Veröffentlichungen der DGK

Ausschuss Geodäsie der Bayerischen Akademie der Wissenschaften

Reihe C | Dissertationen | Heft Nr. 808

Daniel Eggert

Effiziente Verarbeitung und Visualisierung von Mobile Mapping Daten

München 2017

Verlag der Bayerischen Akademie der Wissenschaften

ISSN 0065-5325 ISBN 978-3-7696-5220-8

Diese Arbeit ist gleichzeitig veröffentlicht in:
Wissenschaftliche Arbeiten der Fachrichtung Geodäsie und Geoinformatik der Universität Hannover
ISSN 0174-1454, Nr. 337, Hannover 2017

Veröffentlichungen der DGK

Ausschuss Geodäsie der Bayerischen Akademie der Wissenschaften

Reihe C | Dissertationen | Heft Nr. 808

Effiziente Verarbeitung und Visualisierung von Mobile Mapping Daten

Von der Fakultät für Bauingenieurwesen und Geodäsie
der Gottfried Wilhelm Leibniz Universität Hannover
zur Erlangung des Grades
Doktor-Ingenieur (Dr.-Ing.)
genehmigte Dissertation

Vorgelegt von

M.Sc. Daniel Eggert

Geboren am 10.07.1982 in Stendal

München 2017

Verlag der Bayerischen Akademie der Wissenschaften

ISSN 0065-5325 ISBN 978-3-7696-5220-8

Diese Arbeit ist gleichzeitig veröffentlicht in:
Wissenschaftliche Arbeiten der Fachrichtung Geodäsie und Geoinformatik der Universität Hannover
ISSN 0174-1454, Nr. 337, Hannover 2017

Adresse der DGK:

Ausschuss Geodäsie der Bayerischen Akademie der Wissenschaften (DGK)

Alfons-Goppel-Straße 11 ● D – 80 539 München

Telefon +49 – 331 – 288 1685 ● Telefax +49 – 331 – 288 1759

E-Mail post@dgk.badw.de ● http://www.dgk.badw.de

Prüfungskommission:

Vorsitzender: Prof. Dr.-Ing. habil. Jürgen Müller

Referent: Prof. Dr.-Ing. habil. Monika Sester

Korreferenten: Prof. Dr.-Ing. Christian Heipke
Prof. Dr. Jürgen Döllner (HPI Potsdam)

Tag der mündlichen Prüfung: 23.06.2017

© 2017 Bayerische Akademie der Wissenschaften, München

Alle Rechte vorbehalten. Ohne Genehmigung der Herausgeber ist es auch nicht gestattet,
die Veröffentlichung oder Teile daraus auf photomechanischem Wege (Photokopie, Mikrokopie) zu vervielfältigen

ISSN 0065-5325 ISBN 978-3-7696-5220-8

WISSENSCHAFTLICHE ARBEITEN DER FACHRICHTUNG
GEODÄSIE UND GEOINFORMATIK DER LEIBNIZ UNIVERSITÄT HANNOVER
ISSN 0174-1454

Nr. 337

Daniel Eggert

Effiziente Verarbeitung und Visualisierung von Mobile Mapping Daten

HANNOVER 2017

WISSENSCHAFTLICHE ARBEITEN DER FACHRICHTUNG
GEODÄSIE UND GEOINFORMATIK DER LEIBNIZ UNIVERSITÄT
HANNOVER
ISSN 0174-1454

Nr. 337

Effiziente Verarbeitung und Visualisierung von Mobile Mapping Daten

Von der Fakultät für Bauingenieurwesen und Geodäsie
der Gottfried Wilhelm Leibniz Universität Hannover
zur Erlangung des Grades

Doktor-Ingenieur (Dr.-Ing.)

genehmigte Dissertation von
M.Sc. Daniel Eggert
aus Hannover

HANNOVER 2017

Diese Arbeit ist auch veröffentlicht in:
DEUTSCHE GEODÄTISCHE KOMMISSION bei der Bayerischen Akademie der Wissenschaften

Reihe C, Dissertationen, Heft Nr. xxx, München 2017
ISBN 978 3 7696 xxxx x, ISSN xxxx-xxxx
www.dgk.badw.de

Vorsitzender der Prüfungskommission:	Prof. Dr.-Ing. habil. Jürgen Müller
Referentin:	Prof. Dr.-Ing. habil. Monika Sester
Korreferenten:	Prof. Dr.-Ing. Christian Heipke
	Prof. Dr. Jürgen Döllner
Tag der Promotion:	23.06.2017

Zusammenfassung

Eine in den letzten Jahren zunehmend an Bedeutung gewonnene Methode zur Erfassung von Geodaten stellt das Mobile Mapping dar. Mobile Mapping bezeichnet dabei die Erfassung von Geodaten mittels einer beweglichen Plattform. Im Gegensatz zu statischen Erfassungssystemen ermöglichen bewegliche Plattformen die effiziente Erfassung größerer Bereiche. Dies erfordert jedoch in der Regel eine aufwendige Registrierung und Integration der Daten (hier Kamerabilder und Scanpunkte bzw. -linien) und führt darüber hinaus zu umfangreichen Datenbeständen. Mit diesem Umfang gehen entsprechende Herausforderungen für die Speicherung, Organisation, Verarbeitung und Visualisierung der Daten einher. Neben diesen Herausforderungen spielt die Integration der verwendeten Sensoren eine entscheidende Rolle für die Qualität der Daten. Ist die Zuordnung der Daten der einzelnen Sensoren zueinander von schlechter Genauigkeit oder gar fehlerhaft, so spiegelt sich dies auch in den Analyse- und Verarbeitungsresultaten wieder. Dies birgt die Gefahr von qualitativ schlechten Resultaten und möglichen Fehlinterpretationen von Analyseergebnissen, oder macht die Analyse als solches gar unmöglich.

Die vorliegende Arbeit adressiert die genannten Aspekte im Kontext von Mobile Mapping mittels folgender Zielsetzungen: a) Schaffung einer modularen Verarbeitungskette, b) Effizienzbetrachtung, inklusive der Entwicklung und Umsetzung von hoch-parallelen Prozessierungsmodulen und Datentrukturen für effizienten Zugriff auf die Daten, c) Erhöhung der Qualität der Sensordatenintegration am Beispiel der Kamera- und Laserscanner-Sensoren und d) die Entwicklung skalierbarer Visualisierungskonzepte.

Nach der Identifikation möglicher Anknüpfungspunkte bezüglich der existierenden proprietären Herstellerlösungen wurde im Rahmen dieser Arbeit ein modulares Framework zur Verarbeitung der besagten Mobile Mapping Daten geschaffen. Mittels dieses Frameworks lassen sich nahezu beliebige Verarbeitungsketten abbilden. Im Rahmen einer Effizienzbetrachtung einzelner Verarbeitungsschritte wurden Konzepte zur Parallelisierung der Verarbeitung untersucht und in den Varianten Mehrkern-CPU, GPGPU und Rechencluster/Hadoop umgesetzt. Eine Qualitätssteigerung der Sensordatenintegration wurde über die Entwicklung eines neuen Verfahrens zur Feinkalibrierung der Kamerapositionen und -orientierungen erreicht. Das realisierte Verfahren beruht dabei auf dem automatischen Finden von korrespondierenden Scan- und Bildpunkten und einem anschließenden Rückwärtsschnitt zur Ermittlung der genauen Kameraposition und -orientierung. Anhand mehreren Beispieldatensätzen konnte dabei eine signifikante Reduktion der Residuen, als Pixelabstand zwischen Soll- und Ist-Koordinaten, gezeigt werden. Im Rahmen der Entwicklung skalierbarer Visualisierungskonzepte für die erfassten Mobile Mapping Daten wurden, ausgehend von unterschiedlichen Szenarien, zwei Visualisierungsansätze umgesetzt. Zunächst wurde eine Webbasierte 3D Visualisierung umgesetzt, welche die Darstellung nahezu beliebig umfangreicher Mobile Mapping Daten in hybrider Repräsentation, bestehend aus eingefärbten Scanpunkten und texturierten Flächen, in einem gewöhnlichen Internetbrowser erlaubt. Kern der Skalierbarkeit dieser Visualisierungtechnik bildet eine, für die gewählte Umgebung entworfene, kontinuierliche Level-Of-Detail Technik. Der zweite Visualisierungsansatz beruht auf der Parallax Scrolling Technik. Diese stark komprimierte Form der Darstellung in Kombination mit einer intuitiven Gestenbasierten Steuerung ermöglicht das schnelle Erkunden von umfangreichen Mobile Mapping Daten.

Schlagworte: Mobile Mapping, Lidar, Visualisierung, Framework, Sensorintegration

Abstract

Mobile mapping represents an increasingly popular technique to acquire geodata. In the context of mobile mapping a mobile platform is utilized in order to obtain information of the surrounding area. In contrast to common static data acquisition systems, mobile platforms are able to cover bigger areas more effectively. This, however, demands a comprehensive data registration and integration process and results in voluminous data repositories. This of course yields various challenges regarding the storage, organization, processing and visualization of the recorded data. Furthermore the quality of the data integration is crucial for the quality of derived data products. Bad accuracies and precisions may result in bad, misleading or even futile processing and analysis results.

In order to address the shown challenges in the context of mobile mapping this thesis pursues the following goals: a) the design of a modular framework to allow the definition of arbitrary scientific processing and analysis workflows, b) analyze the efficiency of applied processing modules and the design of highly parallel module implementations, c) improve the quality of the required sensor-fusion of the involved laserscanner and camera sensors, as well as d) the development of scalable concepts for the visualization of comprehensive mobile mapping data repositories.

Based on the identification and analysis of possible interfaces to the existing proprietary solutions provided by the hardware manufacturers a modular processing framework for mobile mapping data was designed. This framework enables users to assemble arbitrary processing and analysis workflows to answer a wide range of practical and scientific questions. In order to allow the fast processing of big mobile mapping data repositories this thesis investigates various concepts for parallel processing and provides and compares module implementations utilizing multi-core CPU, GPGPU and computer cluster. Concerning the integration of multiple sensors a new approach to accurately calibrate the position and orientation of the involved cameras, in respect to the laserscanners, was developed. Based on the automatic identification of corresponding scan- and image-points an iterative resection process is applied to determine the position and orientation of each camera. Multiple experiments prove that the presented approach reduces the residuals significantly. Finally, two scalable visualization approaches are presented, addressing different scenarios. First, a web-based 3D visualization technique was designed and implemented. This technique allows the visualization of extensive mobile mapping data repositories in a common internet browser. Based on the generation of hybrid data models, consisting of textured planes and bare scan-points, a novel continuous Level-Of-Detail technique is presented. The second scalable visualization approach utilizes the parallax scrolling technique. This highly compressed data representation in combination with an intuitive gesture-based navigation allows the fast browsing and exploration of huge mobile mapping data repositories.

Keywords: Mobile Mapping, Lidar, Visualization, Framework, Sensor-fusion

Inhaltsverzeichnis

1 Einleitung **15**

1.1 Zielsetzung . . . 15
1.2 Struktur . . . 17

2 Grundlagen und Stand der Forschung **19**

2.1 Mobile Mapping Systeme . . . 19
2.1.1 Allgemeine Funktionsweise . . . 19
2.1.2 Riegl VMX-250 . . . 20
2.1.3 Alternative Systeme . . . 21
2.2 Punktwolken . . . 22
2.2.1 Definition . . . 22
2.2.2 Abgrenzung zu vermaschten Punkten . . . 22
2.2.3 Speicherformate von Punktwolken . . . 23
2.2.4 Visualisierungstechniken . . . 25
2.2.5 Level Of Detail . . . 29
2.3 Farbmodelle . . . 30
2.3.1 Farbanpassung . . . 30
2.4 Verteiltes Rechnen . . . 31
2.5 Verdeckungsanalyse . . . 32
2.6 Registrierung mehrerer Datensätze . . . 33
2.7 Visualisierungssysteme . . . 34
2.7.1 Standalone Point Cloud Viewer . . . 34
2.7.2 Webbasierte Systeme . . . 34

3 Effizienzbetrachtungen **37**

3.1 Effiziente Verarbeitung von Massendaten durch Parallelisierung . . . 37
3.1.1 Parallelisierungsformen . . . 38
3.1.2 Umsetzung . . . 38
3.1.3 Vergleich . . . 42
3.2 Effiziente Datenstrukturen . . . 43
3.2.1 Scanstreifen . . . 44
3.2.2 Scanstreifenbasierte Pufferstrategie . . . 44
3.2.3 Rasterdatenstruktur . . . 48
3.2.4 Randproblematik und Caching . . . 50

4 Modulare Verarbeitungskette für Mobile Mapping Daten **53**

4.1 Analyse der beteiligten Komponenten des Herstellerworkflows . . . 53
4.2 Exemplarische modulare Verarbeitungskette . . . 54

4.3 Vorverarbeitungsmodul . . . 55
4.3.1 Vereinfachung . . . 55
4.3.2 Zeitsegmentierung . . . 57
4.3.3 Bestimmung von Punktattributen . . . 58
4.4 Segmentierung und Klassifikation . . . 59
4.4.1 Bodenextraktion . . . 60
4.4.2 Objektsegmentierung . . . 61

5 Sensordatenintegration: Kalibrierung der Kameraorientierung 63
5.1 Zeitstempelabweichung . . . 63
5.2 Ansatz . . . 64
5.3 Extraktion von Silhouetten . . . 65
5.3.1 Extraktion von Silhouetten aus Kamerabildern . . . 65
5.3.2 Extraktion von Silhouetten aus Laserscandaten . . . 68
5.4 ICP-basierte Identifikation der Korrespondenzen . . . 70
5.4.1 Beschränkung der Scanpunktbildsilhouette . . . 70
5.4.2 Gruppierung der Scanpunktdaten . . . 71
5.4.3 ICP unter Berücksichtigung der Punktnormalen . . . 71
5.5 Bestimmung der Kameraparameter mittels Rückwärtsschnitt . . . 72
5.5.1 Wahl der Stichprobe . . . 74
5.5.2 Anzahl an Iterationen . . . 75
5.5.3 Bewertung der gefundenen Modelle . . . 75
5.6 Ergebnisse . . . 76
5.7 Verbesserungspotential und Probleme . . . 77
5.7.1 Laufzeiten . . . 79
5.7.2 Robustheit des Verfahrens und Qualität der Ergebnisse . . . 79

6 Farbbestimmung 81
6.1 Farbextraktion . . . 82
6.2 Verdeckungsanalyse . . . 82
6.2.1 Geometrische Verdeckungsanalyse . . . 84
6.2.2 Ballbasierter Tiefenpuffer . . . 85
6.2.3 Ergebnisse . . . 87
6.2.4 Nicht erfasste und dynamische Objekte . . . 88
6.3 Farbanpassung . . . 89
6.3.1 Einfärbesituationen benachbarter Scanpunkte . . . 89
6.3.2 Objektweise Farbanpassung . . . 91
6.3.3 IDP-Interpolierte radiometrische Helligkeitsanpassung von Bodenpunkten . . . 91
6.3.4 Radiometrische Helligkeits- und Sättigungsanpassung von Objektpunkten . . . 94
6.4 Farbsynthese . . . 96
6.4.1 Histogrammbasierte Farbinterpolation . . . 96
6.4.2 Ergebnis . . . 97

7 Aus Punktwolken abgeleitete Modelle 99
7.1 3D Modelle . . . 99
7.1.1 Identifikation planarer Bereiche . . . 100
7.1.2 Nachbearbeitung der erstellten Texturen . . . 103
7.1.3 Effiziente Verwaltung von Texturen . . . 106
7.1.4 Erhöhung der Speichereffizienz . . . 108
7.1.5 Level of Detail . . . 110
7.2 2D Modelle . . . 114
7.2.1 Trackjektorienabschnitte . . . 115
7.2.2 Ermittlung relevanter Ebenen . . . 119
7.2.3 Ergebnis . . . 123

8 Visualisierung von Mobile Mapping Daten 125
8.1 3D Visualisierung . . . 125
8.1.1 Visualisierung via Web-App . . . 126
8.1.2 Performante Client-Server Kommunikation und Serialisierung . . . 128
8.1.3 Scheduling der LOD-Daten . . . 130
8.1.4 GUI Responsiveness . . . 132
8.1.5 Navigation und Nutzerinteraktion . . . 133
8.2 2D Visualisierung . . . 135
8.2.1 Parallax Scrolling Visualisierung via Android-App . . . 137
8.2.2 Beleuchtungsmodell . . . 139
8.2.3 Ergebnis und Ausblick . . . 139

9 Schlussfolgerungen und Ausblick 143
9.1 Ausblick . . . 145

Literaturverzeichnis 148

1 Einleitung

Die Erfassung von Geodaten weist eine lange und vielschichtige Historie auf. Die erfassten Daten verhelfen Wissenschaftlern sowohl natürliche, als auch anthropogene Phänomene zu beobachten und zu analysieren. So vielfältig wie die zu erfassenden Daten, sind auch die eingesetzten Erfassungsmethoden. Eine in den letzten Jahren zunehmend an Bedeutung gewonnene Methode stellt das Mobile Mapping dar. Mobile Mapping bezeichnet dabei die Erfassung von Geodaten mittels einer beweglichen Plattform. Grundsätzlich kann dabei als mobile Plattform alles dienen worauf sich die benötigte Sensorik befestigen lässt. Beispiele dafür reichen vom gewöhnlichen Fahrrad über Zug, Hubschrauber und Boot bis hin zur ferngesteuerten Drohnen. Die eingesetzten Sensoren sind ebenfalls vielfältig und umfassen beispielsweise Laserscanner, Radar und Kameras. Darüber hinaus ist eine Kombination mehrerer Sensoren typisch. Im Gegensatz zu statischen Erfassungssystemen ermöglichen bewegliche Plattformen die effiziente Erfassung größerer Bereiche. Dies gelingt allerdings nur auf Kosten einer aufwendigeren Registrierung und Integration der Daten. Hierfür muss in der Regel die exakte Position und Orientierung der Plattform zu jeder Zeit bekannt sein.

Die schnelle und großflächige Datenerhebung ermöglicht eine Vielzahl an Anwendungsszenarien und aus den Daten abgeleiteten Produkten. In Abhängigkeit der verwendeten Sensoren können beispielsweise 3D Stadtpläne generiert und vorhandene Katasterpläne aktualisiert werden. Ist eine leichte Wiederholbarkeit der Datenerfassung gegeben, sind neben Bestandsaufnahmen auch Monitoring und Änderungsüberwachung als mögliche Anwendungsfälle denkbar. Aktuelle Forschungsarbeiten befassen sich darüber hinaus mit der Frage wie beispielsweise Merkmalskarten aus den erfassten Daten generiert werden können. Solche Merkmalskarten z.B. stellen eine wichtige Voraussetzung für eine Landmarken-basierte Navigation dar. Die gezeigten Eigenschaften einer mobilen Datenerfassung: schnell, großflächig, genau und wiederholbar, führen jedoch zu einem großen Datenumfang. Mit diesem Umfang gehen entsprechende Herausforderungen für die Speicherung, Organisation, Verarbeitung und Visualisierung der Daten einher. Neben diesen Herausforderungen spielt die Integration der verwendeten Sensoren eine entscheidende Rolle für die Qualität der Daten. Ist die Zuordnung der Daten der einzelnen Sensoren zueinander von schlechter Genauigkeit oder gar fehlerhaft, so spiegelt sich dies auch in den Analyse- und Verarbeitungsresultaten wieder. Dies birgt die Gefahr von qualitativ schlechten Resultaten und möglichen Fehlinterpretationen von Analyseergebnissen, oder macht die Analyse als solches gar unmöglich.

Die vorliegende Arbeit adressiert die genannten Aspekte im Kontext von Mobile Mapping. Als Erfassungssystem wurde eine Mobile Mapping Plattform der Firma Riegl in Kombination mit einem gewöhnlichen Kraftfahrzeug genutzt. Die wesentlichen Komponenten des Systems stellen dabei zwei Laserscanner, ein Kamerasystem, sowie GNSS/IMU Sensorik dar. Obgleich sich die im Rahmen dieser Arbeit entwickelten Verfahren und Lösungsansätze auf diese konkrete Konstellation beziehen, lassen sie sich im Kern auch auf andere Mobile Mapping Systeme und Plattformen übertragen.

1.1 Zielsetzung

Die Zielsetzung dieser Arbeit umfasst vier Aspekte:

1. Schaffung einer modularen Verarbeitungskette,
2. Effizienzbetrachtung, inklusive der Entwicklung und Umsetzung von hoch-parallelen Prozessierungsmodulen und Datentrukturen für effizienten Zugriff auf die Daten,
3. Erhöhung der Qualität der Sensordatenintegration am Beispiel der Kamera- und Laserscanner-Sensoren,
4. Entwicklung skalierbarer Visualisierungskonzepte

Für eine ganzheitliche Betrachtung wurde im Rahmen dieser Arbeit die Verarbeitungskette des verwendeten Mobile Mapping Systems analysiert, welche im Wesentlichen über proprietäre Lösungen des Herstellers abgebildet wird. Ausgehend von der eruierten Verarbeitungskette wurden Anknüpfungspunkte für eigene Lösungen identifiziert. Das Ziel liegt darin, eine offene, modulare Verarbeitungskette mit ebenfalls offenen standardisierten Datenaustauschformaten zu schaffen. Im Gegensatz zu den Blackbox-artigen Herstellerlösungen ermöglicht solch ein Konstrukt einen Umgang mit Daten und angewendeten Verfahren, welcher näher an wissenschaftlichen Fragestellungen liegt.

Über das Modularisierungskonzept hinaus wurden einzelne Teilaspekte der identifizierten Verarbeitungskette mittels eigener Module umgesetzt. Anhand der umgesetzten Module sollen Verfahren prototypisch aufgezeigt werden, welche zum einen den umfänglichen Charakter der Daten adressieren und zum anderen, zu qualitativ besseren Resultaten führen. Bei allen umgesetzten Verarbeitungsmodulen stand ein effizienter Umgang mit den Daten im Vordergrund. Dafür wurden mehrere Konzepte zur Parallelisierung der Prozessierung untersucht. Die Untersuchung umfasst dabei die Datenverarbeitung mittels Mehrkern-CPU, General Purpose Computation on Graphics Processing Unit (GPGPU), sowie Rechencluster. Darüber hinaus wurden effiziente Datenstrukturen und Zugriffsstrategien für konkrete Anwendungsfälle entwickelt. Die Kombination einer skalierbaren, hoch-parallelen Verarbeitung und einer effizienten Datenorganisation ermöglicht die Prozessierung selbst umfangreicher Datensätze und erlaubt so die Adressierung neuer wissenschaftlicher Fragestellungen.

