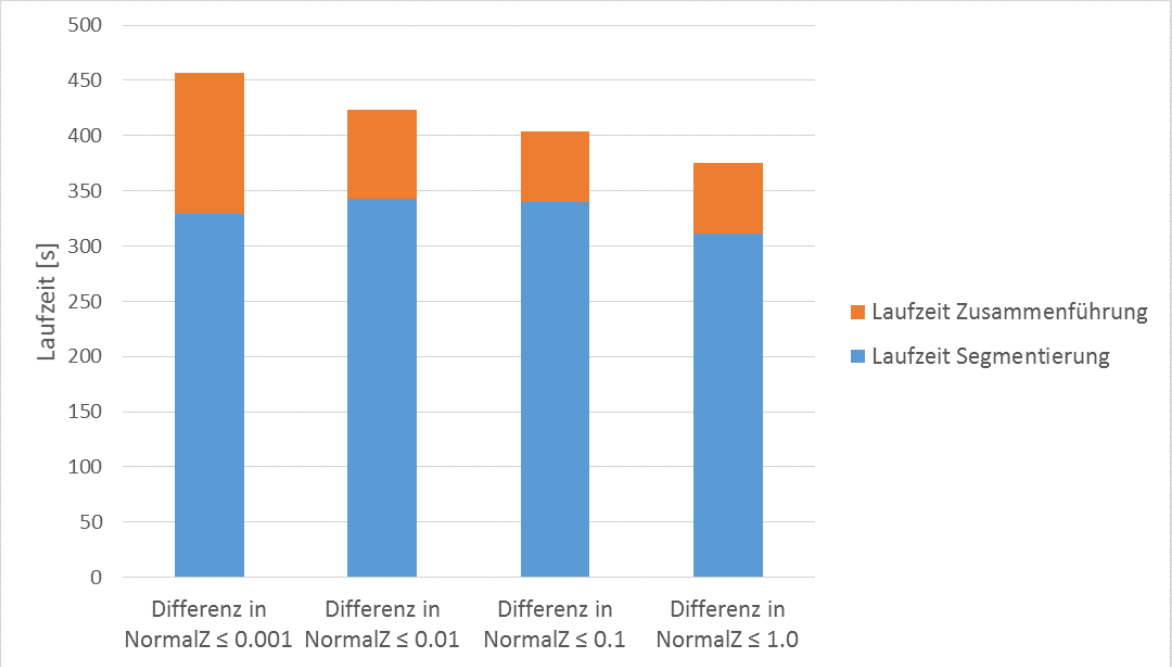

Abbildung 5.12: Vergleich der Laufzeit bei unterschiedlichen Homogenitätskriterien, Wien_aoi_05.laz, Kachelgröße 50 m

Die Laufzeitergebnisse in Abbildung 5.12 und Tabelle 5.8 zeigen, dass die Segmentierungsdauer auf den Kacheln von den Kriterien kaum beeinflusst wird. Bei der Zusammenführung der Segmente sind jedoch trotz gleicher Kacheln Laufzeitunterschiede zu erkennen. Der Grund dafür ist, dass je kleiner die Grenzwerte für die Homogenität sind, umso mehr kleine Segmente (z.T. Einzelpunktsegmente) für die Zusammenführung in Frage kommen. Bei der Zusammenführung müssen also mehr Segmente verglichen und

zusammengeführt werden. Viele dieser kleinen Segmente sind auch nach der Zusammenführung der Kacheln kleiner als die für die Tests definierte minimale Segmentgröße von 50 Punkten. Diese Segmente müssen also im Zusammenführungsschritt verwendet werden, da sie Teil von größeren Segmenten sein könnten, werden aber anschließend in vielen Fällen als zu kleine Segmente verworfen.

Kapitel 6

Zusammenfassung und Ausblick

Wie die Ergebnisse zeigen, liefert das entwickelte Konzept bei unterschiedlichen Kachelgrößen nach Segmentierung und Zusammenführung die gleichen Segmente. Die Segmentierungsergebnisse hängen somit, wie bei einer Segmentierung mit *Region Growing* üblich, nur ab von:

- Homogenitätskriterium $P(p_i, p_j)$
- Nachbarschaftssuche $NH(p_i)$

Zusätzlich hängen die Ergebnisse auch von einer definierten minimalen Segmentgröße ab. Dieser Einfluss wurde im Rahmen der Arbeit jedoch nicht untersucht.