Wie eingangs erläutert spielt die Integration der Sensordaten eine elementare Rolle für die Qualität der Resultate. Ein Beispiel dafür ist die Kombination, bzw. Integration der Daten der Kamera- und Laserscannersensoren. Ziel dieser Integration ist die Ermittlung der Farbe jedes Scanpunktes. Diese kann in anschließenden Analyseverfahren als zusätzliches Attribut einfließen und erlaubt darüber hinaus umfassendere Visualisierungsoptionen. Die Analyse der proprietären Herstellerlösungen hat ein hohes Verbesserungspotential bei der genannten Sensorintegration offenbart. Je exakter die Farbbestimmung der Scanpunkte, desto qualitativ hochwertiger ist eine daraus abgeleitete Visualisierung. Aus diesem Grund wurden im Rahmen dieser Arbeit neue Verfahren zur Feinkalibrierung der Kamera, sowie zur Farbbestimmung unter Berücksichtigung etwaiger Verdeckungen entwickelt.

So relevant wie die Verarbeitung der Mobile Mapping Daten ist auch ihre Visualisierung, sei es in Form der umfangreichen Rohdaten, die Ergebnisse einzelner Analyseverfahren oder speziell für eine Visualisierung erstellter Modelle. Dafür wurden zwei Visualisierungkonzepte entworfen. Zum einen wurde ein Web-basiertes Explorationsszenario umgesetzt, welches eine Erkundung der eingefärbten Punktwolke über einen Webbrowser ermöglicht. Dabei wurden Konzepte zur effizienten Datenübertragung und entsprechende Level Of Detail Techniken entwickelt und implementiert. Zum anderen wurde ein Visualisierungsansatz zur schnellen Betrachtung der Daten entworfen. Dieser basiert auf der Darstellung eines stark komprimierten Modells mittels der Parallax Scrolling Technik.

Abbildung 1.1 gibt einen Überblick über die Verarbeitungsschritte ausgehend von der Datenerfassung bis hin zur Darstellung. Zunächst wird aus den erfassten Daten mittels diverser Verarbeitungsmodule eine eingefärbte Punktwolke generiert. Diese Punktwolke bildet die Grundlage für die anschließende Modellgenerierung, bei der, passend zum beabsichtigten Visualisierungskonzept, die entsprechenden 2D und 3D Modelle erstellt werden. Zur abschließenden Darstellung der jeweiligen Modelle kommen die eigens dafür umgesetzten Visualisierungsprototypen zum Einsatz.

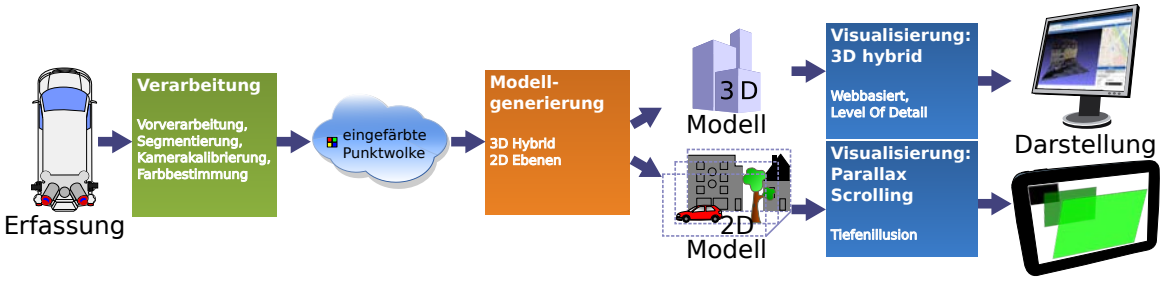


Abbildung 1.1: Prototypischer Workflow ausgehend von den erfassten Daten.

1.2 Struktur

An diese Einleitung schließt sich ein Grundlagenkapitel (Kapitel 2) an, welches essentielle Begriffe einführt und einen Überblick über den aktuellen Stand der Forschung gibt. Der restliche Aufbau (Kapitel 3 bis 8) der vorliegenden Arbeit orientiert sich weitestgehend an den umgesetzten Modulen mit steigendem Abstraktionsgrad. Abbildung 1.2 gibt einen Überblick über das gesamte entworfene Framework, dessen Module, sowie exemplarische Komponenten. Die Module sind in vier aufeinander aufbauenden Gruppen bzw. Schichten unterteilt.

Kernmodule (grün) bilden die unterste Schicht und damit das Fundament für alle Module der höheren Schichten. Zu ihnen gehören grundlegende Komponenten wie die Unterstützung diverser Speicherformate für Punktwolken und Bilder, sowie Datenstrukturen wie Scanstreifen oder Punktwolkenkacheln und Prozessierungsmodelle, wie MapReduce, mittels derer übergeordnete Verfahren ausgeführt werden. Die wesentlichen Komponenten des Datenstruktur-, sowie des Prozessierungsmodellmoduls werden in Kapitel 3 beschrieben. Auf die unterstützten Speicherformate wird in den jeweils relevanten Abschnitten eingegangen.

Basismodule (gelb) umfassen Komponenten, welche grundlegende Verfahren und Methoden bereitstellen. Diese sind in drei Module gegliedert. Wobei das Vorverarbeitungsmodul Komponenten umfasst, welche häufig vor der eigentlichen Verarbeitung ausgeführt werden, wie bspw. die Ermittlung bestimmter Punktattribute oder eine Vereinfachung der Punktwolke. Darüber hinaus wurde ein Modul zur Segmentierung und Klassifikation umgesetzt, deren Komponenten in Verfahren zum Einsatz kommen, welche Segmentierte oder klassifizierte Daten voraussetzen. Kapitel 4 beschreibt beide Basismodule im Detail. Das Modul Kamerakalibrierung adressiert die dritte Zielsetzung (Verbesserung der Sensordatenintegration) und ist in Kapitel 5 beschrieben.

Verarbeitungsmodule (orange) setzen viele Komponenten der Basismodule voraus. Die Komplexität von Verarbeitungsmodulen übersteigt damit die von Basismodulen, weshalb erstere in einer eigenen Schicht modelliert sind. Das Verarbeitungsmodul zur Farbbestimmung der einzelnen Scanpunkte ist in Kapitel 6 beschrieben. Diesem schließt sich die Beschreibung der abgeleiteten Punktwolkenmodelle in Kapitel 7 an.

Visualisierungsmodule (violett) bilden die oberste Schicht und damit die mit dem höchsten Abstraktionsgrad. Sie umfassen Komponenten zur Visualisierung der zuvor generierten Punktwolkenmodelle. Kapitel 8 zeigt die dafür entworfenen Konzepte und die anhand derer umgesetzten Prototypen.

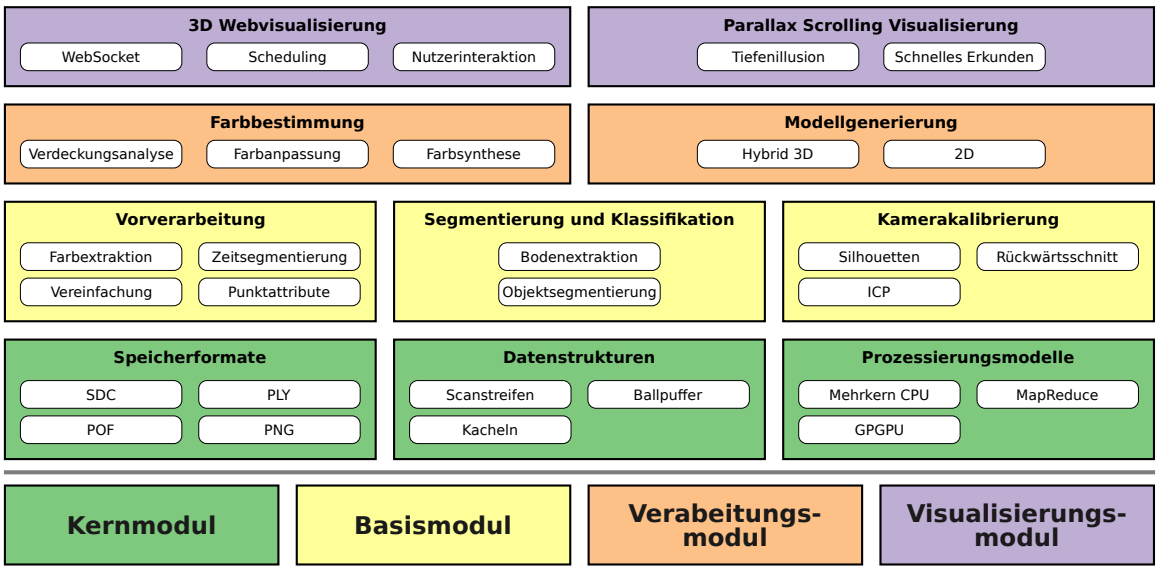


Abbildung 1.2: Übersicht über Module des konzipierten Frameworks.

Zur leichteren Einordnung in den Gesamtkontext wird die Modulübersicht aus Abbildung 1.2 in den folgenden Kapiteln und Abschnitten erneut aufgegriffen. Wie das Beispiel aus Abbildung 1.3 verdeutlicht, werden Komponenten welche im jeweiligen Abschnitt beschrieben werden mittels eines roten Rahmens hervorgehoben. Komponenten ohne roten Rahmen werden danach nicht im jeweilgen Abschnitt erläutert. Alle farbigen Module sind für die beschriebenen Komponenten relevant und damit eine notwendige Voraussetzung. Ausgegraute Module spielen hingegen keine Rolle im jeweils aktuellen Kontext.

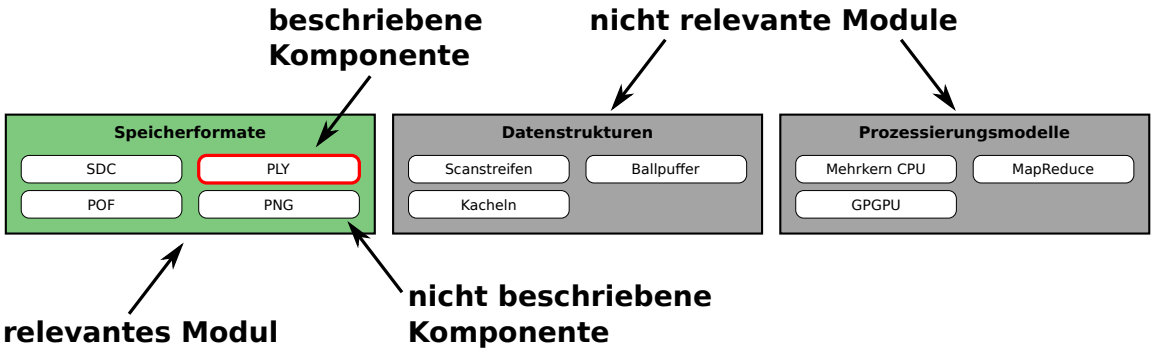


Abbildung 1.3: Erläuterung der Elemente und Bedeutung der Modulübersichten.

2 Grundlagen und Stand der Forschung

2.1 Mobile Mapping Systeme

Als Mobile Mapping wird im allgemeinen die Erfassung räumlicher Daten mittels einer beweglichen Sensorplattform bezeichnet (vgl. Tao und Li (2007)). Der Umfang an möglichen Trägerplattformen reicht dabei von Kraftfahrzeugen über Zügen, Helikoptern und Flugzeugen bis hin zu Booten und unbemannten Drohnen (UAVs). Die Bandbreite der möglichen Sensoren zur Datenerfassung ist dabei ähnlich Umfangreich. Sie reicht von Kamerabasierten Systemen (Hyperspektral, Thermal, ...), über Radar bis hin zu LiDAR (Light Detection And Ranging) Technologien. Darüber hinaus ist eine Kombination mehrerer Sensoren typisch. Nach der Erfassung erfolgt die Verarbeitung der Mobile Mapping Daten, welche sich stark an den finalen Verwendungszwecken orientiert. Üblich sind dabei die Extraktion von Merkmalen (engl. features), Änderungsüberwachung (engl. change detection) und die Erstellung von 3D Stadtmodellen. Abschließend gilt es, das Verarbeitungs- bzw. Analyseergebnis zu visualisieren. Grob lassen sich die genannten Komponenten in Hard- und Softwarekomponenten unterteilen, wobei das Erfassungssystem zu den Hardwarekomponenten zählt und die Verarbeitungs- und Visualisierungsmodule Softwareseitig realisiert sind. Abbildung 2.1 gibt einen entsprechenden Überblick.

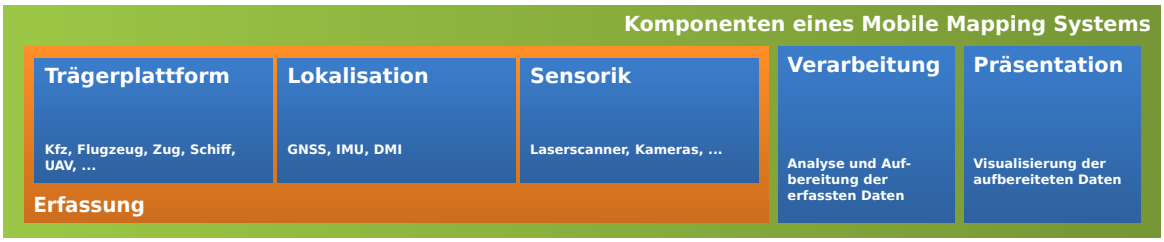


Abbildung 2.1: Typische Komponenten eines Mobile Mapping Systems.

2.1.1 Allgemeine Funktionsweise

Aufgrund des mobilen Charakters der Erfassung ist es für eine ganzheitliche Betrachtung der erfassten Daten notwendig, diese in ein übergeordnetes System zu transformieren. Dies gelingt nur, wenn für jeden Sensor die exakte Position und Orientierung im übergeordneten System zur Erfassungszeit bekannt sind. Dies kann auf vielfältige Art und Weise erfolgen. Der gängigste Ansatz besteht häufig in der Verwendung eines GNSS (Global Navigation Satellite System) Sensors zur Positionsbestimmung, sowie einer IMU (Inertial Measurement Unit) zur Erfassung der Orientierung der Sensorplattform.

Im Falle der Erfassung von Laser Messungen mittels LiDAR erfolgt die Transformation in das übergeordnete System nach Gleichung 2.1 (vgl. Luhmann (2000)). Die erfassten Messdaten liegen dabei als Polarkoordinaten mit Distanz r und Winkel α vor.

$$\begin{pmatrix} X \\ Y \\ Z \end{pmatrix} = \begin{pmatrix} X_0 \\ Y_0 \\ Z_0 \end{pmatrix} + \mathbf{R}_{\omega,\varphi,\kappa} \left(\mathbf{t} + \mathbf{R_m} \begin{pmatrix} 1 & 0 & 0 \\ 0 & \cos\alpha & \sin\alpha \\ 0 & -\sin\alpha & \cos\alpha \end{pmatrix} \begin{pmatrix} 0 \\ 0 \\ -r \end{pmatrix} \right) \tag{2.1}$$

Wobei (X_0, Y_0, Z_0) die globale Position, welche bspw. vom GNSS Sensor zum Zeitpunkt der Erfassung des Scanpunktes ermittelt wurde, repräsentiert.

$\mathbf{R}_{\omega,\varphi,\kappa}$ spiegelt die Rotationsmatrix wieder, welche aus den Drehwinkeln ω, φ und κ der IMU zum Zeitpunkt der Erfassung des Scanpunktes gebildet wird.

Der Verschiebungsvektor $\mathbf{t}$ und die Rotationsmatrix $\mathbf{R_m}$ tragen dabei der Tatsache Rechnung, dass der LiDAR Sensor gegenüber dem GNSS Sensor und der IMU versetzt montiert und anders ausgerichtet ist.

Wie die Formulierung *zum Zeitpunkt der Erfassung des Scanpunktes* andeutet, wird eine exakte zeitliche Synchronisation aller beteiligten Sensoren vorausgesetzt. Da jedoch üblicherweise GNSS basierte Sensoren zum Einsatz kommen und diese ohnehin auf genauen Laufzeitmessungen basieren, liegt somit eine hoch genaue Zeiterfassung und -synchronisation vor. Diese wird mittels eines PPS Signals (Puls per Second) an die beteiligten Sensoren weitergereicht, bzw. bei der Speicherung der erfassten Daten berücksichtigt.

2.1.2 Riegl VMX-250

Das im Rahmen dieser Arbeit genutzte Mobile Mapping System basiert auf der VMX-250 Plattform der Firma Riegl, welche auf einem Volkswagen T5 Transporter montiert ist. Das Mobile Mapping System umfasst folgende Hardware-Komponenten (vgl. Abbildung 2.2):

- VMX-250-MH Measuring Head bestehend aus
  - zwei VQ-250 Laserscannern
  - Applanix LV-510 mit IMU-31 Messeinheit
  - Trimble Zephyr Model 2 GNSS-Antenne
- VMX-250-CS6 Kamerasystem
  - zwei AVT Manta G-504 C Industriekameras
  - zwei Nikon D700 Spiegelreflexkameras
- VMX-250-DMI Distance Measurement Unit
- VMX-250-CU Kontrolleinheit

Abbildung 2.2: Komponenten der Mobile Mapping Plattform Riegl VMX-250, bestehend aus Messkopf und Kamerasystem (links), Distanz-Indikator DMI (mitte) und der Kontrolleinheit (rechts).

Einen detaillierten Überblick über die verwendete Messanordnung gibt Abbildung 2.3. Der Distanz-Indikator befindet sich dabei wie bereits in Abbildung 2.2 (mitte) zu sehen am linken Hinterrad. Das Kamerasystem sowie der eigentliche Messkopf befinden sich auf dem Dach des Transporters. Die beiden Industriekameras sind hier entgegengesetzt der Fahrtrichtung ausgerichtet, wohingegen die beiden Spiegelreflexkameras senkrecht dazu zur Seite ausgerichtet sind. Details über die verwendeten Kameras können Tabelle 2.1 entnommen werden. Die GNSS-Antenne befindet sich oben auf dem Messkopf wobei die IMU fest im Messkopf verbaut ist. Das Herzstück des Systems bilden die beiden V-förmig angeordneten Laserscanner VQ-250, welche ebenfalls fest am Messkopf verbaut sind. Bei den Scannern handelt es sich um Profil- bzw. Zeilenscanner, welche lediglich zweidimensionale Daten erfassen (Entfernung und Winkel). Die exakte Lage des erfassen Messpunktes ergibt sich durch die bereits gezeigte Transformation in das übergeordnete Koordinatensystem. Die komplette Spezifikation der Scanner kann Tabelle 2.2 entnommen werden.

	Industriekameras (Manta G-504C)	**Spiegelreflexkameras (Nikon D700)**
Sensor	2/3"color CCD (progressive scan)	CMOS 36,0 x 23,9 mm (Nikon-FX)
Pixelanzahl	5MP	12MP
Bildgröße	2452x2056	4256x2832
Bildrate	8FPS (bei 2 Kameras)	5FPS
Trigger	Zeit- oder Distanzintervall	
Speicherformat	PGM (2x2 Bayer Pattern RGGB)	JPG
Sichtfeld	80° x 65°	84° x 60°
Brennweite	5mm	20mm

Tabelle 2.1: Spezifikation der installierten Industrie- und Spiegelreflexkameras (vgl. Riegl (2012) und Nikon (2008)).

Sämtliche Komponenten sind via Kabel mit der im Kofferraum befindlichen Kontrolleinheit (siehe Abbildung 2.2 (rechts)) verbunden. Die Kontrolleinheit ermöglicht die TCP/IP basierte Kommunikation der Einzelkomponenten und die Steuerung des Systems durch einen Operator während der Messfahrt.

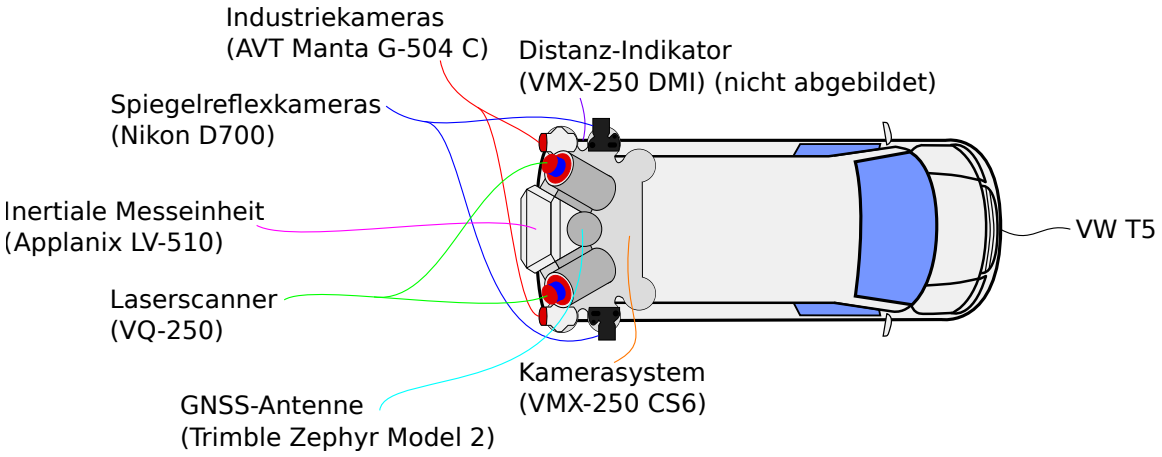


Abbildung 2.3: Übersicht über die verwendete Messanordnung.

2.1.3 Alternative Systeme

Die Analyse alternativer Systeme kann auf unterschiedlichen Systemstufen erfolgen. Betrachtet man das integrierte Gesamtsystem aus Erfassungs- und Verarbeitungskomponenten, so sind neben dem verwendeten Riegl VMX-250 System unter anderem folgende integrierte Systeme verfügbar:

- Leica Pegasus:Two (Leica (2014))
- Trimble MX8 (Trimble (2013))
- StreetMapper 360 (IGI (2014))
- Riegl VMX-450 (Riegl (2015))

Dabei unterscheiden sich die aufgezeigten integrierten Systeme in der Regel nur im Aufbau der Plattform, das grundlegende Erfassungsprinzip und die dafür benötigte Sensorik haben jedoch alle Varianten gemein. Eine stete Aktualisierung der Sensormodule erfolgt dabei genauso wie die der Softwaremodule. Das letztgenannte Riegl VMX-450 ist im Wesentlichen eine aktualisierte Version

Messprinzip	Laufzeitmessung, Echo-Signal-Digitalisierung, Online Analyse der Wellenform
Scanmechanismus	rotierender Spiegel
Sichtfeld	lückenlose 360°
Effektive Messrate	bis zu 300.000 Messungen pro Sekunde
Pulsrate	bis zu 300 kHz
Scanfrequenz	bis zu 100 Linien pro Sekunde
Wikelschrittweite $\Delta\varphi$	$0,001° \leq \Delta\varphi \leq 0,72°$
Winkelauflösung	$0,001°$
Echo Signal Intensität	16bit Intensitätsinformation
Laser Wellenlänge	nahe Infrarot
Laser Strahlbreite	7mm (Austritt) bis 36mm (100m)
Reichweite	bei natürlichen Zielen 75m (für $\rho \geq 10\%$) bis 200m (für $\rho \geq 80\%$)
Anzahl Ziele pro Puls	bis zu 5 bei Pulsrate von 300kHz
Genauigkeit	10mm
Präzision	5mm

Tabelle 2.2: Spezifikation der Riegl VQ-250 Laserscanner nach Riegl (2012).

der im Rahmen dieser Arbeit verwendeten VMX-250 Plattform. Der Hauptunterschied liegt dabei in den verbauten Laserscannern, welche in der aktualisierten Version eine höhere Reichweite, Messrate sowie Genauigkeit aufweisen.

Im Gegensatz zu den Hardwaremodulen werden die Softwaremodule häufig kontinuierlich weiterentwickelt und mit zusätzlichen Funktionen ausgestattet. Grundlegende Funktionen sind hier stets die bereits zuvor gezeigte Verarbeitung der Messdaten (bspw. Transformation ins Zielkoordinatensystem und Einfärbung der Punktwolke), sowie der Export der Resultate für den jeweiligen Einsatzzweck. Ein direkter Vergleich der verfügbaren Produkte fällt aufgrund der permanenten Weiterentwicklung eher schwer und ist für die Vorliegende Arbeit ohnehin von untergeordneter Relevanz.

2.2 Punktwolken

2.2.1 Definition

Eine Punktwolke (engl. point cloud) beschreibt eine Menge an räumlich verteilten Punkten, der Terminus Wolke unterstreicht dabei die unorganisierte Struktur (im Gegensatz zu bspw. Raster- bzw. Bilddaten) der Daten (Otepka u. a. (2013)). Neben drei Raumkoordinaten können die Punkte diverse weitere Attribute, wie beispielsweise Punktnormale, Farbe, Intensität, u.ä. besitzen.

2.2.2 Abgrenzung zu vermaschten Punkten

Im Gegensatz zur komplett unorganisierten Struktur einer Punktwolke werden Punkte häufig zu einem Polygonnetz vermascht. Dieses Netz, häufig in Form eines unregelmäßigen Dreiecksnetzes (engl. triangulated irregular network, TIN), repräsentiert i.d.R. die Oberfläche eines erfassten Objektes. Somit liegt eine kontinuierliche Beschreibung der Oberfläche vor, wohingegen eine reine Punktwolke selbige nur mittels entsprechend diskreter Punkte widerspiegelt. Dies hat direkten Einfluss auf Analyse- und Visualisierungstechniken. Somit können Verfahren und Visualisierungen, welche auf Polygonnetzen operieren nur bedingt auf reine Punktwolken übertragen werden.

2.2.3 Speicherformate von Punktwolken

Sowie bei der Verarbeitung, als auch bei der Visualisierung von Punktwolken spielt das verwendete Speicherformat eine wichtige Rolle. In Abhängigkeit der genutzten Verarbeitungsschritte werden die Punktwolkendaten wiederholt eingelesen und am Ende eines Verarbeitungsschrittes wieder persistent gespeichert.

Mögliche Speicherformate lassen sich in zwei Klassen unterscheiden, text-basiert und binär. Textbasierte Speicherformate lassen sich verhältnismäßig einfach erzeugen, wieder einlesen und es besteht darüber hinaus sogar die Möglichkeit die Daten über einen gewöhnlichen Texteditor zu ändern. Binäre Formate speichern dahingegen die jeweiligen Werte direkt in ihrer binären Repräsentation, also exakt so wie sie bereits im Arbeitsspeicher vorliegen. Dies hat gegenüber textbasierten Formaten den Vorteil, dass weder eine textuelle Repräsentation der Daten erzeugt, noch eine vorliegende textuelle Repräsentation interpretiert, also in die benötigte binäre Repräsentation umgewandelt werden muss. Somit können binäre Formate üblicherweise schneller gelesen und geschrieben werden und benötigen i.d.R. weniger Speicherplatz. Ein solches binär Format lässt sich jedoch nicht mehr mit einem gewöhnlichen Texteditor betrachten oder gar editieren.

Darüber hinaus lassen sich Speicherformate nach ihrer Flexibilität einordnen. Flexibilität meint in diesem Kontext die Möglichkeit, beliebige Punktattribute oder gar über Punktdaten hinaus weitere Informationen zu umfassen. Abschließend spielt eine möglichst breite Unterstützung durch vorhandene Programmbibliotheken und Visualisierungstools eine entscheidende Rolle bei der Wahl des genutzten Speicherformats.

2.2.3.1 VRML/X3D Speicherformat

VRML steht für *Virtual Reality Modeling Language* und wurde bereits 1995 als Beschreibungssprache für virtuelle 3D-Szenen entwickelt und später nach der Spezifikation als VRML97 (Carey u. a. (1997)) als ISO-Standard etabliert. Es wurde als textbasiertes und für Menschen lesbares Format konzipiert. Neben den eigentlichen Geometrien konnten darüber hinaus Lichtquellen, Animationen und Interaktionsmöglichkeiten definiert werden. Listing 2.1 zeigt die Definition eines Gebäudepolygons im VRML Schema. Zeilen 8 bis 17 definieren die 10 Eckpunkte des Gebäudes, wohingegen die Zeilen 21 bis 27 die jeweiligen Polygonflächen basierend auf den Punktindizes spezifizieren. 2005 wurde VRML durch das Web3D Konsortium um eine XML-basierte Syntax erweitert und unter dem Namen X3D (*Extensible 3D*) als ISO-Standard (Web3D-Consortium (2005)) spezifiziert.

Obgleich VRML und das nachfolge Format X3D etablierte und gut unterstützte Formate darstellen, sind sie für die persistente Speicherung von umfangreichen Punktwolkendaten ungeeignet. Die notwendigen Umwandlungen von Binärformat in Text und umgekehrt machen ein performantes Lesen und Schreiben der Daten unmöglich. Darüber hinaus bläht die textuelle Repräsentation das ohnehin bereits große Datenvolumen weiter auf. Die durch X3D eingeführte XML Syntax verstärkt diesen Fakt sogar noch. Die genannten Argumente lassen quasi auf alle textbasierten Speicherformate, wie beispielsweise (City-)GML ((City) Geography Markup Language, Open-GIS-Consortium-Inc. (2002) und Open-Geospatial-Consortium (2012)), übertragen. Insgesamt sind diese Formate besser für die Beschreibung einzelner Objekte (wie Gebäude, Grundstücke, Straßen, etc.) geeignet, als für Massendaten mit vielen Punktmessungen.

2.2.3.2 XYZ Speicherformat

Das XYZ Speicherformat stellt im Grunde kein vollständig spezifiziertes Format dar und wird je nach Kontext unterschiedlich interpretiert (neben Geodaten z.B. auch als Positionen von Atomen in Molekülen in der Computerchemie). Bei diesem Format werden die zu speichernden Daten einfach binär hintereinander in die entsprechende Datei geschrieben. Wie der Name suggeriert wird hier von drei räumlichen Koordinaten ausgegangen. Diese Art der persistenten Speicherung ist im Gegensatz zu den zuvor vorgestellten textbasierten Formaten sehr effizient, was das Datenvolumen, als auch die Lese- und Schreibgeschwindigkeit angeht.

```
Shape {
  appearance Appearance {
    material Material {diffuseColor 1 1 0}
  }
  geometry IndexedFaceSet {
    coord Coordinate {
      point [
        -1 0  2,
        -1 0 -2,
         1 0 -2,
         1 0  2,
        -1 1  2,
        -1 1 -2,
         1 1 -2,
         1 1  2,
         0 2  2,
         0 2 -2
      ]
    }
    coordIndex [
      0, 1, 2, 3, -1,
      0, 3, 7, 8, 4, -1,
      1, 5, 9, 6, 2, -1,
      2, 6, 7, 3, -1,
      0, 4, 5, 1, -1,
      7, 6, 9, 8, -1,
      4, 8, 9, 5, -1
    ]
  }
}
```

Listing 2.1: VRML Beschreibung eines Gebäudes mit 10 Eckpunkte und 7 Flächen.

Da jedoch keinerlei Metadaten (wie z.B. Koordinatensystem und -ursprung), beispielsweise in Form eines Kopfteils (engl. Header) gespeichert werden, ist dieses Format jedoch nicht sehr flexibel. Zwar können Informationen wie zum Beispiel die Punktanzahl leicht aus der Dateigröße abgeleitet werden, jedoch ist eine genaue Definition der enthaltenden Daten (bspw. der verwendete Datentyp) unmöglich. Sollen mehr als nur die Koordinaten, wie Farbe oder Punktnormale, gespeichert werden, so lässt sich dies auf Grund der fehlenden Metainformationen später nicht mehr rekonstruieren.

2.2.3.3 PLY Speicherformat

PLY steht für *Polygon File Format* und wurde von Turk (1994) entwickelt. Es definiert einen textbasierten Kopfteil und wahlweise einen binären oder textbasierten Datenteil. Im Kopf sind sämtliche Metainformationen bezüglich der gespeicherten Daten, wie Datenfelder und Datentypen, spezifiziert. Zu den Geometriedefinitionen von Punkten, Linien und Polygonen, lassen sich zusätzlich zu den üblichen Attributen, wie Farbe oder Normale, auch eigene Attribute definieren. Darüber hinaus lassen sich sogar eigene Elemente mit eigenen Attributen definieren. Da sämtliche Elemente und Attribute im Kopf spezifiziert sind, kann jedes Programm jede Datei im PLY Format lesen. Kann ein Programm selbst definierte Elemente oder Attribute nicht interpretieren, so können diese einfach übersprungen werden, da durch die einleitende Definition klar ist, wie die Daten im Datenblock zusammengesetzt sind.

Listing 2.2 zeigt beispielhaft die elementweise Datendefinition im Kopfteil. Zeile 1 stellt dabei die bei Speicherformaten übliche *Magic Number* zur Kennzeichnung des Dateityps dar. Zeile 2 spezifiziert das Format des Datenblocks, hier als binär (in Big Endian), alternative Formate wären binär (in Little Endian) und „ascii“als textbasiertes Format. Zeile 3 zeigt einen Kommentar, wohingegen Zeile 4 ein Vertex (Punkt Geometrie) Element mit der zugehörigen Anzahl an Einträgen (630879) definiert. Die Zeilen 5 bis einschließlich 14 spezifizieren die Attribute des Vertex Elements und deren Datentypen, das letzte Attribut *class* stellt ein nutzerdefiniertes Attribut dar. Das *face* Element (Zeilen 15 und 16) definiert etwaige Polygone, wobei die gegebene Datei keine enthält (Anzahl 0). Das *end_ header* in Zeile 17 schließt den Kopfteil ab, gefolgt (Zeile 18) vom Datenteil im zuvor definierten Format.

```
1  ply
2  format binary_big_endian 1.0
3  comment generator: de.hannover.uni.ikg.core.fileformats.ply.PLYWriter
4  element vertex 630879
5  property float x
6  property float y
7  property float z
8  property uchar red
9  property uchar green
10 property uchar blue
11 property float nx
12 property float ny
13 property float nz
14 property int class
15 element face 0
16 property list uchar int vertex_index
17 end_header
18 [binary data ...]
```

Listing 2.2: PLY Kopfteil mit Definition der Datenelemente und deren Attribute.

Die gezeigte Möglichkeit eigene Attribute oder gar ganze Elementstrukturen zu definieren und dennoch auf existierende Programmbibliotheken und Visualisierungsprogramme zurückgreifen zu können, bietet die für eigene Verarbeitungsmodule für Punktwolkendaten notwendige Flexibilität. Die darüber hinaus vorhandene Möglichkeit der effizienten Speicherung in binärer Form machen das PLY Speicherformat für alle Schritte einer Verarbeitungskette von Punktwolkendaten praktikabel und wurde daher im Rahmen dieser Arbeit als das Standard Speicher- und Austauschformat über alle implementierten Verfahren und Module hinweg eingesetzt. Damit bildet das PLY Datenformat die erste beschriebene Komponente des Speicherformatemoduls. Eine Einordnung bezüglich anderer Module gibt Abbildung 2.4. Neben PLY umfasst dieses Modul weitere Speicherformate wie SDC (Scan-Data-Computed), POF (Position and Orientation File) und PNG (Portable Network Graphic), welche in späteren Kapiteln näher erklärt werden.

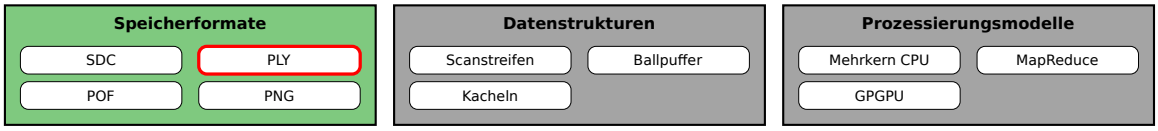


Abbildung 2.4: Einordnung des PLY Formates in das Kernmodul Speicherformate des Frameworks.

2.2.4 Visualisierungstechniken

Visualisierungstechniken beschreiben die Art der Darstellung von beliebigen 3D Modellen. Die möglichen Darstellungstechniken sind dabei eng mit den jeweiligen 3D Modellen verknüpft. Liegt ein 3D

Modell in Form einer vermaschten Punktwolke vor (vgl. Abschnitt 2.2.2, so kann das entsprechende Polygonnetz zur Darstellung genutzt werden. Dafür werden die einzelnen Polygone (i.d.R. in Form von Dreiecken) gezeichnet. Für die Erhöhung des virtuellen Detailgrades kann das Polygonnetz mit einer entsprechenden Textur belegt werden.

Da die Datengrundlage dieser Arbeit aus rohen Punktwolken besteht, liegt zunächst kein solches Polygonnetz vor. In Abhängigkeit vom Visualisierungsszenario ist eine Vermaschung der Punktwolke zu einem Polygonnetz unter Umständen nicht erwünscht oder zu aufwendig. In diesen Fällen bieten sich punktbasierte Verfahren an, welche auf die aufwendige Erzeugung von speziellen 3D Modellen verzichten und direkt Punktwolken visualisieren. punktbasierte Verfahren sind jedoch nicht in jedem Fall effizient, beispielsweise bei der Darstellung einfacher Oberflächen wie Ebenen. Dies lässt sich jedoch durch die Kombination von punkt- und polygonbasierten Verfahren optimieren. Ein solches hybrides Verfahren benötigt jedoch ein gewisses Maß an Vorverarbeitung und Modellerzeugung, kann aber die Speicher- und Darstellungsgeschwindigkeiten positiv beeinflussen. Ist der Datenumfang sehr groß oder das zur Visualisierung genutzte Gerät nicht sehr leistungsfähig, so bieten sich andere Techniken an, die sowohl die Daten, als auch deren Darstellung dahingehend optimieren. Als Vertreter einer solch hoch optimierten Darstellungstechnik wurde in dieser Arbeit das Parallax Scrolling Verfahren herangezogen. Im Gegensatz zu den punktbasierten Verfahren, welche die rohen Punktdaten visualisieren, befindet sich diese Darstellungsform am anderen Ende des Spektrums, da es eine umfassende Modellerzeugung voraussetzt. Dies ermöglicht die Darstellung umfangreicher Datensätze selbst auf leistungsschwachen Endgeräten.

2.2.4.1 Point-Based Rendering (PBR)

Point-based Rendering, also die punktbasierte Bilderzeugung, stellt im Gegensatz zu den üblichen polygonbasierten Ansätzen den Punkt als grundlegende Geometrie in den Mittelpunkt. Den Kern dieser Technik entwickelten bereits Csuri u. a. (1979). Sie wurde von Levoy und Whitted (1985) und später von Grossman und Dally (1998) wieder aufgegriffen und weiterentwickelt. Zunehmend populärer wurde diese Art der Visualisierung mit der weiten Verbreitung von Sensoren welche Objekte bzw. deren Oberflächen punktuell erfassen und damit leicht umfangreiche Punktdatensätze (Punktwolken) generieren. Die erste weitgehend prominente Implementierung der PBR Technik stellt QSplat von Rusinkiewicz und Levoy (2000) dar.

Sainz und Pajarola (2004) geben einen Überblick über existierende PBR Varianten und Implementierungen und stellen darüber hinaus einen Vergleich der Rendergeschwindigkeiten und visuellen Resultate an. Beides wird im Wesentlichen sowohl von der Repräsentation der Punkte, als auch von der verwendeten Level Of Detail Technik (vgl. Abschnitt 2.2.5) getrieben. Wie Abbildung 2.5 verdeutlicht, lassen sich Punkte grob auf zwei Art und Weisen (visuell) repräsentieren (engl. splatting). Zum einen als Quadrat welches üblicherweise zur Bildschirmebene hin ausgerichtet ist und im Gegensatz dazu als eine kreisförmige oder elliptische Scheibe, welche senkrecht zur Normalen des jeweiligen Punktes ausgerichtet ist. Ersteres kann in der Regel als Punktprimitiv von den meisten Grafikbibliotheken (wie bspw. OpenGL) dargestellt werden. Dies ist sehr effizient, da hier keine Polygone gefüllt werden müssen, sondern lediglich für jeden Punkt ein Quadrat gezeichnet wird. Dies ermöglicht eine sehr hohe Rendergeschwindigkeit, was der Vergleich von Sainz und Pajarola (2004) bestätigt. Im Falle der elliptischen Scheiben, welche senkrecht zur Punktnormalen ausgerichtet sind, ist dieser Ansatz jedoch nicht möglich. Die genutzten Ellipsen müssen wie alle darzustellenden Oberflächen durch ein Dreiecksnetz approximiert werden. Abschließend werden dann die elliptischen Polygone gefüllt, was zu einer deutlich geringeren Rendergeschwindigkeit führt. In Abhängigkeit davon wie gut die elliptischen Scheiben approximiert werden, ist diese Form der punktbasierten Darstellung sogar langsamer als das Rendern eines äquivalenten Modells in Form eines Dreiecksnetzes. Alternativ lässt sich eine Ellipse auch über ein texturiertes Dreieck darstellen, was gegenüber der besagten Polygonapproximation eine leicht höhere Rendergeschwindigkeit erlaubt. Betrachtet man das visuelle Ergebnis beider Ansätze, so stellen Sainz und Pajarola (2004) fest, dass letzter die qualitativ hochwertigste Darstellung produziert. Jedoch stellen sie ebenfalls heraus, dass der Qualitätsunterschied bei weitem nicht so hoch wie der Performanzunterschied ausfällt.

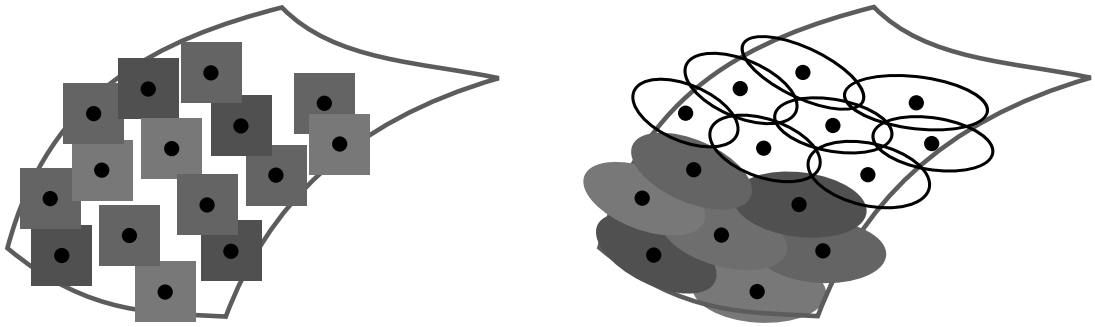


Abbildung 2.5: Visuelle Repräsentation (splatting) von Punktwolken: einfaches Quadrat zur Bildschirmebene (links) und elliptische Scheiben senkrecht zur Punktnormalen (rechts).

Neben der hohen Rendergeschwindigkeit liegt der Vorteil des PBR darin, dass keine Polygonmodelle, sieht man von etwaigen elliptischen Polygonen für einzelne Punkte ab, aus den Punktdaten erzeugt werden müssen. Die Erstellung eines Dreiecksnetzes aus 2D bzw. 2,5D Daten ist mittels einer Delaunay-Triangulierung grundsätzlich problemlos möglich. Geht man wie beschrieben von beliebigen Szenen mit beliebigen Objekten aus, so ist die Erzeugung eines Polygonmodells aus diesen Punkdaten sehr viel anspruchsvoller und fehlerträchtiger. Grundsätzlich ist das Feld der Oberflächenrekonstruktion aus Punktdaten umfangreich erforscht. Als Lösung existieren vielfältige Ansätze begonnen bei *Marching Cubes* von Lorensen und Cline (1987), über *Ball-Pivoting* von Bernardini u. a. (1999) bis hin zu *Poisson Surface Reconstruction* von Kazhdan u. a. (2006). Geht man von ungleichmäßig verteilten, unvollständigen, verrauschten und mit Ausreißern behafteten Rohdaten aus, so erfordern alle Verfahren eine umfangreiche Datenvorverarbeitung. Trotzdem sind die Resultate der vollautomatischen Oberflächenrekonstruktion häufig von schwankender Qualität (Tang u. a. (2013)). Diese Problematik kann mit PBR Techniken nahezu vollständig umgangen werden.

Eine Erweiterung der eingangs erwähnten QSplat Technik stammt von Richter und Döllner (2010). Sie erweitern die QSplat Technik um eine Out-of-Core (engl. für ausserhalb des Arbeitsspeichers) Fähigkeit zur Unterstützung massiver Punktwolken, die nicht komplett im Arbeitsspeicher vorgehalten werden können. Als Datenstruktur findet, wie bei QSplat, ein Octree Verwendung, welcher beim Rendering traversiert wird. Diese Traversierung wurde ebenfalls optimiert. Erreicht wird dies durch die Wiederverwendung der Liste an Knoten des letzten Renderingschrittes, genannt *rendering front*, was die erneute Traversierung des Octrees signifikant verkürzt und somit höhere Bildwiederholraten ermöglicht.

2.2.4.2 Hybrides (Point-Based) Rendering

Erstmals vorgestellt von Chen und Nguyen (2001), vereint hybrides Rendering Punkt- und Polygonbasierte Modelle bzw. Darstellungstechniken. Wie im vorherigen Abschnitt beschrieben können Punktwolken mittels PBR effizient dargestellt werden, wobei darüber hinaus auf eine aufwendige Vorverarbeitung der Daten weitestgehend verzichtet werden kann. Enthält eine Punktwolke leicht beschreibbare Oberflächen wie Ebenen, so stellt eine punktbasierte Repräsentation eine prinzipiell ineffiziente Repräsentation dar. Für die Darstellung einer annähernd geschlossenen, ebenen Fläche werden verhältnismäßig viele Punkte benötigt, was den Datenumfang erhöht und die Rendergeschwindigkeit senkt. Effizienter ist es, die besagte Ebene durch ein entsprechendes Polygon anstatt durch einzelne Punkte darzustellen. Dies ermöglicht eine äquivalente geometrische Repräsentation, bei stark reduziertem Datenumfang und erhöhter Rendergeschwindigkeit. Für eine adäquate Farbrepräsentation der ursprünglichen Punktdaten wird das Polygon mit einer Textur belegt. Daher werden solche Modelle treffenderweise häufig als *Billboards* bezeichnet.