Wie die Laufzeitanalyse zeigt, kann die Berechnungsdauer sowohl bei der Segmentierung, als auch bei der Zusammenführung, bei einer schlechten Wahl der Segmentierungsparameter stark ansteigen. Die Effizienz der Methode ist also von folgenden Parametern abhängig:

- Kachelgröße
- Wahl der Nachbarschaftssuche
- (Wahl des Homogenitätskriteriums)

Eine sinnvolle Wahl der Kachelgröße ist notwendig um die Laufzeit für die Zusammenführung gering zu halten. Während die Segmentierungsdauer nicht beeinflusst wird, kann die Berechnungsdauer der Zusammenführung bei sehr kleinen Kacheln relativ groß werden. Da die Fläche der *Merge*-Nachbarschaft linear mit der Kachelanzahl zunimmt, nimmt auch die Laufzeit für die Zusammenführung mit der Kachelanzahl linear zu.
Eine geeignete Nachbarschaftssuche kann zusätzlich sowohl die Segmentierungszeit, als auch die Dauer der Zusammenführung reduzieren. Der bei der Nachbarschaftssuche

entscheidende Faktor ist die Anzahl der Punkte, welche in der Nachbarschaft liegen. Die Dauer der Berechnungen von Segmentierung und Zusammenführung steigt mit einer größeren Anzahl an Nachbar-Punkten.
Das Homogenitätskriterium hat keinen Einfluss auf die Segmentierungsdauer der Kacheln. Bei einer sinnvollen Wahl des Homogenitätskriteriums ist auch der Einfluss auf die Zusammenführung gering. Wenn durch die Wahl des Homogenitätskriteriums sehr viele kleine Segmente entstehen, hat dies negative Auswirkungen auf die Berechnungsdauer.

Ein Ausblick zeigt Entwicklungsmöglichkeiten in einigen Bereichen. Ein weiterer Schritt für die Optimierung der Laufzeit des Verfahrens wäre dabei, einen sinnvollen Wertebereich für die Parameter automatisch abzuschätzen und den Benutzer gegebenenfalls darauf hinzuweisen.
Zudem bietet das Konzept für eine schnellere Berechnung die Möglichkeiten der Parallelisierung. Sowohl die Segmentierung auf den Kacheln, als auch die Zusammenführung können auf mehrere *Threads* verteilt berechnet werden. Abhängig von der Anzahl der parallelen Prozesse kann die Laufzeit reduziert werden. Im optimalen Fall könnte zum Beispiel eine Berechnung der Segmentierung mit vier parallelen Threads die Laufzeit auf ein Viertel reduzieren. Grundsätzlich ist zu erwarten, dass der Prozess aufgrund des Konzepts sehr gut skaliert, was mit Laufzeittests entsprechend evaluiert werden muss.
Das vorgestellte Konzept baut auf einer regelmäßigen Kachelung auf. Grundsätzlich kann dieses Konzept auch auf Quadtree-Strukturen angepasst werden. Dazu muss die Nachbarschaftsbeziehung in der Zusammenführung überarbeitet werden.
Die aktuelle Implementierung der Segmentierung liefert für jeden Punkt eine Segment-ID beziehungsweise eine einfache Statistik über die Punktanzahl pro Segment. Für aufbauende Prozessierungen können zusätzliche Segment-Statistiken von Interesse sein. Beispielsweise können Umrandungen (minimale Bounding-Box, Alpha-Shape, etc.) oder Attributstatistiken aller Punkte im Segment zu einem besseren Verständnis der Daten beitragen. Diese Repräsentation bringt einen Übergang vom Einzelpunkt zu Objekten. Diese berechneten Objekte können wiederum mit räumlichen Indizes (z. B. Quadtrees, Octtrees, R-Baum) verwaltet werden. Diese Statistiken zu den Objekten können auch dazu beitragen, die Ergebnisse zu interpretieren beziehungsweise zu beurteilen und so gegebenenfalls die Qualität der Segmentierung durch Anpassung der Parameter zu verbessern.

Abbildungsverzeichnis

Tabellenverzeichnis

Literaturverzeichnis

[Achanta u. a. 2012] Achanta, Radhakrishna ; Shaji, Appu ; Smith, Kevin ; Lucchi, Aurelien ; Fua, Pascal ; Susstrunk, Sabine: SLIC superpixels compared to state-of-the-art superpixel methods. In: *Pattern Analysis and Machine Intelligence, IEEE Transactions on* 34 (2012), Nr. 11, S. 2274–2282

[Adams und Bischof 1994] Adams, Rolf ; Bischof, Leanne: Seeded region growing. In: *Pattern Analysis and Machine Intelligence, IEEE Transactions on* 16 (1994), Nr. 6, S. 641–647

[Bentley 1975] Bentley, Jon L.: Multidimensional Binary Search Trees Used for Associative Searching. In: *Commun. ACM* 18 (1975), September, Nr. 9, S. 509–517. – URL `http://doi.acm.org/10.1145/361002.361007`. – ISSN 0001-0782