Abzugrenzen ist die *Billboard* von der *Imposter* Technik von Maciel und Shirley (1995), sowie Wimmer u. a. (2001). Im Gegensatz zu Billboard Modellen werden bei der Imposter Technik Blickwinkelabhängige Repräsentationen der Punktwolke erzeugt. Billboard Modelle weisen diese Blickwin-

kelabhängigkeit nicht auf und eigenen sich daher besser für die Darstellung umfangreicher Punktwolken. Während Decoret u. a. (2003) den Billboard Ansatz umfassend beschreiben, konzentrieren sich Wahl u. a. (2005) auf das Finden von Ebenen welche durch texturierte Polygone ersetzt werden können. Wahl u. a. (2005) erreichen dabei eine bis zu vierzig-fach höhere Rendergeschwindigkeit im Vergleich zu den zuvor vorgestellten Splatting Techniken.

2.2.4.3 Parallax Scrolling

Die Parallax Scrolling Technik (zu deutsch: Bewegungsparallaxe) stellt eine Szene zusammengesetzt aus mehreren hintereinander gelagerten Abbildungen dar. Um Darstellungen aus den hinteren Ebenen sehen zu können sind die leeren Bereiche der vorderen Abbildungen transparent. Die Besonderheit dieser Darstellungsform liegt in der Erzeugung einer Tiefenillusion. Dies wird im Wesentlichen durch drei Effekte erzielt. Zunächst erzeugt die teilweise Überlappung von einzelnen Objekten eine Vordergrund-Hintergrund Relation. Verdeckte Objekte werden somit als *im Hintergrund* eingestuft. Der zweite Effekt beruht auf einer perspektivischen Darstellung der Objekte. Objekte in hinteren Ebenen werden verkleinert dargestellt, was der natürlichen Wahrnehmung entspricht. Den stärksten Einfluss auf die beabsichtigte Tiefenillusion hat jedoch das Verschieben der Ebenen mit unterschiedlichen Geschwindigkeiten. Dabei werden Ebenen im Vordergrund mit einer höheren Geschwindigkeit verschoben, als Hintergrundebenen. Dies entspricht ebenfalls der natürlichen Wahrnehmung des Menschen.

Die Technik fand erstmals Anwendung bei der Multiplan Kamera von Disney (1937). Der Fokus lag hier jedoch primär auf der Wiederverwendbarkeit der Hintergrundbilder. Später wurde die Parallax Scrolling Technik in vielen Computerspielen der 80er und 90er Jahre eingesetzt, verlor jedoch mit dem Aufkommen von 3D fähiger Grafikhardware zunehmend an Bedeutung. Eine Wiederbelebung der Technik fand mit der Einführung der ersten Smartphone- und Tabletgeräte statt. Aufgrund der zunächst eingeschränkten Leistungsfähigkeit nutzen Spiele für diese Plattformen ebenfalls Parallax Scrolling. Abbildung 2.6 zeigt die Darstellungsform am Beispiel des Spiels Angry Birds, welches für die genannten Plattformen entwickelt wurde. Zur Verdeutlichung der eingesetzten Technik wurde jeder Ebene eine eigene Farbe zugewiesen, von Rot über Blau und Grün bis hin zu Gelb für die nicht transparente Hintergrundebene.

Der Einsatz dieser Technik für die Darstellung von Punktwolken bedarf, wie eingangs beschrieben, einer umfassenden Modellerzeugung. Da lediglich hintereinander gelagerte Abbildungen dargestellt werden, müssen die Punkte der Punktwolke ähnlich wie bei der Billboard und Imposter Technik bei der Modellerzeugung auf entsprechende Bildebenen projiziert werden.

Abbildung 2.6: Tiefenillusion mittels Bewegungsparallaxe im Spiel Angry Birds

2.2.5 Level Of Detail

Die Darstellung komplexer und hoch detaillierter Modelle ist in der Regel sehr rechenintensiv. Daher sinkt die Bildwiederholrate mit steigendem Detailgrad unter Umständen soweit ab, dass keine flüssige Darstellung (15 Bilder pro Sekunde) mehr möglich ist. Um diesem Problem entgegenzuwirken, muss ein Kompromiss zwischen Darstellungsgeschwindigkeit und -detailgrad gefunden werden. Dies wird als Level Of Detail Technik (kurz: LOD, deutsch: Detailgrad) bezeichnet. Das Grundprinzip ist dabei denkbar einfach: Soll ein Modell, bzw. eine Szene, gerendert werden, so werden für kleine, weit entfernte oder unwichtige Teile oder Objekte Repräsentationen mit reduziertem Detailgrad verwendet. Luebke u. a. (2002) unterscheiden drei wesentliche Ansätze: diskretes, kontinuierliches und blickwinkelabhängies LOD.

Diskretes LOD stellt den traditionellen Ansatz dar, welcher auf Clark (1976) zurück geht und nach wie vor von vielen Visualisierungsframeworks genutzt wird. Dabei werden im Rahmen eines Vorverarbeitungsschritts mehrere Repräsentationen eines Objekts erstellt. Jede der erstellten Varianten repräsentiert dasselbe Objekt, jedoch mit einem unterschiedlichen Detailgrad. Zum Zeitpunkt des Renderings des Objektes muss lediglich entschieden werden, welche der Varianten dargestellt wird. Da die Erstellung der unterschiedlichen LOD Varianten vor dem eigentlichen Rendering stattfindet, unterliegt das genutzte Vereinfachungsverfahren keinen Laufzeitrestriktionen.

Kontinuierliches LOD weicht die Entkopplung zwischen Rendering und Vereinfachung des Objekts, wie sie beim diskreten LOD vorliegt, auf. Anstatt mehrerer eigenständiger Repräsentationen eines Objektes zu erzeugen, wird die Vereinfachung des Objekts auf eine Datenstruktur abgebildet, welche ein mehr oder weniger kontinuierliches LOD Spektrum für jedes Objekt simuliert. Der zum Zeitpunkt der Darstellung benötigte Detailgrad wird dann aus der besagten Struktur extrahiert. Diese Form des LOD erlaubt eine feinere Granularität, da nicht aus vordefinierten Stufen gewählt werden muss, sondern das Objekt den Anforderungen entsprechend kontinuierlich verfeinert werden kann. Da diese progressive Darstellung leicht unterbrochen und wieder fortgeführt werden kann, eignet sich dieser Ansatz besonders für Szenarien, bei denen umfangreiche Modelle zunächst von der Festplatte oder über ein Netzwerk geladen werden müssen.

Eine Erweiterung des kontinuierlichen Ansatzes stellt das blickwinkelabhängige LOD dar. Dabei wird der genutzte LOD in Abhängigkeit des aktuellen Blickwinkels auf ein Objekt dynamisch bestimmt. Dies wird als richtungsabhängig (engl. anisotropic) bezeichnet und bedeutet, dass Teile ein und desselben Objekts in unterschiedlichen LODs dargestellt werden können. Während also nahe gelegene Teile mit einem hohen Detailgrad dargestellt werden, können gleichzeitig entfernte Teile desselben Objekts in einem geringeren Detailgrad vorliegen. Dies erlaubt eine noch feinere Granularität bei der Darstellung eines Modells, erhöht jedoch gleichzeitig den Rechenaufwand zur Bestimmung, welcher Teil, welches Objekts in welchem LOD gerendert werden soll.

2.2.5.1 Virtuelle Kamera und View Frustum

Im Kontext von Renderingframeworks im Allgemeinen und LOD-Techniken im Besonderen ist das Konzept der genutzten virtuellen Kamera von Bedeutung. Die Parameter einer virtuellen Kamera (Position, Orientierung, Öffnungswinkel, usw.) bestimmen die Erscheinung des darzustellenden Objektes im sog. Viewport, welches den Bereich bzw. das Fenster auf dem Bildschirm beschreibt, in dem das gerenderte Modell bzw. Objekt angezeigt wird. Dies ist im Wesentlichen dem Sichtkegel des menschlichen Auges nachempfunden. Da ein Viewport üblicherweise rechteckig ist, wird aus dem Sichtkegel eine Sichtpyramide. Ausschließlich Objekte, welche sich in dieser Sichtpyramide befinden oder diese schneiden sind in der finalen Darstellung im Viewport sichtbar. Objekte, die sich komplett außerhalb befinden, sind nicht sichtbar und können daher vom Renderingprozess ausgeschlossen werden, was Ressourcen spart. Wie Abbildung 2.7 verdeutlicht, werden für die besagte Sichtpyramide zwei Ebenen, Nah und Fern, definiert. Dabei werden Objekte ebenfalls ausgeschlossen, welche sich vor der Nah-Ebene oder hinter der Fern-Ebene befinden. Dieser (Sicht-)Pyramidenstumpf (blau) wird als View Frustum bezeichnet. LOD-Techniken nutzen dieses Konzept zur Ermittlung der Ob-

jektentfernung und damit zur Bestimmung, welches Objekt bzw. Objektteil mit welchem Detailgrad gerendert werden soll.

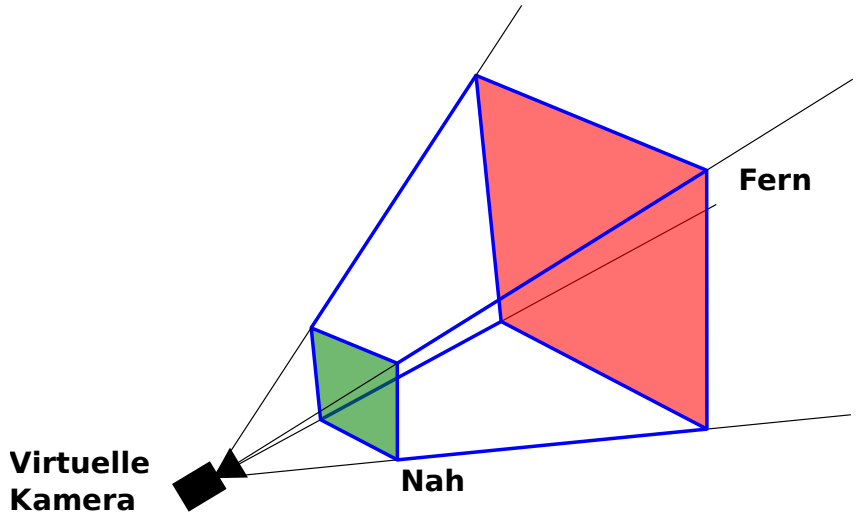


Abbildung 2.7: Virtuelle Kamera und View Frustum als (Sicht-)Pyramidenstumpf (blau).

2.3 Farbmodelle

Farbe ist eine für den Menschen durch Licht erzeugte visuelle Wahrnehmung. Das menschliche Auge besitzt drei farbempfindliche Sinneszellen, welche als Zapfen bezeichnet werden. Ein Farbreiz kann somit als eine Kombination dieser drei Sinneszellen verstanden werden. Auf Basis dieser Kombination lassen sich beliebige Farben über sogenannte Farbmodelle, welche üblicherweise ebenfalls dreidimensional sind, beschreiben. Farbmodelle lassen sich unter anderem in folgende Typen kategorisieren: *technisch-physikalisch*, *perzeptuell* und *sonstige* (Fairchild (2013)). Technisch-physikalische Farbmodelle wie RGB (Rot-Grün-Blau) oder CMYK (Cyan-Magenta-Yellow-Key) dienen in der Regel dazu den gewünschten Farbreiz technisch zu erzeugen, beispielsweise mittels eines Bildschirms oder Druckers. Perzeptuelle Modelle orientieren sich an der menschlichen Wahrnehmung und nutzen Parameter wie Farbton, Sättigung und Helligkeit für die Beschreibung einer Farbe. Der prominenteste Vertreter ist das HSV (Hue-Saturation-Value) Farbmodell und seine Derivate HSB (Hue-Saturation-Brightness) und HSI (Hue-Saturation-Intensity). Aufgrund der Anlehnung an die menschliche Wahrnehmung eignet sich dieses Modell sehr gut für etwaige Farbanpassungen zum Ausgleich wahrnehmbarer, jedoch ungewollter Farbunterschiede. Beispielsweise kann die Helligkeit einer Farbe problemlos über den entsprechenden Parameter angepasst werden, wobei der Farbton und die Sättigung konstant bleiben. Eine solche Anpassung ist im technischen RGB Farbmodell nicht ohne weiteres möglich.

2.3.1 Farbanpassung

Eine Möglichkeit für Punkte einer Punktwolke einen Farbwert zu ermitteln, ist die Aufnahme von Fotos des Realweltobjektes, idealerweise zum Erfassungszeitpunkt der Objektgeometrie. Bei mehreren Objekten oder großflächigeren Erfassungen wie beim Mobile Mapping sind jedoch mehrere Fotos notwendig, um möglichst für alle Objekte bzw. Punkte Farbwerte ermitteln zu können. Dabei gibt es unweigerlich benachbarte Regionen, deren Farbwerte aus unterschiedlichen Bildern extrahiert wurden. Dies führt wiederum zu deutlich sichtbaren Farbübergängen in diesen Bereichen. Um diese Farbübergänge zu beseitigen kann eine Farbanpassung durchgeführt werden.

El-Hakim u. a. (1998) adressieren dieses Problem mittels einer globalen, sowie lokalen Grauwertanpassung. Grundlage bildet ein für ein Realweltobjekt erfasstes Dreiecksnetz. Die Farbwerte der einzelnen Dreiecke wurden dabei aus während der Erfassung aufgenommenen Bildern extrahiert. Im

Rahmen der globalen Grauwertanpassung werden Grauwertänderungen für jedes genutzte Bild bestimmt. Zunächst werden dazu aneinander grenzende Regionen identifiziert welche aus unterschiedlichen Bildern eingefärbt wurden. Anschließend werden sämtliche Grauwertdifferenzen von benachbarten Dreiecken entlang der Regionengrenzen ermittelt. Diese Differenzen werden über die Kleinste Quadrate-Methode ausgeglichen und ein entsprechender Wert für die Grauwertanpassung pro Bild berechnet. Zusätzlich zur globalen Anpassung der Grauwerte wird eine lokale Anpassung der Farbe jedes Dreiecks durchgeführt. Dafür wird iterativ die Grauwertdifferenz zu allen Nachbarn eines Dreiecks, abermals über die Kleinste Quadrate-Methode ausgeglichen. Dies führt zu deutlich glatteren Farbverläufen bei benachbarten Dreiecken.

2.4 Verteiltes Rechnen

Komplexe Problemstellungen und umfangreiche Daten können von einzelnen Computersystemen häufig nicht mehr verarbeitet werden. Die Gründe dafür liegen üblicherweise in der Beschränktheit der Ressourcen eines einzelnen Systems. Diese Betrachtung lässt sich auf das Gesamtsystem (gesamter Computer), als auch auf einzelne Systemkomponenten (bspw. Prozessor) anwenden. Betrachtet man die Prozessorenentwicklung der letzten Jahre, so fällt auf, dass die Geschwindigkeit einzelner Prozessorenkerne seit Jahren nahezu stagniert. Um dennoch immer Leistungsfähigere (Prozessor)Systeme zu entwickeln wurde der Fokus auf die Integration mehrerer Kerne in einen Prozessor gelegt. Somit kann die Rechenlast auf mehrere Prozessoren *verteilt* werden. Dieses Konzept skaliert jedoch nur bis zu einem gewissen Grad, da die Anzahl der Kerne für ein einzelnes Computersystem technisch begrenzt ist. Um eine stärkere Skalierbarkeit zu Erreichen muss das Konzept der Verteilung der Rechenlast auf das gesamte Computersystem angewendet werden. Somit wird das Problem nicht von einem einzigen Computer verarbeitet, sondern von einer Vielzahl. Dieser Zusammenschluss von Computersystemen (im folgenden Knoten genannt) wird als verteiltes System bezeichnet (Tanenbaum und Van Steen (2002)).

Für die Realisierung eines Verteilten Systems müssen diverse Komponenten adressiert werden. Diese reichen von Verwaltungsaufgaben, über Speicherverwaltung, Kommunikation und Synchronisation bis hin zur Aufgabendefinition. Obgleich jede der genannten Komponenten auf individuellen Lösungen beruhen kann, ist eine ganzheitliche Lösung aus Komplexitäts- und Kompatibilitätsgründen vorzuziehen. Eine solche ganzheitliche Lösung stellt *Apache Hadoop* dar. Hadoop ist ein Javabasiertes Framework zur Realisierung eines Verteilten Systems. Die Kernkomponente bildet dabei das Hadoop Distributed File System (HDFS) welches als verteiltes Dateisystem den Zugriff und die Verwaltung der persistenten Daten bereitstellt. Als Aufgaben- bzw. Programmiermodell nutzt Hadoop das von Dean und Ghemawat (2008) entwickelte MapReduce-Verfahren.

Das HDFS zerlegt die zu persistierenden Daten in Blöcke gleicher Länge und verteilt diese Datenblöcke redundant auf mehrere Knoten. In Abhängigkeit von der Art der zu speichernden Daten ist die Zerlegung in gleichgroße Blöcke unter Umständen problematisch. Zerteilt man Daten denen ein Abschnitt mit Metadaten vorangestellt, wie beispielsweise der Header des PLY Formates, fehlt, bis auf dem ersten Block, der besagte Metadaten Abschnitt allen Datenblöcken. Darüber hinaus ist nicht klar wo die Daten korrekt geteilt werden können, um sie später unabhängig von den restlichen Blöcken wieder einlesen zu können. Dieses Problem adressiert das HDFS mit einem speziellen Format (Avro-Format) welches eine spezielle Schema-Definition (Avro-Schema) beinhaltet (vgl. Abschnitt 3.1.2.3). Aus diesem Grund kann das zuvor als Standardaustauschformat festgelegte PLY Format nicht für die Verarbeitung mittels Hadoop genutzt werden. Daher müssten die zu verarbeitenden Daten zunächst in das besagte Avro-Format überführt werden.

Das von Hadoop verwendete MapReduce Konzept basiert im Wesentlichen auf drei Schritten: *Map*, *Shuffle* und *Reduce*. Jeder Eingangsdatenblock wird dabei auf eine Map-Funktion verteilt. Die Resultate der Map-Funktion werden mittels des Shuffle-Schritts entsprechend zu Zwischenergebnissen gruppiert und an die Reduce-Funktion übergeben, welche das eigentliche Endergebnis berechnet. Sowohl Map- als auch Reduce-Funktionen sind dabei Nutzerspezifisch. Die Parallelisierung äussert sich darin, dass so viele Map-Funktionen ausgeführt werden, wie es Datenblöcke gibt. Analog dazu entspricht die Anzahl der Reduce-Funktionen der Anzahl der Zwischenergebnisse des Shuffle-Schritts.

Ist ein Algorithmus zu komplex um ihn mit genau einer Map- und einer Reduce-Funktion abzubilden, so können beliebig viele Map- und Reduce-Schritte konkateniert werden.

2.5 Verdeckungsanalyse

Nahezu unabhängig von der Erfassungsmethodik stellen Verdeckungen ein signifikantes Problem bei der Verarbeitung der Daten, sowie der Erzeugung von 3D Modellen dar. Elmqvist und Tsigas (2008) formalisieren Verdeckungen wie folgt. Ausgehend von einem Standpunkt S, gilt eine (Sicht-)Linie r von einem Objekt O als blockiert, falls r O schneidet. Ein Objekt V gilt ausgehend vom Standpunkt S als verdeckt, falls keine nicht-blockierte (Sicht-)Linie r existiert. Das Objekt O wird somit als *Verdecker* und das Objekt V als *Verdeckter* bezeichnet.

Werden Teile eines Objektes von anderen Objekten oder von anderen Teilen desselben Objektes verdeckt, können nicht alle Geometrie- und/oder Farbinformationen erfasst werden. Fehlende Geometrieinformationen führen zu Löchern im Modell. Ist die Verdeckungssituation nicht bekannt, so kann dies im Falle der Farberfassung zu Fehlzuweisungen führen, sollte die Farbe des Verdeckers dem verdecken Objekt zugeordnet werden. Im Rahmen einer Verdeckungsanalyse soll zum einen die Verdeckungssituation ermittelt werden, um zum anderen fehlende Geometrieinformation zu ergänzen, sowie Fehlzuweisungen der Farbinformation verhindern.

Die eingangs gegebene Definition von Verdeckern O und Verdeckten V lässt sich auf unterschiedliche Weise auf die Verdeckungsanalyse von Punktwolkendaten übertragen. Intuitiv lassen sich verdeckende und verdeckte Objekte mit realen Objekten (Gebäude, Autos, Schilder, Passanten, ...) assoziieren. Hier wird einfach der Semantik gefolgt, *das Auto verdeckt das Gebäude*. Dies wird im Folgenden als *semantischer Ansatz* bezeichnet. Löst man sich gedanklich von realen Objekten, so können verdeckende und verdeckte Objekte auch durch einzelne Punkte der Punktwolke repräsentiert werden. Die Situation wird also rein geometrisch betrachtet, *Punkt A verdeckt Punkt B*. Dies wird im folgenden als *geometrischer Ansatz* bezeichnet.

Hammoudi u. a. (2012) (erweitert in Hammoudi u. a. (2013)) beschreiben ein Verfahren zu Erzeugung von verdeckungsfreien Fassadentexturen. Die Ausgangsdaten bilden eine Punktwolke und georeferenzierte Fotos, welche im Rahmen einer Mobile Mapping Aufnahme erfasst wurden. Im ersten Schritt wird die Punktwolke segmentiert und anschließend in verdeckende (Straßenmöblierung) und verdeckte (Fassaden) Objekte klassifiziert. Aufgrund dieser Unterscheidung lässt sich dieses Verfahren nach obiger Definition als semantischer Ansatz einordnen. Zur Klassifikation und weiteren Segmentierung der Fassadenobjekte werden dabei existierende Katasterdaten herangezogen. Nach der Identifikation der (potentiell) verdeckten Fassaden und den verdeckenden Vordergrundobjekten werden die Punkte der Verdecker in die erfassten Fotos projiziert. Nach einer Reihe morphologischer Operationen wird eine Verdeckungsmaske für jedes Foto erzeugt. Abbildung 2.8 zeigt ein Foto, welches im Rahmen einer Mobile Mapping Fahrt erfasst wurde (links) und eine nach dem beschriebenen Verfahren erzeugte Verdeckungsmaske (rechts). Abschließend wird für jede identifizierte Fassade eine Textur erzeugt, wobei nur Bildteile verwendet werden, welche nicht über die jeweiligen Verdeckungsmasken ausgeblendet werden.

Bouvier u. a. (2011) befassen sich mit der Identifikation von Verdeckungen in Punktwolken, welche mittels eines terrestrischen Laserscanners erfasst wurden. Als Datenstruktur wird hier ein Winkel-Quadtree genutzt. Dabei werden Punkte nicht anhand ihrer kartesischen Koordinaten, sondern über die Winkel ihrer Polarkoordinaten in den Baum eingefügt. In den jeweiligen Blättern werden die Distanzen der Polarkoordinaten der jeweiligen Punkte gespeichert. Die Punkte eines Blattes lassen sich abschließend über einen Distanzschwellwert in Verdecker und Verdeckte (streng genommen als Nachbar eines verdeckten Bereichs, da, wie zuvor beschrieben, die Geometrieinformation für verdeckte Bereiche fehlt) unterteilen. Da hier einzelne Punkte als verdeckende und verdeckte Objekte herangezogen werden, lässt sich dieses Verfahren als geometrischer Ansatz einordnen.

Abbildung 2.8: Foto aus Mobile Mapping Fahrt (links) und erstellte Verdeckungsmaske (rechts).

2.6 Registrierung mehrerer Datensätze

Wie bereits einleitend beschrieben, sind für die meisten Anwendungen die aufgezeichneten Rohdaten eines Messsystems nicht direkt nutzbar. Ein typisches Verfahren zur Aufbereitung, bzw. Veredelung der Rohdaten ist deren Registrierung. Die Registrierung ist der Prozess um zwei oder mehr Messreihen, bzw. Datensätze, wie beispielsweise zwei Bilder, in ein gemeinsames (Koordinaten-)System zu transformieren Zitova und Flusser (2003). Zu registrierende Datensätze können, im Kontext von Mobile Mapping Daten, sowohl Bilder, als auch Punktwolken sowie eine Kombination aus beiden sein. Sie weisen üblicherweise unterschiedliche Aufnamepositionen, -winkel und -zeitpunkte auf und können darüber hinaus von unterschiedlichen Sensoren erfasst worden sein. Ein Registrierverfahren muss diesen Eigenschaften entsprechend Rechnung tragen.

Einen weit verbreiteten Ansatz zur Registrierung stellen merkmalsbasierte Verfahren dar. Bei diesen Verfahren werden aus allen zu registrierenden Datensätzen zunächst Merkmale extrahiert. Die extrahierten Merkmale werden dann in einem zweiten Schritt auf Übereinstimmungen überprüft. Diese Übereinstimmungen werden auch Korrespondenzen genannt und symbolisieren das selbe Merkmal in mehreren Datensätzen. Wurde eine gewisse Menge an Korrespondenzen identifiziert, so können die Parameter berechnet werden, welche alle Datensätze in ein gemeinsames Koordinatensystem transformieren. Prominente Vertreter für die merkmalsbasierte Registrierung von Bildern sind SIFT von Lowe (2004), SURF von Bay u. a. (2006) und ORB von Rublee u. a. (2011).

Zur Registrierung von Laserscanpunktwolken werden häufig Marker eingesetzt. Diese Marker weisen besondere Reflektionseigenschaften auf und werden vor der Erfassung in der Umgebung verteilt. Aufgrund der besagten Reflektionseigenschaften lassen sich die Marker leicht in den erfassten Scandaten identifizieren und so zur Korrespondenzermittlung heranziehen. Nüchter u. a. (2011) hingegen stellen ein Verfahren zur markerlosen Registrierung von Laserscanpunktwolken vor. Bei diesem Ansatz werden mehrere Messreihen eines terrestrischen Laserscanners registriert. Dafür wird im ersten Schritt ein Tiefenbild als 360-Grad-Panorama aus den Laserscanmessungen bestimmt. Da der Him-

mel den Laserstrahl nicht reflektiert enthält das Tiefenbild ausschließlich die erfassten Objekte, wie die Gebäude. Die Kante zwischen den erfassten Gebäuden und dem nicht erfassten Himmel (engl. Skyline) lässt sich leicht aus dem erzeugten Tiefenbild extrahieren und wird daher als Merkmal für die anschließende Korrespondenzbestimmung genutzt.

2.7 Visualisierungssysteme

Visualisierungssysteme dienen der Darstellung der aufgezeichneten (Mobile Mapping) Daten. Ein Visualisierungssystem bedient sich dabei üblicherweise einer bestimmten Visualisierungstechnik (vgl. Abschnitt 2.2.4), unterstützt gewisse Datenformate (vgl. Abschnitt 2.2.3) und bietet Möglichkeiten zur Interaktion (bspw. Navigation) mit den dargestellten Daten. Je nach Umfang und Ausprägung dieser Eigenschaften adressieren Visualisierungssysteme unterschiedlichste Szenarien, begonnen bei einfacher Datenexploration, über Darstellung von Analyseergebnissen bis hin zur Unterstützung von komplexen Verarbeitungs- und Modellierungsverfahren.

2.7.1 Standalone Point Cloud Viewer

Standalone Point Cloud Viewer sind eigenständige Programme zur Darstellung von Punktwolken. Ein verbreiteter Vertreter dieser Kategorie ist Meshlab (ISTI-CNR (2014)). Meshlab unterstützt diverse Dateiformate, unter anderem auch das für diese Arbeit gewählte Standardformat PLY (vgl. Abschnitt 2.2.3.3).

Meshlab bringt dabei viele nützliche Funktionen zur Bereichselektion, Vereinfachung und weiterführenden Verarbeitung von Punktwolken mit. Leider wird kein LOD Verfahren (vgl. Abschnitt 2.2.5) angewendet, was den Umfang der darstellbaren Daten einschränkt. Dies ist insbesondere bei umfanglichen Mobile Mapping Datensätzen problematisch. Hier besteht lediglich die Möglichkeit einzelne Ausschnitte der gesamten Punktwolke zu laden, bzw. zu visualisieren. Die von Meshlab genutzte PBR Splattingtechnik (vgl. Abschnitt 2.2.4.1) sorgt für eine ansprechende Darstellung, dies jedoch auf Kosten der Bildwiederholrate. Dies verhindert eine flüssige Darstellung von Punktwolken mit mehr als zwei Millionen Punkten, selbst auf aktuellen Systemen. Wie einleitend erwähnt unterstützt Meshlab das PLY Speicherformat, was die Darstellung von Punkt-, sowie texturierter Polygondaten erlaubt. Jedoch kann die Visualisierungstechnik nicht beide Formen gleichzeitig (vgl. Abschnitt 2.2.4.2) darstellen, sondern jeweils nur Punkte, bzw. nur texturierte Polygone. Abbildung 2.9 zeigt exemplarisch einen Screenshot der Meshlaboberfläche.

2.7.2 Webbasierte Systeme

Webbasierte, begehbare Stadtpläne, wie Googles Street View (Vincent (2007)) und Bing Maps Streetside (Microsoft (2016)) basieren auf der Visualisierung von 360-Grad-Panoramabildern. Solche Systeme haben allerdings nur eingeschränkte Darstellungs- und Interaktionsmöglichkeiten. Beispielsweise kann der Standort für die Betrachtung nicht freigewählt werden. Hier ist der Betrachter an die Aufnahmestandorte der Panoramabilder gebunden.

Street Slide, vorgestellt von Kopf u. a. (2010), ist ein System zur kontinuierlichen Visualisierung von Gebäudefassaden. Im Gegensatz zu den erstgenannten Systemen kann der Standort der Betrachtung in gewissen Grenzen (senkrecht zur Fassadenebene) frei gewählt werden. Eine durchgängige Fassadenabbildung wird hier aus Teilen mehrerer Panoramabilder zusammengesetzt. Die einzelnen Teilbilder werden dabei entsprechend des aktuellen Betrachtungsstandpunktes und -winkels reprojiziert, um eine möglichst verzerrungsfreie Ansicht zu erzeugen. Während dies für die entsprechenden Fassaden gute Ergebnisse liefert, werden Vordergrundobjekte, wie Autos oder Straßenmöblierung, sowie Hintergrundobjekte, wie andere Gebäude oder Landschaften, verzerrt, überlagert oder gar mehrfach dargestellt.

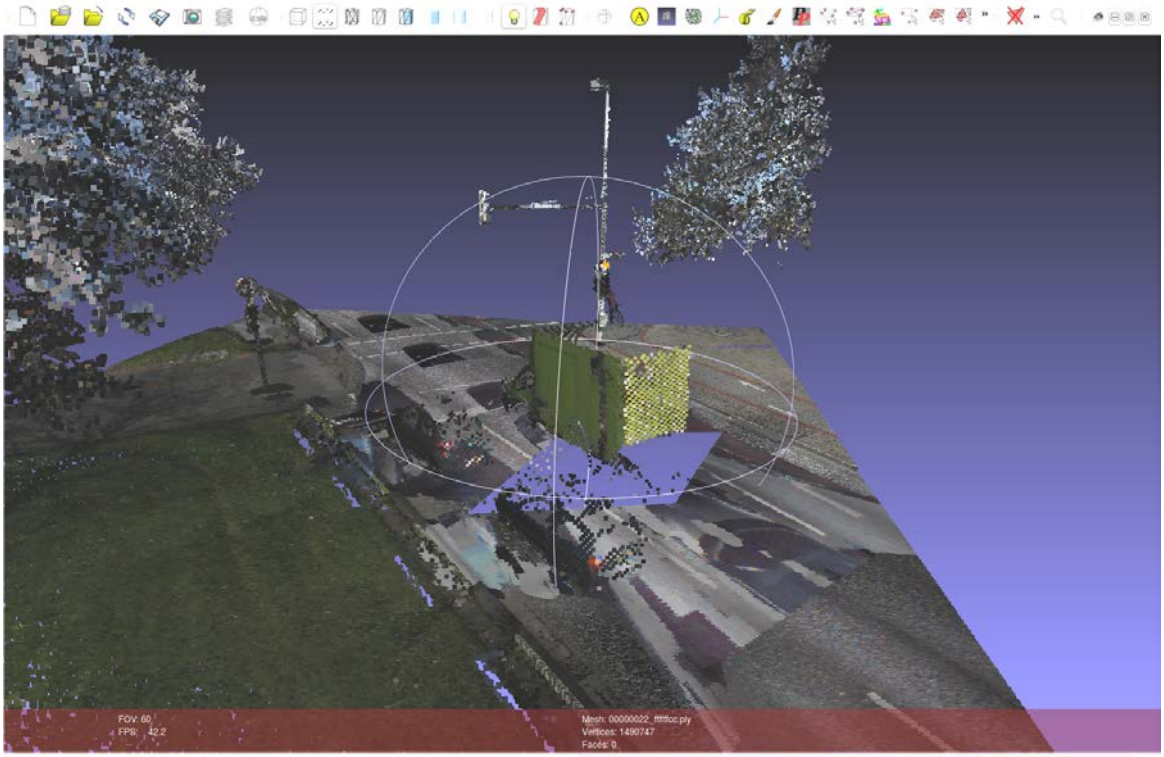


Abbildung 2.9: Exemplarischer Screenshot der Meshlab Anwendungsoberfläche mit gerenderter Punktwolke.

Nguyen u. a. (2016) präsentieren eine technische Lösung zur Visualisierung von 3D Stadtmodellen auf einem webbasierten virtuellen Globus, ähnlich Google Earth. Die Umsetzung basiert auf iTowns, einem quelloffenen Framework zur Darstellung von virtuellen Globen in Webbrowsern. Für die effiziente Darstellung umfangreicher Modelldaten stellen sie eine blickwinkelabhängige LOD Technik namens Chunked-LOD vor. Diese Technik nutzt eine Kachelaufteilung der Daten. Die darzustellenden 3D Stadtmodelle liegen dabei als Dreiecksnetz vor. Die LOD-Datenstruktur basiert nicht auf einzelnen Dreiecken, sondern auf einer Menge von benachbarten Dreiecken (engl. chunk). Dies ermöglicht sowohl eine effektive Übertragung, als auch ein effizientes Rendering der Daten.

3 Effizienzbetrachtungen

Die Verarbeitung von Massendaten stellt hohe Ansprüche an die genutzten Computersysteme. Diesen Anforderungen an Rechenleistung und Speicherbedarf kann auf verschiedenste Weise begegnet werden. Im Wesentlichen lassen sich drei Stellschrauben zur Steigerung der Verarbeitungseffizienz identifizieren:

1. Nutzung effizienter Algorithmen,
2. Parallelisierung des angewendeten Verfahrens und
3. Nutzung geeigneter Datenstrukturen und Zugriffsmechanismen.

Grundsätzlich bestehen zum Teil starke Abhängigkeiten zwischen den genannten Punkten. Das heißt, nicht jeder ausgeklügelte Algorithmus lässt sich gleichzeitig auch parallelisieren und nicht jede Datenstruktur oder Zugriffsstrategie ist mit jedem Algorithmus kompatibel. Eine effiziente Kombination komplexer Datenstrukturen und parallelisierten Verarbeitungsalgorithmen zeigen Eggert und Dalyot (2012) und Eggert und Paelke (2010).

In den folgenden zwei Abschnitten soll das Potential der Parallelisierung (2) und passender Datenstrukturen und (Daten-)Zugriffsstrategien (3) analysiert und für das vorliegende Anwendungsfeld, die Verarbeitung Mobile Mapping Daten, mögliche Lösungsvorschläge entwickelt werden. Sie bilden die Komponenten der Kernmodule *Prozessierungsmodelle* und *Datenstrukturen*. Eine diesbezügliche Einordnung in den Gesamtkontext gibt Abbildung 3.1. Auf die *algorithmische Effizienz* (1) wird bei der Beschreibung der entwickelten Verfahren in den nachfolgenden Kapiteln näher eingegangen.

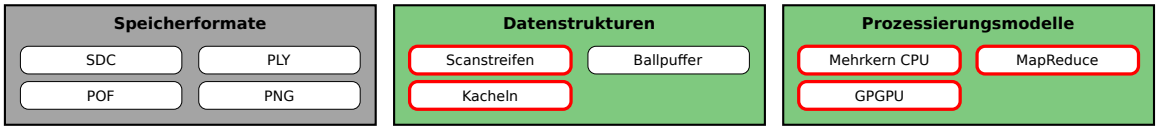


Abbildung 3.1: Einordnung der Module Prozessierungsmodelle und Datenstrukturen als Kernmodule des Frameworks.

3.1 Effiziente Verarbeitung von Massendaten durch Parallelisierung

Ein wichtiges Merkmal bei der Verarbeitung von Massendaten stellt deren Skalierbarkeit dar. Einfach ausgedrückt bedeutet Skalierbarkeit in diesem Kontext: Wie schnell ist die Verarbeitung (noch) wenn (noch) größere Datenmengen verarbeitet werden und wie leicht lässt sich die Verarbeitung weiter beschleunigen. Ersteres stellt eine Eigenschaft des verwendeten Algorithmus dar. Ein Algorithmus mit einer linearen Laufzeitkomplexität ($\mathcal{O}(n)$) benötigt bei doppelter Datenmenge etwa die doppelte Laufzeit. Beschleunigen lässt sich dies durch einen effizienteren Algorithmus, beispielsweise mit einer logarithmischen Laufzeitkomplexität ($\mathcal{O}(log(n))$), oder durch die Parallelisierung des ursprünglichen Verfahrens. Die Entwicklung eines Verfahrens mit geringerer Laufzeitkomplexität ist in den meisten Fällen kein triviales Unterfangen und daher unter Umständen keine reelle Option. Daher konzentriert sich diese Arbeit auf die Analyse von Konzepten zur Parallelisierung der Verarbeitungsverfahren. Neben der potentiell einfacheren Realisierbarkeit hat die Parallelisierung auch einen gewissen Skalierungsvorteil. Durch das bloße Hinzufügen von zusätzlichen Verarbeitungskapazitäten kann die effektive Verarbeitungszeit weiter reduziert werden, ohne neue Verfahren entwickeln zu müssen. In den folgenden Abschnitten werden unterschiedliche Arten der parallelen Verarbeitung von Massendaten beschrieben und bewertet. Zur Bewertung wurde jeweils ein Verfahren zur Ermittlung der Oberflächennormalen der Punkte einer Punktwolke implementiert.

3.1.1 Parallelisierungsformen

Eine Parallelisierung der Verarbeitung kann auf verschiedenste Weise realisiert werden. Die Bandbreite reicht dabei von paralleler Verarbeitung auf einem einzelnen Rechner mit mehreren CPUs über die Verarbeitung auf Grafikarten mittels GPGPU bis hin zur Nutzung ganzer Rechencluster.

3.1.2 Umsetzung

Zu Analysezwecken wurde ein Teil der Datenverarbeitung, die Bestimmung der Punktnormalen, in den drei Varianten *Mehrkern CPU*, *GPGPU* und *Rechencluster* umgesetzt. Einen Vergleich der Laufzeitergebnisse der drei Varianten zieht Abschnitt 3.1.3.

Zur Bestimmung der Punktnormalen werden alle Punkte zunächst in eine Rasterdatenstruktur überführt. Anschließend werden für jeden Punkt mittels der erstellten Rasterstruktur die k-nächsten Nachbarn (kNN) ermittelt. Für die ermittelten Punkte wird eine Hauptachstransformation durchgeführt. Eine Hauptachsentransformation bzw. Hauptkomponentenanalyse (engl. principal component analysis, PCA) nach Pearson (1901) ergibt einen aus den Eigenvektoren der Kovarianzmatrix gebildeten Vektorraum. Vom geometrischen Standpunkt aus kann der gebildete Vektorraum als ausgleichende Ebene betrachtet werden, wobei die Normale der Ebene durch den Eigenvektor mit dem kleinsten Eigenwert repräsentiert wird (vgl. Belton und Lichti (2006) und Gross u. a. (2007)). Die Normale der ausgleichenden Ebene, bzw. der Eigenvektor mit dem kleinsten Eigenwert, wird somit als Punktnormale herangezogen.

3.1.2.1 Mehrkern CPU

Zur Umsetzung des Verfahren zur Bestimmung der Punktnormalen mittels Mehrkern CPUs wurde das *thread pool pattern*, auch *worker-crew model* nach (Garg u. a., 2002) genutzt. Dabei wird die Aufgabe durch mehrere unabhängige Teilaufgaben abgebildet. Die Verarbeitung der Teilaufgaben übernehmen sogenannte Arbeiter (engl. worker). Je nach Grad der Parallelisierung existieren mehrere Arbeiter. Die Verteilung der Aufgaben erfolgt über Warteschlangen (engl. queues).

Wie Abbildung 3.2 zeigt, wurde das Verfahren in vier Arbeitsschritte unterteilt. Die Verteilung der Aufgaben und Daten erfolgt über drei Warteschlangen. Die ersten beiden Arbeitsschritte ermitteln die k nächsten Nachbarn, wohingegen Schritt drei die Punktnormale auf Basis der ermittelten Nachbarn bestimmt und Schritt vier abschließend die Ergebnisse speichert. Bereits jeder dieser vier Schritte lässt sich parallel ausführen, was im Wesentlichen einer Verarbeitungspipeline entspricht. Sind weitere Resourcen (Rechenkerne) verfügbar, so lassen sich Schritt zwei und drei weiter parallelisieren und mehrere Arbeiter in einer Arbeitergruppe zusammenfassen.

Die genutzten Warteschlagen ermöglichen die Rückgabe von vollendeten Aufgaben, womit diese vom Aufgabenersteller wiederverwendet werden können, was den Speicherbedarf senkt, da keine neuen Aufgaben erstellt oder alte vernichtet werden müssen.

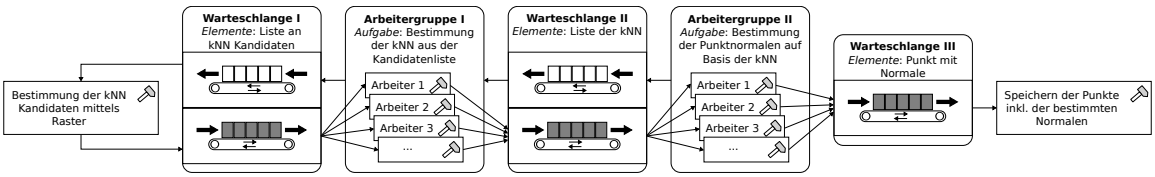


Abbildung 3.2: Übersicht der Mehrkern CPU Implementierung mittels thread pool pattern.

3.1.2.2 GPGPU

Neben Mehrkern CPUs bieten auch moderne Grafikarten eine Schnittstelle zur Nutzung als gewöhnliche Rechenprozessoren, engl. General Purpose Computation on Graphics Processing Unit, kurz GPGPU.

Neben proprietären Frameworks, wie CUDA von Nvidia, stellt OpenCL den offenen und plattformübergreifenden De-facto-Standard dar, welcher auch im Rahmen dieser Arbeit verwendet wurde.

Soll der Grafikprozessor GPU Daten verarbeiten, so müssen selbige zunächst in den Grafikkartenspeicher übertragen werden. Abschließend müssen die Ergebnisdaten vom Grafikspeicher zurück in den Hauptspeicher geschrieben werden. Wie Abbildung 3.3 zeigt, gliedert sich das umgesetzte Verfahren grob in drei GPGPU Schritte.

Kernel1 bestimmt die Rasterzelle, in der sich die jeweiligen Punkte befinden. Dazu müssen zunächst sämtliche Punktkoordinaten in den Speicher der Grafikkarte übertragen werden (1). Das Ergebnis wird dann anschließend zurück zum Host übertragen.

Kernel2 berechnet die Distanzen aller Punkte innerhalb einer Zelle. Dazu werden vom Host zunächst alle Punktindizes zur jeweiligen Zelle übertragen (3). Abschließend werden die berechneten Distanzen zurück zum Host übertragen. Dieser Vorgang wird für alle von *Kernel1* ermittelten Zellen durchgeführt.

Kernel3 berechnet, analog zu *Kernel2*, die Distanzen aller Punkte einer Zelle zu allen Punkten der umgebenden Nachbarzellen. Dazu werden wieder zunächst die jeweiligen Punktindizes in den Grafikspeicher übertragen (5) und abschließend die ermittelten Distanzen vom Grafikspeicher zurück auf den Host.

Im letzten Schritt (7) werden über die ermittelten Distanzen die k nächsten Nachbarn identifiziert und auf Basis dieser die Punktnormalen berechnet. Abschließend wird das Ergebnis gespeichert.