[Brown und Lowe 2005] Brown, Matthew ; Lowe, David G.: Unsupervised 3D object recognition and reconstruction in unordered datasets. In: *3-D Digital Imaging and Modeling, 2005. 3DIM 2005. Fifth International Conference on* IEEE (Veranst.), 2005, S. 56–63

[Chen und Pavlidis 1990] Chen, Ming-Hua ; Pavlidis, Theo: Image seaming for segmentation on parallel architecture. In: *Pattern Analysis and Machine Intelligence, IEEE Transactions on* 12 (1990), Nr. 6, S. 588–594

[Felzenszwalb und Huttenlocher 2004] Felzenszwalb, Pedro F. ; Huttenlocher, Daniel P.: Efficient graph-based image segmentation. In: *International Journal of Computer Vision* 59 (2004), Nr. 2, S. 167–181

[Fernandez-Diaz u. a. 2014] Fernandez-Diaz, Juan C. ; Carter, William E. ; Shrestha, Ramesh L. ; Glennie, Craig L.: Now you see it... now you don't: Understanding airborne mapping LiDAR collection and data product generation for archaeological research in Mesoamerica. In: *Remote Sensing* 6 (2014), Nr. 10, S. 9951–10001

[GISGeography 2016] GISGeography: *Number of LIDAR Returns*. 05 2016. – URL http://i0.wp.com/gisgeography.com/wp-content/uploads/2015/08/Number-Returns.png

[Hirschmüller 2008] Hirschmüller, Heiko: Stereo processing by semiglobal matching and mutual information. In: *Pattern Analysis and Machine Intelligence, IEEE Transactions on* 30 (2008), Nr. 2, S. 328–341

[Kraus 2004] Kraus, Karl: *Photogrammetrie: Geometrische Informationen aus Photographien und Laserscanneraufnahmen*. Walter de Gruyter, 2004

[MacQueen 1967] MacQueen, James: Some methods for classification and analysis of multivariate observations. In: *Proceedings of the fifth Berkeley symposium on mathematical statistics and probability* Bd. 1 Oakland, CA, USA. (Veranst.), 1967, S. 281–297

[Otepka u. a. 2012] Otepka, J ; Mandlburger, G ; Karel, W: The OPALS Data Manager—Efficient Data Management for Processing Large Airborne Laser Scanning Projects. In: *Proceedings of the ISPRS Annals of the Photogrammetry, Melbourne, Australia* 25 (2012), S. 153–159

[Otepka u. a. 2013] Otepka, Johannes ; Ghuffar, Sajid ; Waldhauser, Christoph ; Hochreiter, Ronald ; Pfeifer, Norbert: Georeferenced point clouds: A survey of features and point cloud management. In: *ISPRS International Journal of Geo-Information* 2 (2013), Nr. 4, S. 1038–1065

[Rutzinger u. a. 2008] Rutzinger, Martin ; Höfle, Bernhard ; Hollaus, Markus ; Pfeifer, Norbert: Object-based point cloud analysis of full-waveform airborne laser scanning data for urban vegetation classification. In: *Sensors* 8 (2008), Nr. 8, S. 4505–4528

[Shi und Malik 2000] Shi, Jianbo ; Malik, Jitendra: Normalized cuts and image segmentation. In: *Pattern Analysis and Machine Intelligence, IEEE Transactions on* 22 (2000), Nr. 8, S. 888–905

[TerraImaging 2016] TerraImaging: *Principle of ALS*. 05 2016. – URL http://www.terraimaging.de/upload/Bilder/Technology/9_tech_laser_1_pop.jpg

[Teutsch u. a. 2011] Teutsch, Christian ; Trostmann, Erik ; Berndt, Dirk: A parallel point cloud clustering algorithm for subset segmentation and outlier detection. In: *SPIE Optical Metrology* International Society for Optics and Photonics (Veranst.), 2011, S. 808509–808509

[Vosselman u. a. 2004] Vosselman, George ; Gorte, Ben G. ; Sithole, George ; Rabbani, Tahir: Recognising structure in laser scanner point clouds. In: *International archives of photogrammetry, remote sensing and spatial information sciences* 46 (2004), Nr. 8, S. 33–38

[Waldhauser u. a. 2014] Waldhauser, Christoph ; Hochreiter, Ronald ; Otepka, Johannes ; Pfeifer, Norbert ; Ghuffar, Sajid ; Korzeniowska, Karolina ; Wagner, Gerald: Automated classification of airborne laser scanning point clouds. In: *Solving Computationally Expensive Engineering Problems*. Springer, 2014, S. 269–292