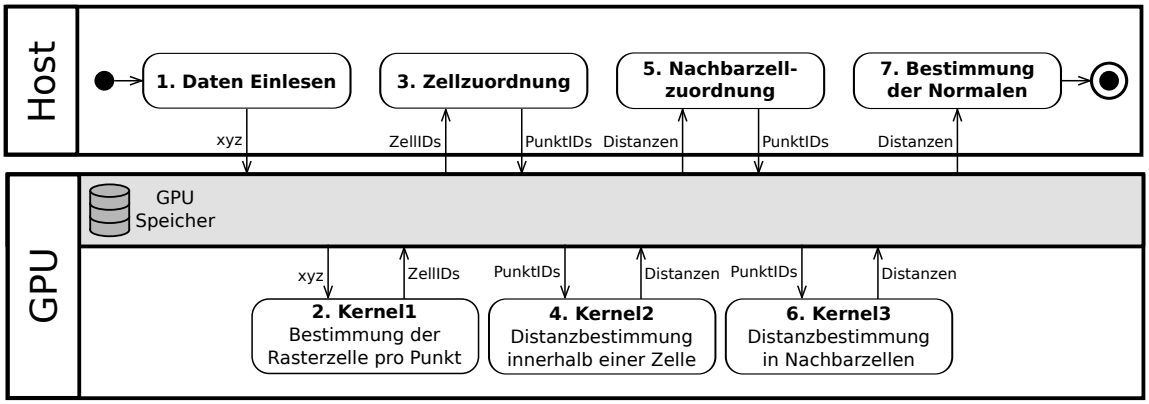


Abbildung 3.3: Übersicht der GPGPU Umsetzung der Bestimmung der Punktnormalen.

3.1.2.3 Rechencluster - Hadoop

Als Beispiel für eine Parallelisierung mittels eines Rechenclusters wurde das Hadoop Framework genutzt. Wie bereits beschrieben ist Hadoop eine Umsetzung des MapReduce Verfahrens. Dabei gilt es das gewünschte Verfahren mittels aufeinander folgender *map* und *reduce* Schritte umzusetzen.

Zur Nutzung des HDFS müssen die zu prozessierenden Daten zunächst in ein unterstütztes Format umgewandelt werden. Das vom HDFS unterstützte Format für Binärdaten ist *Apache Avro*. Vor der Konvertierung der Punktwolkendaten muss zunächst ein Avro-Schema definiert werden. Avro-Schemata werden in JSON definiert und anschließend in Quelltext für die daraus gebildeten Klassen konvertiert. Listing 3.1 zeigt das Avro-Schema der Eingangsdaten. Neben den Koordinaten ist die Zuweisung einer eindeutigen ID pro Punkt notwendig (vgl. MapReduce Phase 3). Das Avro-Schema der Ausgangsdaten zeigt Listing 3.2. Es beinhaltet die Koordinaten und die bestimmte Normale des Punktes. Die für die MapReduce Verarbeitung benötigte ID entfällt.

Für die Verarbeitung der Zwischenergebnisse ist ein weiteres Avro-Schema notwendig, welches im wesentlichen die Rasterzellenstruktur, sowie die Nachbarschaftsrelationen abbildet. Listing 3.3 zeigt

```
{
  "namespace" : "de.hannover.uni.ikg.avro",
  "name"      : "AvroPoint",
  "type"      : "record",
  "fields"    :
  [
    { "name" : "pointId", "type" : "int" },
    { "name" : "x",  "type" : "float" },
    { "name" : "y",  "type" : "float" },
    { "name" : "z",  "type" : "float"}
  ]
}
```

Listing 3.1: Avro-Schema der Eingangsdaten für einen Punkt mit ID und Koordinaten.

```
{
  "namespace": "de.hannover.uni.ikg.avro",
  "type": "record",
  "name": "AvroPointNormal",
  "fields":
  [
    {"name": "x", "type": "float"},
    {"name": "y", "type": "float"},
    {"name": "z", "type": "float"},
    {"name": "nx", "type": "float"},
    {"name": "ny", "type": "float"},
    {"name": "nz", "type": "float"}
  ]
}
```

Listing 3.2: Avro-Schema der Ausgangsdaten für einen Punkt mit Koordinaten und Normale.

das entworfene Avro-Schema, wohingegen Abbildung 3.4 das Klassendiagramm des kompilierten Avro-Schemas veranschaulicht.

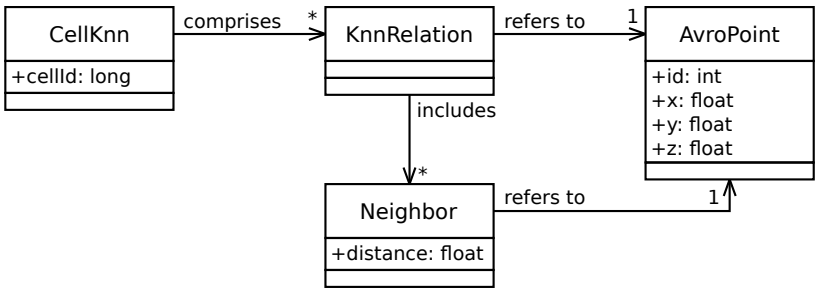


Abbildung 3.4: Klassendiagramm des kompilierten kNN Avro-Schemas aus Listing 3.3.

Wie beschrieben müssen die zu verarbeitenden Daten als erstes in ein HDFS kompatibles Format konvertiert und auf das HDFS übertragen werden. Dies lässt sich in einem Vorverarbeitungsschritt zusammenfassen. Zur Bestimmung der Punktnormalen müssen zunächst die Nachbarpunkte der jeweiligen Punkte ermittelt werden. Dies wird durch drei MapReduce Phasen abgebildet.

Phase 1: Im *map* Schritt der ersten Phase wird für die Eingangsdaten der Form <*AvroPunkt*> die Rasterzelle ermittelt, in der sich der jeweilige Punkt befindet. Jede Rasterzelle erhält basierend auf ihrer Lage im Raster eine eindeutige cellId, welche durch ein *long* Attribut repräsentiert wird und

```
{
   "namespace" : "de.hannover.uni.ikg.avro",
   "type"      : "record",
   "name"      : "CellKnn",
   "fields"       :
   [
      {"name" : "cellId",  "type" : "long"},
      {"name" : "knnList", "type" :
         {"type" : "array", "items" :
            {
               "type"   : "record",
                "name"   : "KnnRelation",
               "fields" :
               [
                  {"name": "point", "type":
                     {
                        "name"   : "AvroPoint",
                        "type"   : "record",
                        "fields" :
                        [
                           {"name" : "pointId", "type" : "int"},
                           {"name" : "x",  "type" : "float"},
                           {"name" : "y",  "type" : "float"},
                           {"name" : "z",  "type" : "float"}
                        ]
                     }
                  },
                  {"name" : "neighbors",
                   "type" :
                     { "type" : "array", "items":
                        {
                           "type"   : "record",
                           "name"   : "Neighbor",
                           "fields" :
                           [
                              {"name" : "point",    "type" : "AvroPoint"},
                              {"name" : "distance", "type" : "float"}
                           ]
                        }
                     }
                  }
               ]
            }
         }
      }
   ]
}
```

Listing 3.3: Avro-Schema der Ausgangsdaten für einen Punkt mit Koordinaten und Normale.

gleichzeitig den notwendigen Schlüssel für die Übergabe an den folgenden *reduce* Schritt darstellt. Die Ausgangsdaten des *map* Schrittes haben somit die Form <*cellId, AvroPunkt*>. Im Kombinationsschritt nach dem *map* Schritt werden dann alle Punkte mit gleichem Schlüssel (hier: cellId) gruppiert und gesammelt an den *reduce* Schritt übergeben. Hier können dann bereits die Nachbarschaftsrela-

tionen aller zur jeweiligen Rasterzelle gehörigen Punkte bestimmt werden. Die Ausgangsdaten haben somit die Form <*cellId, CellKnn*>, welche gleichermaßen die Eingangsdaten der zweiten Phase darstellen.

Phase 2: Der *map* Schritt der zweiten Phase bestimmt für jede Rasterzelle die 26 Nachbarzellen und gibt für jede Nachbarzelle ein entsprechendes Wertepaar der Form <*cellId, CellKnn*> aus, wobei $cellId$ hier der Schlüssel der Nachbarzelle darstellt und $CellKnn$ die Nachbarschaftsrelationen innerhalb der aktuellen Zelle repräsentiert. Im *reduce* Schritt werden nun die Nachbarschaftsrelationen aller Punkte der jeweiligen Nachbarzellen ermittelt. In in der Form <*cellId, CellKnn*> ausgegeben. Hierbei werden die Nachbarschaftsrelationen zur gesamten Zellnachbarschaft gespeichert, welche in der dritten und letzten Phase noch zusammengefasst werden müssen.

Phase 3: Um die Nachbarschaftsrelationen der Punkte in allen untersuchten Rasterzellen zusammenzufassen gibt der *map* Schritt der dritten Phase alle Relationen eines Punktes über die jeweilige PunktId in der Form <*pointId, KnnRelation*> weiter. Im anschließenden *reduce* Schritt alle Nachbarschaftsrelationen auf Basis ihrer jeweiligen Distanzen sortiert und die k nächsten behalten. Abschließend wird anhand dieser k Nachbarpunkte die Normale des Punktes bestimmt und in der Form <*pointId, AvroPointNormal*> gespeichert.

Abschließend müssen die Ergebnisdaten vom HDFS zurückübertragen und wieder in das PLY Format konvertiert werden. Abbildung 3.5 gibt eine Übersicht über die beschriebenen Phasen und Verarbeitungschritte.

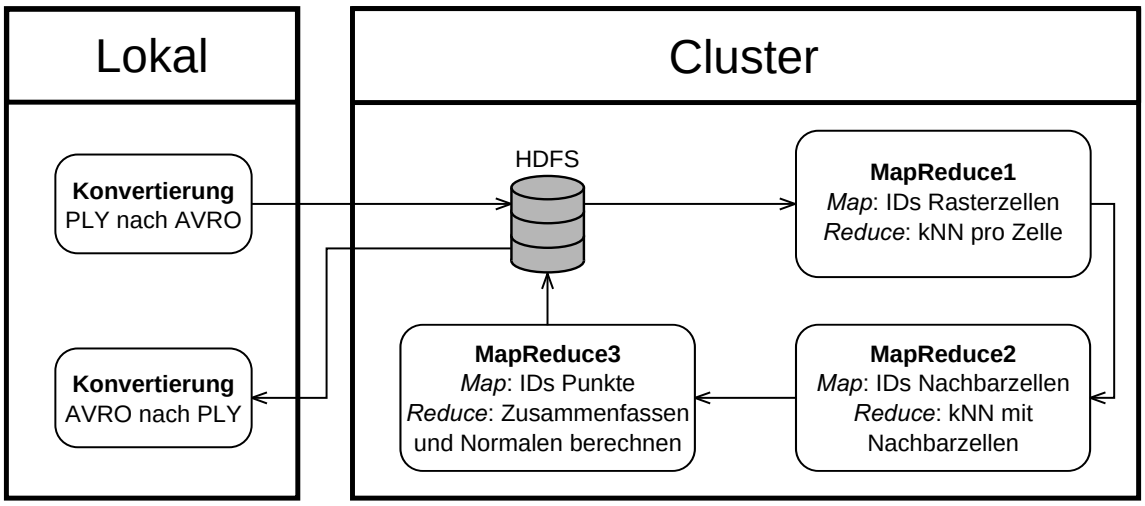


Abbildung 3.5: Übersicht über die Map-Reduce Prozesse bei der Bestimmung der Punktnormalen mittels Hadoop.

3.1.3 Vergleich

Zum Vergleich der umgesetzten Verfahren wurden Punktnormalen für einen Testdatensatz mittels der drei Implementierungen ermittelt. Der Testdatensatz bestand dabei aus 1456 Kacheln, welche insgesamt etwa 316 Millionen Punkte enthalten.

Das Testsystem für die Mehrkern CPU Implementierung bestand aus einem Intel Core i7 870 mit 4 physischen und 8 logischen Kernen via Hyperthreading. Für die GPGPU Variante wurde eine Nvidia GeForce GTX 285 mit 240 Streamprozessoren verteilt auf 10 Shaderclustern verwendet. Das genutzte Hadoop Rechencluster umfasste vier Worker Knoten mit jeweils 4 physischen und 8 logischen Kernen via Hyperthreading.

Das Diagramm aus Abbildung 3.6 zeigt die jeweils benötigte Laufzeit jeder Umsetzung. Dabei schneidet die Mehrkern CPU Variante mit gerade einmal 1h und 8min mit Abstand am besten ab. Die

GPGPU Umsetzung benötigt mit 5h und 10min fast fünfmal solang, wohingegen die Rechencluster Variante mit 13h und 41min schon die dreizehnfache Zeit benötigt.

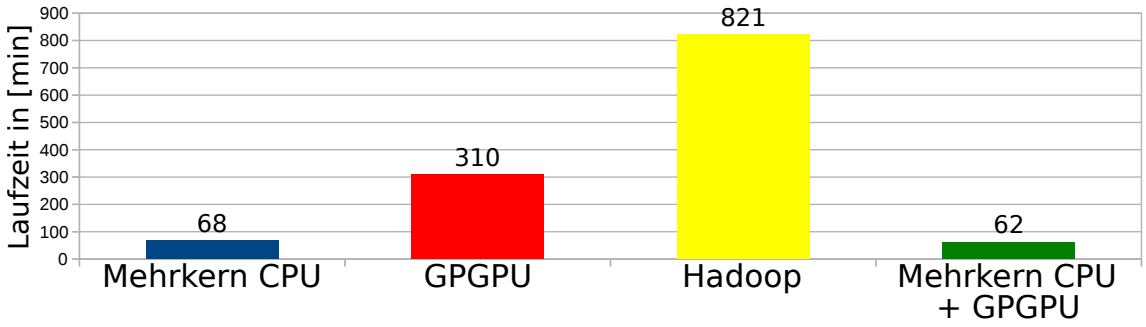


Abbildung 3.6: Diagramm der Laufzeiten der Umgesetzten Verfahren.

Die hohe Laufzeit der Rechencluster Variante kann mehrere Ursachen haben. Das implementierte Verfahren könnte für die Verarbeitung mittels Hadoop ungeeignet sein. Darüber hinaus besteht die Möglichkeit, dass die Implementierung unter Umständen schwächen aufweist und somit keine optimale Laufzeit erzielt werden konnte. Der verhältnismäßig hohe Overhead des Hadoop Frameworks stellt einen weiteren Faktor dar. Hinzu kommt die geringe Anzahl an Rechenknoten und deren Leistungsfähigkeit.

Die Analyse des unerwartet schlechten Abschneidens der GPGPU Implementierung ergab, dass fast 80% der benötigten Laufzeit für die Übertragung der Daten zum Grafikspeicher (33,7%) bzw. vom Grafikspeicher zurück zum Host (46,0%) anfallen. Auf die Kernellaufzeit entfielen dabei 18,7%. Abhilfe könnte hier ein Implementierung schaffen, die auf eine Übertragung der Zwischenergebnisse verzichtet und lediglich das finale Ergebnis vom Grafikspeicher zum Host überträgt.

Ein weiteres Argument für die Mehrkern CPU Implementierung liegt in der Möglichkeit des Einsatzes auf einem entsprechenden Rechencluster, welches kein bestimmtes Framework, wie bspw. Hadoop, voraussetzt. Dabei werden die Kapazitäten des Clusters transparent als Mehrkern CPU dem Verfahren zur Verfügung gestellt.

Abschließend wurde noch eine weitere Variante umgesetzt. Als Basis diente die Mehrkern CPU Implementierung. Als Erweiterung wurde ein GPGPU Arbeiter der Arbeitergruppe I (vgl. Abbildung 3.2) hinzugefügt. Die resultierende Laufzeit betrug 1h und 2min und ist als *Mehrkern CPU + GPGPU* ebenfalls im Laufzeitdiagramm aus Abbildung 3.6 dargestellt. Die Analyse der Anzahl der von jedem Arbeiter verarbeiteten Aufgaben ergab, dass jeder CPU Arbeiter etwa 50 Millionen Aufgaben und der einzelne GPGPU Arbeiter etwa 10 Millionen Aufgaben verarbeitete, welches die nur geringe Laufzeitverbesserung erklärt. Wie schon bei der reinen GPGPU Umsetzung betrug der Laufzeitanteil der Schreib- und Leseoperationen zum und vom Grafikkartenspeicher etwa 80%.

3.2 Effiziente Datenstrukturen

Wie eingangs beschrieben stellen geeignete Datenstrukturen eine weitere Möglichkeit dar, die Laufzeiteffizienz der Datenverarbeitung zu steigern. Die verwendete Datenstruktur muss allerdings immer der Art der Daten, als auch dem Verarbeitungsverfahren gerecht werden. Die Verarbeitung von Mobile Mapping Daten umfasst zwei häufig genutzte Verarbeitungstypen, zum einen eine sequentielle Verarbeitung der Daten in Erfassungsreihenfolge und zum anderen die gemeinsame Verarbeitung von räumlich benachbarten Punkten. Beide Verfahrensmuster haben ihre Schnittmengen, also Daten die zeitlich nahe erfasst wurden, weisen üblicherweise auch eine gewisse räumliche Nähe auf, jedoch gilt dies umgekehrt nicht.

Aus diesem Grund wurden zwei Datenstrukturen entworfen, um beide Verarbeitungsmuster optimal zu unterstützen: *Scanstreifen mit einer Pufferstrategie* zum wiederholten Zugriff auf bereits

geladenen Daten zur Unterstützung der Datenverarbeitung in Erfassungsreihenfolge und eine *Rasterdatenstruktur mit einer Cachingstrategie* zur Unterstützung eines wahlfreien Zugriffs auf räumlich benachbarte Punkte.

3.2.1 Scanstreifen

Ein von Riegl spezifiziertes binäres Ausgabeformat der Laserscan Daten ist *Scan Data Computed*, oder kurz SDC. Neben einem sehr kurzen Header, welcher lediglich die Versions-Nummer beinhaltet, speichert das Format sämtliche Laserscanmessungen als sogenannten SDCRecord, welcher alle relevanten Informationen zur Messung, wie bspw. Zeitstempel, Distanz und Winkel enthält. Für jedes Messintervall wird eine neue SDC Datei erzeugt, welche alle im Intervall erzeugten Messwerte beinhaltet. Da während einer Mobile Mapping Messfahrt die Messung gestoppt wird, sobald das Messfahrzeug anhält, besteht eine komplette Messfahrt entsprechend aus mehreren Messintervallen mit den jeweiligen SDC Dateien. Zur Unterscheidung erhält jedes Messintervall eine eigene Identifikationsnummer, häufig als „run id“bezeichnet.

Zur leichteren Verarbeitung der einzelnen Distanzmessungen werden diese in Scanstreifen zusammengefasst. Ein Scanstreifen besteht dabei aus allen Distanzmessungen einer Umdrehung des Scannerkopfes.

3.2.2 Scanstreifenbasierte Pufferstrategie

Auf Grund der großen Datenmenge bei Mobile Mapping Daten ist es in der Regel nicht möglich alle Messdaten zur selben Zeit im Speicher zu halten. Daher liest man für gewöhnlich die Daten sequenziell ein und verarbeitet jedes Datum (z.B. ein SDCRecord oder ein Scanstreifen) unmittelbar nach dem Einlesen ohne es zu speichern. Diese Vorgehensweise ist jedoch nicht für alle Anwendungen praktikabel, insbesondere dann, wenn die jeweiligen Daten im späteren Prozess wiederholt benötigt werden, was ein erneutes Einlesen erforderlich macht. Ein Beispiel für einen solchen Prozess ist die Ermittlung der RGB Farbwerte aus den während der Messfahrt aufgenommenen Fotos. Dabei müssen alle Laserscanmessungen in das Bildkoordinatensystem transformiert werden, um den dazugehörigen Farbwert für die jeweilige Messung zu ermitteln. Da im Vorfeld nicht bekannt ist, welche Laserscanmessung in welches Foto transformiert werden kann, müssen potentiell alle Messungen in alle Fotos transformiert werden. Diese Vorgehensweise ist laufzeittechnisch jedoch nicht praktikabel. Unter der Annahme, dass zeitlich aufeinander geschossene Fotos auch räumlich eine gewisse Nähe aufweisen, ist davon auszugehen, dass ein Teil der Laserscanmessungen, welche erfolgreich in ein Foto transformiert werden konnten auch in das anschließend gemachte Foto erfolgreich transformiert werden können. Basierend auf dieser räumlichen und zeitlichen Nähe wurde im Rahmen dieser Arbeit eine Pufferstrategie entwickelt, welche das erneute Einlesen der Scandaten verhindert und dennoch minimale Speicheranforderungen hat.

Wie einleitend beschrieben geht die vorgestellte Pufferstrategie von einem wiederholten Zugriff auf bereits eingelesene Daten aus. Um festzulegen welche Daten im Puffer vorgehalten werden müssen, werden wie in Abbildung 3.7 gezeigt, vier Pufferfenster festgelegt, sowie ein *relevanter Bereich* definiert. Der relevante Bereich repräsentiert dabei den Datenumfang, welcher für den letzten Verarbeitungsschritt genutzt wurde, also *relevant* war.

Abbildung 3.7: Puffer mit relevantem Bereich und definierten Fensterbereichen.

Gemäß der anfänglichen Prämisse überlappt der relevante Bereich des nächsten Verarbeitungsschrittes den vorangegangenen, wie in Abbildung 3.8 dargestellt. Am Beispiel der Farbwertbestimmung umfasst der relevante Bereich alle Messwerte bzw. Scanstreifen welche erfolgreich in das letzte Foto transformiert werden konnten.

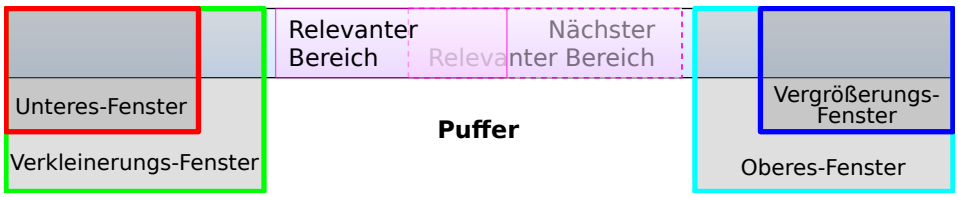


Abbildung 3.8: Puffer mit Überlappung aufeinander folgender relevanter Bereiche.

Da zur Sicherheit immer alle gepufferten Scanstreifen verarbeitet werden, ist es von Vorteil, wenn sich möglichst wenige im Puffer befinden. Jedoch unterscheidet sich die Anzahl an Scanstreifen, die beispielsweise in ein Bild projiziert werden können, erheblich von Bild zu Bild, in Abhängigkeit von der jeweiligen Erfassungssituation. In Bildern aus Kurvenfahrten ist die Anzahl zum Teil mehr als doppelt so groß, gegenüber Bildern aus Geradeausfahrten. Prinzipiell muss also die Puffergröße der maximalen Anzahl an Scanstreifen über alle Bilder entsprechen. Dies ist zum einen schwer abzuschätzen und widerspricht zum anderen dem anfänglich beschriebenen Streben nach einem möglichst kleinen Puffer. Die Lösung des Problems liegt hier in der Verwendung eines Puffers mit variabler Größe. Die eingangs erwähnten Pufferfenster regulieren wie nachfolgend beschrieben die aktuelle Puffergröße.

Die vier festgelegten Pufferfenster regeln dabei, ob und welche der fünf möglichen Pufferoperationen *Verschieben*, *Verschieben und Nachprozessieren*, *Vergrößern*, *Verkleiner*, *Verkleiner und Verschieben* ausgeführt wird. Die Überlappungen des relevanten Bereiches mit den vier Pufferfenstern werden für diese Entscheidung herangezogen. Tabelle 3.1 gibt eine Übersicht über alle Möglichen Überlappungskombinationen und den daraus abgeleiteten Pufferoperationen. *U* steht dabei für das Untere-Fenster, *VK* für das Verkleinerungs-Fenster, *O* für das Obere-Fenster und *VG* für das Vergrößerungs-Fenster. Bei vier Fenstern ergeben sich grundsätzlich 16 Überlappungskombinationen mit dem relevanten Bereich, da jedoch *U* in *VK* und *VG* in *O* enthalten ist, reduziert sich die Anzahl der Kombinationen auf neun.

Überlappung mit				Pufferoperation
U	VK	O	VG	
X	X	-	-	Keine Operation
X	X	X	-	Keine Operation
-	X	-	-	Keine Operation
X	X	X	X	*Vergrößern*
-	-	-	-	*Verkleinern*
-	X	X	-	*Verschieben*
-	-	X	-	*Verkleinern und Verschieben*
-	-	X	X	*Verschieben und Nachprozessieren*
-	X	X	X	*Verschieben und Nachprozessieren*
X	-	-	-	nicht Möglich
X	-	X	-	nicht Möglich
X	-	X	X	nicht Möglich
X	-	-	X	nicht Möglich
-	-	-	X	nicht Möglich
-	X	-	X	nicht Möglich
X	X	-	X	nicht Möglich

Tabelle 3.1: Überlappungskombinationen des Relevanten Bereichs mit den definierten Pufferfenstern (U=Unteres-Fenster, VK=Verkleinerungs-Fenster, O=Oberes-Fenster und VG=Vergrößerungs-Fenster) und die daraus abgeleiteten Pufferoperationen.

Abbildung 3.9 veranschaulicht die Überlappungssituationen für die Operationen *Vergrößern*, *Verkleinern* und *Verschieben*. Ein vergrößern der Puffers wird notwendig wenn sich der relevante Bereich über nahezu den gesamten Puffer erstreckt. Im Anschluss an eine Puffervergrößerung muss überprüft werden, ob die neu hinzugekommenen Scanstreifen ebenfalls zum relevanten Bereich gehören. Ist dies der Fall, so muss der Puffer abermals vergrößert werden. Der Puffer wird daher so lange vergrößert, bis keine der im letzten Vergrößerungsschritt hinzugekommenen Scanstreifen mehr zum relevanten Bereich gehört. Eine Verkleinerung des Puffers kann erfolgen wenn der relevante Bereich keine Überlappungen mit den Pufferfenstern aufweist. Erstreckt sich der relevante Bereich über die beiden inneren Fenster *VK* und *O* so kann der Puffer verschoben werden. Beim Verkleinern, als auch beim Verschieben des Puffers wird der linke Rand des Puffers (also die ältesten Scanstreifen) entfernt und etwaig gesammelte Ergebnisse (wie bspw. die Punktfarben) gespeichert. Im Falle des Verschiebens werden anschließend noch neue Daten an den rechten Rand des Puffers angefügt.

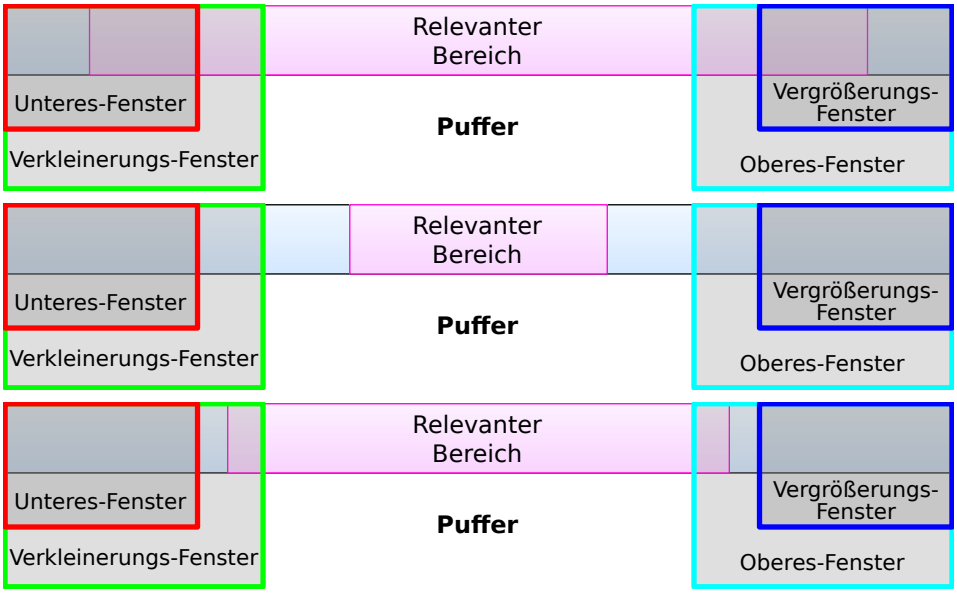


Abbildung 3.9: Übersicht über Überlappungskombinationen für Vergrößern *(oben),* Verkleinern *(Mitte) und* Verschieben *(unten).*

Überlappt der relevante Bereich lediglich mit dem Oberen-Fenster, so kann der Puffern nicht nur Verschoben, sondern auch verkleinert werden, da ein großer Bereich am linken Rand des Puffers nicht mehr relevant ist. Abbildung 3.10 veranschaulicht diese Situation.

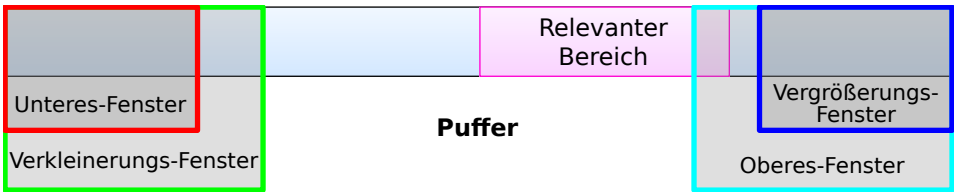


Abbildung 3.10: Überlappungskombination von kombinierter Pufferoperation Verkleinern und Verschieben.

Liegt der relevante Bereich weit am rechten Pufferrand, also im *VG* Fenster, jedoch weit genug vom linken Rand entfernt, also nicht im unteren Fenster *U*, so wird der Puffer verschoben. Da jedoch eine Überlappung im *VG* Fenster vorliegt, wird analog zur Puffervergrößerung überprüft, ob die neu zum Puffer hinzugefügten Daten noch zum relevanten Bereich gehören (Nachprozessieren) und unter Umständen sogar zusätzliche Puffervergrößerungen notwendig sind. Abbildung 3.11 zeigt beide beschriebenen Überlappungssituationen.

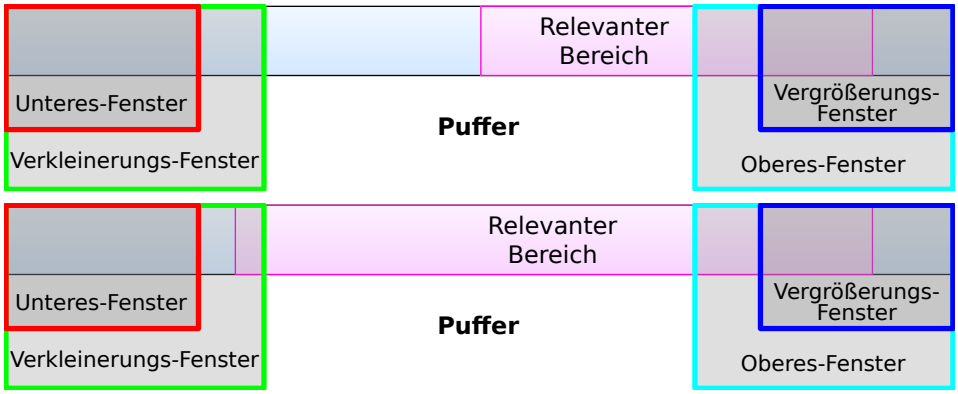


Abbildung 3.11: Überlappungskombinationen von kombinierter Pufferoperation Verschieben und Nachprozessieren.

Um ein ständiges Ändern der Puffergröße zu verhindern wird noch eine Mindestpuffergröße, welche nicht unterschritten werden darf eingeführt. Darüber hinaus müssen die Fenstergrößen U, VK, O und VG sinnvoll gesetzt werden. Bei der Wahl der Größe sollten die zu erwartenden Unterschiede in den relevanten Bereichen aufeinanderfolgender Prozessierungsschritte berücksichtigt werden. Schwanken diese stark, oder Überlappen sich deren relevanten Bereiche nicht so wie in Abbildung 3.8 gezeigt, sollten zumindest die beiden unteren Fenster U und VK prinzipiell breiter gewählt werden, um zu verhindern, dass Daten aus dem Puffer entfernt werden, welche in den nächsten Prozessierungsschritten noch benötigt werden könnten.

Die gezeigte Pufferstrategie garantiert bei einer geeigneten Wahl der Fenstergrößen eine vollständige und korrekte Prozessierung aller Daten. Darüber hinaus werden sämtliche Scanstreifen genau einmal geladen, was gerade bei langsameren Datenträgern, bspw. Netzlaufwerke, einen entscheidenden Laufzeitvorteil bringt. Durch die automatische Anpassung der Puffergröße an die relevanten Bereiche wird eine weitere Laufzeitverbesserung erzielt, da nicht mehr Daten pro Prozessierungsschritt verarbeitet werden müssen, als unbedingt notwendig. Dies erhöht darüber hinaus auch die Speichereffizienz des Verfahrens.

Abbildung 3.12 zeigt beispielhaft die zeitliche Entwicklung der Puffergröße (cyan) bei der Anwendung der beschriebenen Strategie zur Farbwertermittlung der Scanpunkte. Als Mindestpuffergröße wurden 800 Scanstreifen gewählt, für die Fenster U und O wurde eine Größe von jeweils 200 Scanstreifen genutzt, wohingegen VK auf $2 * U$ also 400 und VG $\frac{1}{2}O$ also 100 betrugen. Neben der Puffergröße ist ebenfalls die Größe des relevanten Bereichs (violett) der Prozessierten Bilder abgetragen. Deutlich zu erkennen ist, dass die Puffergröße jeder Zeit in etwa der Größe des relevanten Bereichs entspricht und die Verarbeitung somit Laufzeit- und Speichereffizient ablief. Die Laufzeitersparnis bei der Nutzung der automatischen Puffergrößenanpassung gegenüber einer statischen Puffergröße lässt sich wie folgt abschätzen. Bei der Nutzung einer statischen Puffergröße beträgt diese immer konstant die maximale Größe bei der Verarbeitung jedes einzelnen Bildes. Im Falle der in Abbildung 3.12 gezeigten Verarbeitung von 1995 Bildern betrug diese 9100. Das ergibt eine Gesamtanzahl an prozessierten Scanstreifen von $1995 * 9100 = 18.154.500$. Die Summe der prozessierten Scanstreifen bei automatischer Puffergrößenanpassung betrug im gleichen Datensatz lediglich 5.568.450. Geht man von einer Konstanten Verarbeitungszeit pro Scanstreifen pro Bild aus, so reduziert sich die Laufzeit auf etwa 30%.

Gut zu erkennen sind die immer wieder auftretenden starken Puffervergrößerungen. Die Ursache liegt, wie bereits eingangs erwähnt, in Kurvenfahrten, bei denen pro Bild mehr Scanstreifen relevant werden. Die Darstellung der Puffergröße in Abhängigkeit von der Position des jeweils verarbeiteten Bildes zeigt Abbildung 3.13 am Beispiel des Datensatzes aus Abbildung 3.12 und eines weiteren Datensatzes.

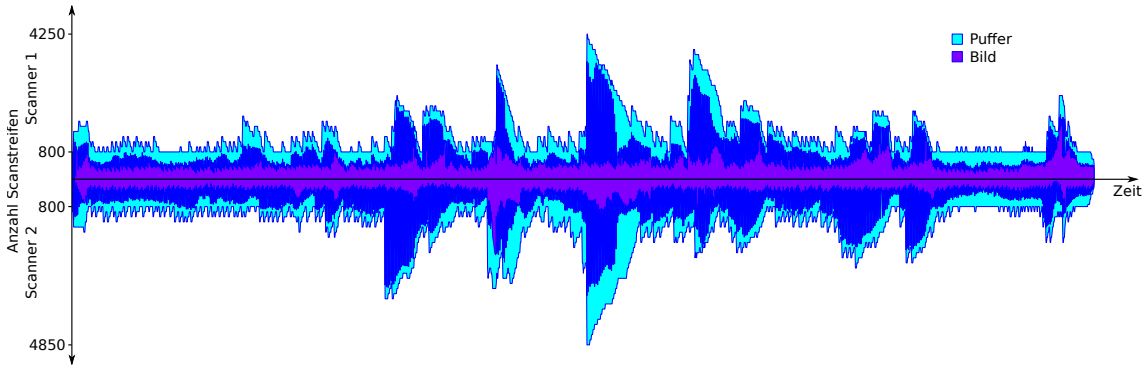


Abbildung 3.12: Zeitliche Entwicklung der Puffergröße, sowie Anzahl an Scanstreifen pro Bild.

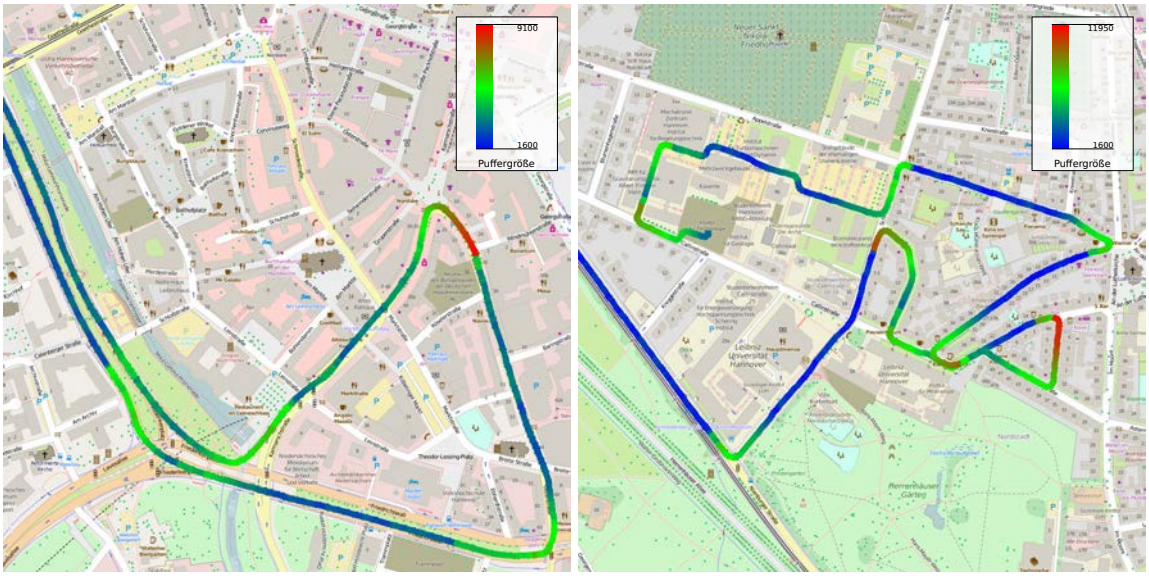


Abbildung 3.13: Farbcodierte Puffergröße, hohe Puffergröße (rot) jeweils bei Kurvenfahrten.

3.2.3 Rasterdatenstruktur

Wie die zuvor beschriebene Strategie zur Verarbeitung der Scanstreifen zeigt, ist die sequentielle Verarbeitung der Daten nur bedingt flexibel und daher unter Umständen nicht für alle Probleme bzw. Verarbeitungsschritte nutzbar. Sollen beispielsweise Nachbarschaften von Scanpunkten analysiert werden, so ist dies mit einer (zeit-)sequentiellen Verarbeitungsstrategie nicht effizient möglich, da räumlich benachbarte Scanpunkte möglicherweise zu unterschiedlichen Zeitpunkten erfasst wurden. Für eine effiziente Verarbeitung von derartigen Problemstellungen ist daher die Verwendung einer räumlichen Datenstruktur notwendig.

Neben diversen baumbasierten Strukturen eignet sich eine rasterbasierte Struktur für den Zugriff auf die Mobile Mapping Daten. Gegenüber baumbasierten Strukturen hat letztere den Vorteil, dass jede erzeugte Rasterzelle (Kachel) als eigenständige Datei persistent gespeichert werden kann. Bei der Verwendung eines ebenen Koordinatensystems wie UTM oder Gauß-Krüger ist die Z Komponente (Höhe) der Scanpunkte durch die maximale Höhe der erfassten Objekte (Gebäude, Bäume, usw.) limitiert, was einen sehr begrenzten Wertebereich ergibt. Dieser geringe Wertebereich macht eine zusätzliche Unterteilung der in Richtung der Z-Achse unnötig, daher kann sich hier auf ein lediglich zweidimensionales Raster beschränkt werden.

Zur Unterstützung von Verfahren, welche einen wahlfreien Zugriff auf räumlich benachbarte Punkte voraussetzen, werden die erfassten Scanpunkte in ein zweidimensionales Raster überführt. Dafür

müssen die Ausgangsdaten zunächst in ein ebenes Koordinatensystem transformiert werden. Als Referenzsystem wurde ETRS89 mit UTM Abbildung verwendet. Schließlich muss noch die Ausdehnung einer Kachel des Rasters festgelegt werden. Diese hängt im Wesentlichen von der finalen Verwendung und den Speicherkapazitäten ab. Für die im Rahmen der Verarbeitungskette durchzuführenden Operationen wurde eine Kachelgröße von 25 Metern, als guter Kompromiss zwischen Anzahl an Kacheln und Anzahl an Punkten pro Kachel, festgelegt. Dabei werden für eine typische Messfahrt Kacheln mit bis zu drei Millionen Punkten pro Kachel generiert. Abbildung 3.14 zeigt den Ausschnitt eines Rasters für ein typisches Beispielprojekt. Die Trajektorie des Messfahrzeugs ist in Rot dargestellt.

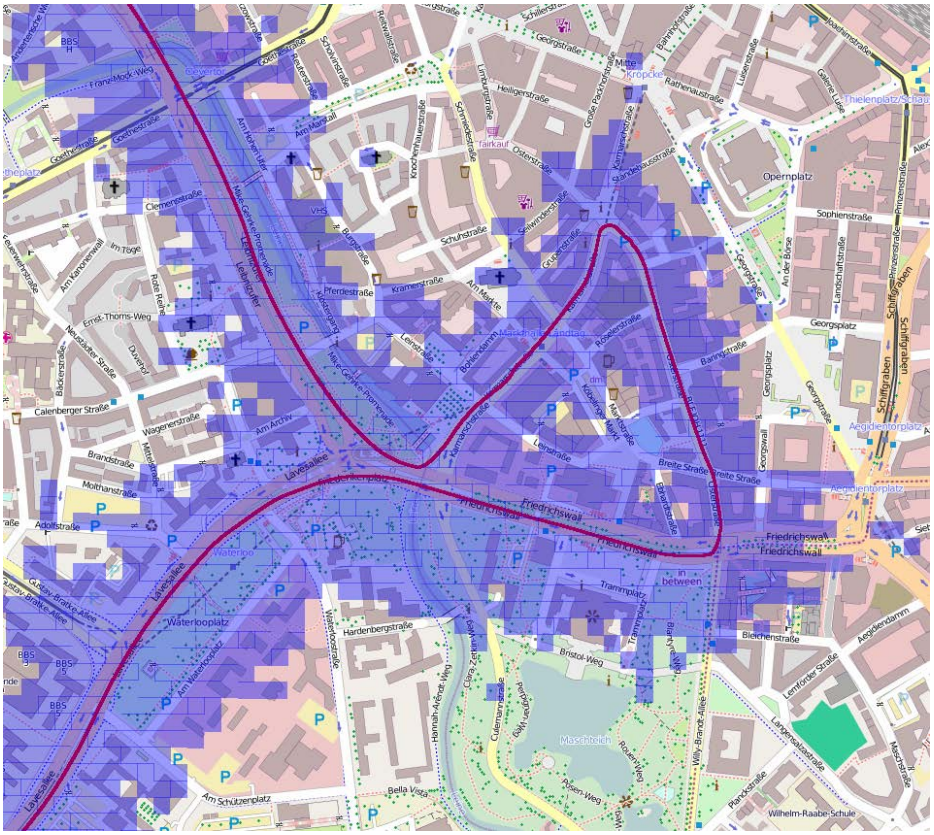


Abbildung 3.14: Raster von Punktwolkenkacheln mit 25 Metern Kantenlänge (transp. blau), inkl. der Trajektorie des Messfahrzeugs (rot).

Pro Kachel werden alle enthaltenen Punkte relativ zum Kachelursprung gespeichert. Zusammen mit der Speicherung des jeweiligen Kachelursprungs im Referenzkoordinatensystems können so die globalen Koordinaten jedes Scanpunktes mit geringerem Speicheraufwand pro Punkt rekonstruiert werden. Liegt die benötigte Auflösung in X und Y Richtung beispielsweise im Millimeterbereich, so genügt für die Speicherung der X und Y Komponenten eine Speicherbreite nach Gleichung 3.1 von zwei Byte pro Komponente, anstelle der sonst notwendigen acht Byte bei Speicherung mit doppelter Genauigkeit im globalen Referenzsystem. Die Z Komponente als Höhe über dem Referenzellipsoid ist jedoch nicht auf die Kachelgröße beschränkt und muss daher mit einer höheren Speicherbreite bspw. mit vier Byte bei einfacher Genauigkeit, gespeichert werden.

$$\begin{aligned} \text{Kantenlänge: } & 25m = 25.000mm \\ \text{Speicherbreite in Bit: } & \log_2(25.000) = 14,61Bit \\ \text{Speicherbreite in Byte: } & \left\lceil \frac{14,61}{8} \right\rceil = 2Byte \ (\hat{=}\textbf{short}) \end{aligned} \tag{3.1}$$

Der effiziente Zugriff auf die Kacheln und bspw. deren Nachbarn wird durch einen zweidimensionalen Rasterindex sichergestellt. Mit diesem wird für jede beliebige Kachel eine Zugriffszeit von $O(1)$ gewährleistet.

3.2.4 Randproblematik und Caching

Neben den vielen Vorteilen, die eine Kacheldatenstruktur bietet, gilt es auch die damit einhergehenden Probleme zu adressieren. Basieren alle Verarbeitungsschritte auf einzelnen Kacheln, so können unter Umständen Probleme bei Punkten auftreten, welche sich am Rand der Kachel befinden. Dies wird als Randproblematik bezeichnet und tritt in verschiedenster Form in vielen Verarbeitungsschritten auf. Ein simples Beispiel ist die Bestimmung der k nächsten Nachbarn (kNN) eines Punktes. Für Punkte nahe des Randes befindet sich ein Teil der nächsten Nachbarn unter Umständen am Rand der Nachbarkachel.

Der gängigste Lösungsansatz liegt darin alle Punkte innerhalb eines Puffers um die zu verarbeitende Kachel für die Verarbeitung mit heranzuziehen. Abbildung 3.15 visualisiert die Situation bei unterschiedlichen Puffergrößen. Je nach Größe der Kacheln und dem jeweiligen Anwendungsfall genügt eine Puffergröße von maximal der Kachelgröße. Das heißt es werden höchstens alle Punkte aller Kacheln der direkten Achternachbarschaft (Moore Neighborhood) berücksichtigt.

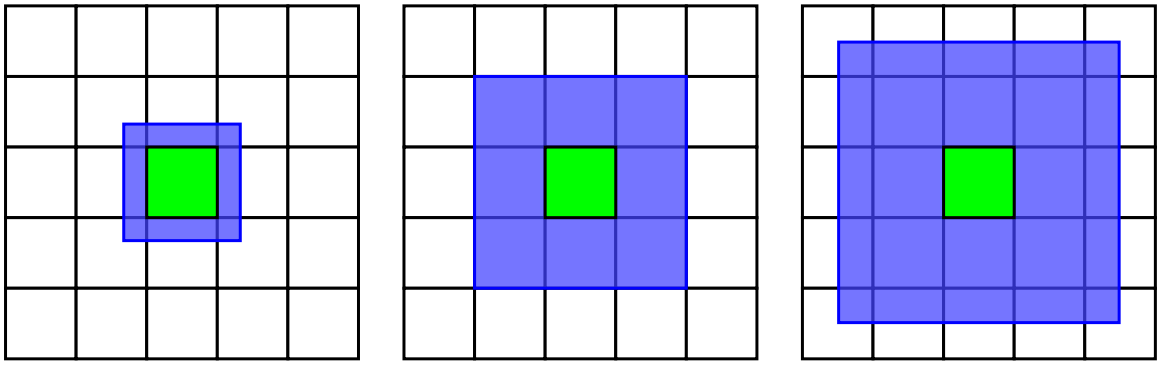


Abbildung 3.15: Lösungsansatz zur Randproblematik: Puffer (blau) um zu verarbeitende Kachel (grün) mit den Puffergrößen $Puffer < Kachel$ (links), $Puffer = Kachel$ (mitte) und $Puffer > Kachel$ (rechts).

Je nach Anwendungsfall stellt das Laden der Kacheldaten einen signifikanten Teil der Programmlaufzeit dar. Werden ohnehin alle Kacheln prozessiert, so bietet sich ein Zwischenspeichern (engl. Caching) der bisher geladenen Kacheln an. Ohne Caching würde jede Kachel genau neun mal in den Hauptspeicher geladen. Einmal bei der Verarbeitung der Kachel selbst und acht mal bei der Verarbeitung der jeweiligen Nachbarkacheln.

Für eine maximale Effizienz des Caches spielen im Wesentlichen zwei Faktoren eine Rolle, zum einen die Größe des Caches und zum anderen die verwendete Cachingstrategie. Dabei ist grundsätzlich anzumerken, dass je größer der Cache, desto einfacher die Strategie. Hat der Cache bspw. eine unbegrenzte Kapazität, so ist die Cachingstrategie unerheblich, da einmal geladene Daten nie aus dem Cache entfernt werden. Je kleiner die Kapazität des Caches desto wichtiger ist die verwendete Strategie, da bspw. bei einem komplett gefüllten Cache entschieden werden muss, welche Daten im Falle des Eintreffens neuer Daten verworfen werden dürfen. Dies wird als Verdrängungsstrategie bezeichnet.

Grundsätzlich kann zwischen drei Verdrängungsstrategien unterschieden werden, FIFO, LFU und LRU. FIFO steht für *first in first out* und entfernt jeweils das älteste Element. LFU steht für *least frequently used* und entfernt jeweils das am seltensten genutzte Element. LRU steht hingegen für *least recently used* und entfernt das Element auf das am längsten kein Zugriff erfolgte. Ein LFU-Cache ist für die Verarbeitung von Kacheln (wie sie im Rahmen dieser Arbeit stattfindet) eher ungeeignet, da sich die Anzahl der Zugriffe pro Kachel auf maximal neun beläuft und anschließend entfernt werden

kann, da kein weiterer Zugriff erfolgen wird. Ein LFU-Cache würde dahingegen jedoch versuchen diesen Eintrag maximal lang bereitzustellen, da maximal häufig darauf zugegriffen wurde. Bei FIFO- und LRU-Caches werden Elemente aufgrund ihres Alters entfernt, wobei bei einem FIFO-Cache das Alter anhand des Ladezeitpunktes und bei einem LRU-Cache anhand des letzten Zugriffszeitpunktes bestimmt wird. Da die Zugriffszeitpunkte in der Regel relevanter sind als die Ladezeitpunkte sind LRU Caches im allgemeinen flexibler einsetzbar und gehören damit zu den meistgenutzten Strategien, beispielsweise auch bei Prozessorencaches. Aufgrund der hohen Flexibilität wird bei der Verarbeitung von gekachelten Datensätzen auf einen LRU-Cache zurückgegriffen.

Die Effizienz eines Caches kann über die Cache-Miss bzw. Cache-Hit Rate beurteilt werden. Dabei bezeichnet man die Situation bei der ein Element abgefragt und im Cache vorhanden ist als Cache-Hit (Cachetreffer) und das Nichtvorhandensein des Elements als Cache-Miss (Cacheverfehlung). Ziel ist dabei immer eine hohe Cache-Hit bzw. niedrige Cache-Miss Rate. Da beide Werte im direkten Zusammenhang stehen ($Rate_{Hit} = 1 - Rate_{Miss}$) genügt es sich auf einen der beiden Werte zu beschränken, in der folgenden Bewertung wird dazu die Cache-Miss Rate herangezogen.

Die Werte aus Abbildung 3.16 wurden bei der Verarbeitung eines Datensatzes bestehend aus 3314 Kacheln bei einer Cachegröße von 12 ermittelt. Zu jeder Kachel wurden dabei alle acht Nachbarkacheln geladen. Wie erwähnt wird auf jede Kachel neun mal zugegriffen, jedoch besitzen Kacheln am Rand des Datensatzes keine vollständige Nachbarschaft und somit weniger als acht Nachbarn. Die Gesamtanzahl an Kachelzugriffen betrug daher lediglich 26.292 anstatt $9 * 3314 = 29.826$. $Rate_{Miss}$ ergibt sich dabei aus dem Verhältnis von Anzahl der Cache-Miss und Anzahl der Gesamtzugriffe. Die maximale $Rate_{Miss} = 1.0$ ergibt sich bei einer Cachegröße von 1, was einem Fehlen eines Caches gleichzusetzen ist. Dabei müssen immer alle Kacheln bei jedem Zugriff neu geladen werden. Da jede Kachel mindestens ein mal geladen werden muss ergibt sich eine minimale Rate von $Rate_{Miss} = \frac{3314}{26292} = 0,13$. Da das einmalige, bzw. erstmalige, Laden von Daten in den Cache kein Cache Miss im eigentlichen Sinne darstellt, wird in Abbildung 3.16 zusätzlich $Rate^*_{Miss} = \frac{Anzahl_{Miss} - 3314}{26292}$ aufgeführt, welche eine minimale Rate von 0.0 als Optimum ermöglicht.

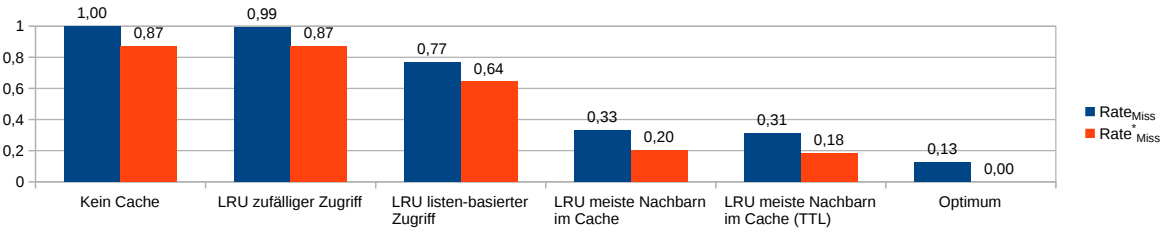


Abbildung 3.16: Übersicht über Cache-Miss Raten bei Verwendung unterschiedlicher Zugriffsstrategien.

Wie die Cache-Miss Rate von 99% bei der Verwendung eines LRU Caches mit zufälligem Zugriff zeigt, wird für einen effizienten Cache nicht nur eine gute Verdrängungsstrategie, sondern auch eine passende Zugriffsstrategie benötigt. Dies ist im Wesentlichen der geringen Cachegröße geschuldet. Bei mehreren Millionen Punkten pro Kachel ist ein großer Cache jedoch sehr speicherintensiv und somit nicht realisierbar. Wählt man keine zufällige Zugriffsreihenfolge, sondern beispielsweise die Reihenfolge in der die Kacheln generiert wurden, verbessert sich die Cache-Miss Rate bereits auf 77%, was aber nach wie vor weit vom Optimum entfernt ist. Ist der Inhalt des Caches bekannt, so kann basierend auf der Anzahl der bereits gecachten Daten der nächste Zugriff bestimmt werden. Die beste Wahl besteht üblicherweise darin, auf eine Kachel zuzugreifen, für die sich die meisten Nachbarn bereits im Cache befinden. Darüber hinaus können Kacheln bevorzugt werden, welche nicht die meisten gecachten Nachbar besitzen, sich jedoch alle Nachbarn im Cache befinden. Die beschriebene Zugriffsstrategie ist in Abbildung 3.16 unter *LRU meiste Nachbarn im Cache* aufgeführt und erreicht eine Rate von 33%, bei einer Cachegröße von nach wie vor 12. Eine weitere Verbesserung auf 31% wird durch das gezielte Entfernen von gecachten Elementen, welche ihre maximale Zugriffszahl (time to life, TTL) erreicht haben erzielt. Auf diesem Wege kann das Optimum von 0% nur über eine Erhöhung der Cachegröße erreicht werden. Beim genutzten Beispieldatensatz gelang dies bei einer Cachegröße von 427, was etwa 13% des gesamten Datensatzumfangs entspricht.

Da das erneute Laden einer Kachel aufgrund der Cachegröße nicht verhindert werden kann, kann abschließend noch optimiert werden, bei welchen Kacheln dies in Kauf genommen wird. Hier bieten sich Kacheln mit geringer Punktmenge an, da diese schneller wieder in den Cache geladen werden können als solche mit vielen Punkten. Dafür wird die LRU Verdrängungsstrategie dahingehend angepasst, dass nicht zwangsläufig das älteste Element, sondern von den x-ältesten, das kleinste Element entfernt wird. Dies verringert nicht die Cache-Miss Rate, jedoch kostet damit jeder Cache-Miss weniger Ladezeit und die Laufzeiteffizienz des Verfahrens steigt.

Eine weitere Optimierungsmöglichkeit ergibt sich abschließend noch in der Reihenfolge, in der die Nachbarkacheln verarbeitet werden. Ist die Reihenfolge für den Verarbeitungsschritt unerheblich, so werden zunächst die im Cache befindlichen Daten verarbeitet und parallel dazu die restlichen Daten in den Cache geladen. Dauert die Verarbeitung der vorhanden Daten länger als das Nachladen der nicht im Cache befindlichen Daten, so hat ein Cache-Miss keine negative Auswirkung auf die Verarbeitungslaufzeit.

4 Modulare Verarbeitungskette für Mobile Mapping Daten

Mobile Mapping ermöglicht eine detaillierte und großflächige Erfassung urbaner Räume. Die erfassten Daten ermöglichen beispielsweise Bestandsaufnahmen oder eine Durchführung von kontinuierlichen Änderungsanalysen. Darüber hinaus können die Daten zur Beantwortung weiterführender (geo-) wissenschaftlicher und praktischer Fragestellungen herangezogen werden, wie beispielsweise in Eggert u. a. (2014) oder Bock u. a. (2015). In allen Fällen müssen die erfassten Daten vorgehalten, verarbeitet und analysiert und etwaige Ergebnisse i.d.R. abschließend visualisiert werden. Für das für die genannten Aufgaben verwendete Framework ergeben sich, nahezu unabhängig vom tatsächlichen Anwendungsfall, folgende Anforderungen: *erweiterbar*, *flexibel*, *wiederverwendbar*, *effizient* und *transparent*.

Sollen die Mobile Mapping Daten im wissenschaftlichen Kontext verarbeitet und analysiert werden, so muss das genutzte Framework ein hohes Maß an Erweiterbarkeit und Flexibilität mitbringen. Nur so können neue Analyseverfahren effektiv entwickelt und umgesetzt werden. Darüber hinaus müssen dafür alle Teile des Frameworks flexibel kombinierbar sein. Unabhängig vom Kontext umfassen viele Verarbeitungsabläufe häufig wiederkehrende Teilaufgaben, wie das Lesen und Schreiben von Daten oder die Ermittlung abgeleiteter Attribute. Aus diesem Grund spielt eine umfassende Wiederverwendbarkeit und Rekombination einzelner Komponenten eine entscheidende Rolle bei der Nutzbarkeit des Frameworks. Der umfängliche Charakter der zu verarbeitenden Daten erfordert darüber hinaus eine hohe Effizienz der eingesetzten Verfahren, sowie deren Implementierungen. Ist dies nicht gegeben, so ergeben sich unter Umständen unzweckmäßige Laufzeiten oder eine Verarbeitung ist aufgrund begrenzter Ressourcen gar nicht erst möglich. Die Forderung nach Transparenz meint die Möglichkeit die vorhandene Implementierung inspizieren zu können. Dies erleichtert die Entwicklung neuer und Erweiterung existierender Verfahren. Die proprietären Blackbox-artigen Closed-Source Lösungen vieler Hersteller bieten diese Möglichkeit in der Regel nicht.

Das im Rahmen dieser Arbeit entwickelte Framework zur Verarbeitung von Mobile Mapping Daten adressiert die besagten Anforderungen. Zum einen wurden sämtliche Komponenten und Verfahren selbst implementiert und stehen allen Nutzern des Frameworks als Quelltext zur Verfügung. Darüber hinaus wurde beim Entwurf und der Umsetzung auf eine feingranulare Strukturierung in Schichten, Module und Komponenten geachtet. Zusammen mit klaren Schnittstellen und möglichst geringen Abhängigkeiten wird damit eine erweiterbare, flexible und wiederverwendbare Nutzung gewährleistet. Die Effizienz der erstellten Verfahren wird, wie im vorherigen Kapitel diskutiert, über entsprechende Prozessierungsmodelle und die Verwendung geeigneter Datenstrukturen sichergestellt.

Das vorliegende Kapitel befasst sich mit der Analyse des Workflows zur Datenverarbeitung mittels der proprietären Herstellerlösungen am Bespiel der Erzeugung einer eingefärbten Punktwolke. Im Rahmen dieser Analyse wurden mögliche Anknüpfungspunkte für eigene Verarbeitungsmodule identifiziert. Anschließend wird der besagte Workflow zur Erzeugung einer eingefärbten Punktwolke erneut aufgegriffen und als Verarbeitungskette, bestehend aus Modulen des selbstentwickelten Frameworks, umgesetzt. Dazu werden die notwendigen Komponenten der Basismodule *Vorverarbeitung* und *Segmentierung und Klassifikation* beschrieben. Einen Überblick zur Einordnung in das Framework gibt Abbildung 4.1.

4.1 Analyse der beteiligten Komponenten des Herstellerworkflows

Die aufgezeichneten Daten umfassen im Wesentlichen die Trajektorie, sowie Kameraaufnahmen und Laserscanmessungen. Die Verarbeitung der erfassten Daten erfolgt dabei mittels der Herstellersoftwarepakete Applanix POSPac MMS (Vorverarbeitung der Trajektorie), sowie Riegl RiProcess (Integration der vorverarbeiteten Trajektorie, sowie der Bild und Scan Daten).

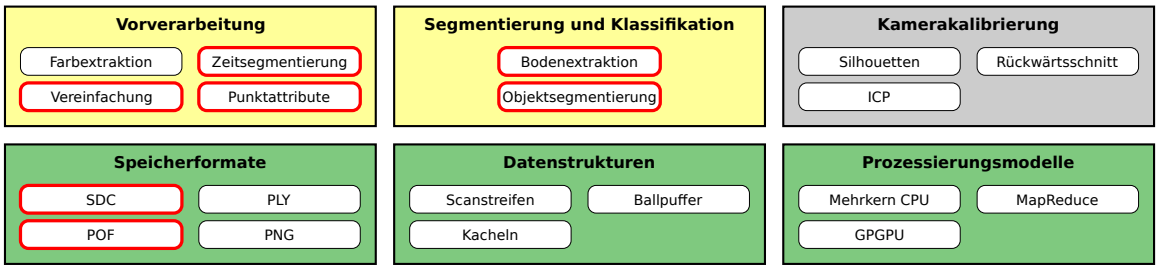


Abbildung 4.1: Einordnung der im vorliegenden Kapitel relevanten und beschriebenen Komponenten und Module des erstellten Frameworks.

Die während der Messfahrt aufgezeichnete Trajektorie (IMU und GPS Daten) kann lediglich über die Software POSPac MMS der Firma Applanix verarbeitet werden. Diese ermöglicht über zusätzliche SAPOS Korrekturdaten den Export einer verbesserten Trajektorie, welche anschließend in die Riegl Software RiProcess importiert werden kann. Dort wird die Trajektorie dann im offnen .pof Format gespeichert, was eine weitere Verwendung bzw. Verarbeitung auch ausserhalb der genannten Softwareumgebungen erlaubt.

Die rohen Laserscan Daten werden über das RiProcess Modul SDCImport in Koordinaten im scannereigenen Koordinatensystem transformiert und im gleichnamigen, ebenfalls offenen, .sdc Format gespeichert. Die weitere Verarbeitung mittels RiProcess umfasst zunächst die Transformation der Scandaten in ein übergeordnetes Koordinatensystem, hier konkret das der (verbesserten) Trajektorie, also *ETRS89*. Anschließend erfolgt die Einfärbung der Punktwolke über die erfassten Bilder. Im letzten Schritt wird die Punktwolke, inklusive etwaiger Punktattribute wie Farbe und Intensität exportiert. Abbildung 4.2 veranschaulicht den geschilderten Workflow. Sollen eigene Verarbeitungsmodule genutzt werden, so kann dies nur an Stellen geschehen, wo das genutzte Datenformat offengelegt ist. Eine direkte Verarbeitung der Rohdaten ist daher nicht möglich, da diese in den herstellereigenen proprietären Formaten vorliegen. Anknüpfungspunkte bilden somit die offenen .pof und .sdc Formate für die Trajektorie beziehungsweise Laserscandaten.

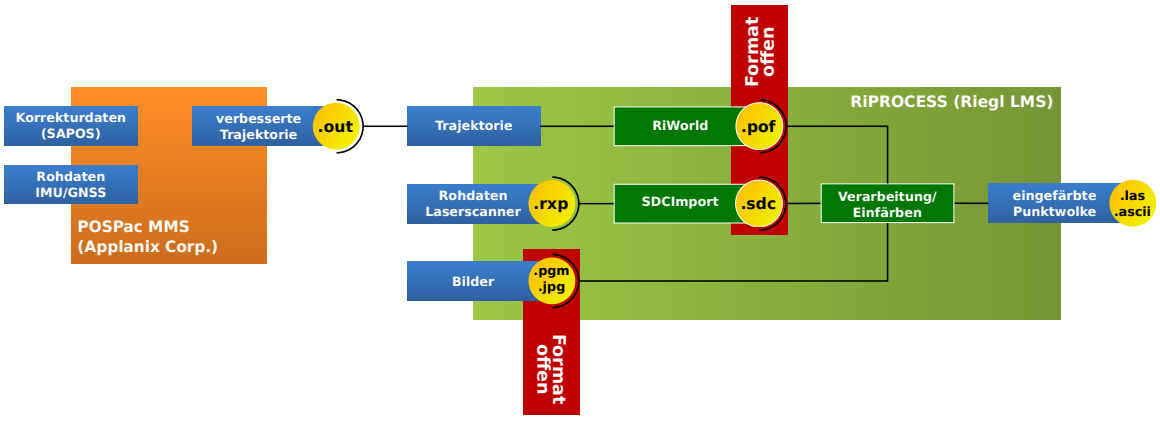


Abbildung 4.2: Applanix/Riegl Workflow Schema zur Erstellung einer eingefärbten Punktwolke.

4.2 Exemplarische modulare Verarbeitungskette

Für eine Visualisierung der Mobile Mapping Daten müssen die erfassten Rohdaten zunächst verarbeitet werden. Diese Verarbeitung muss die speziellen Eigenschaften der erfassten Daten adressieren, um zum einen effizient zu sein und zum anderen ein visuell ansprechendes Ergebnis zu erzielen. Bei den vorliegenden Mobile Mapping Daten handelt es sich wie bereits beschrieben um Massendaten in Form von Laserscanmessungen und Kamerabildern. Sämtliche Daten werden über möglichst ex-

akte Zeitstempel miteinander registriert. Zur globalen Georeferenzierung wird die Trajektorie des Erfassungsfahrzeugs hochgenau aufgezeichnet.

Ziel ist hier zunächst die Erzeugung einer eingefärbten Punktewolke. Die dafür erstellte Verarbeitungskette, bestehend aus den Modulen des entworfenen Frameworks, steht exemplarisch für weitere beliebige Verarbeitungsketten mit anderen Zielstellungen. Abbildung 4.3 zeigt die Module der exemplarischen Verarbeitungskette einschließlich der Datenstrukturen, welche den jeweiligen Modulen zugrunde liegen. Zunächst findet eine Feinkalibrierung der Kameras statt, welche im Detail in Kapitel 5 beschrieben ist. Anschließend werden die möglichen Farbwerte der Scanpunkte aus den erfassten Kamerabildern wie in Kapitel 6 beschrieben extrahiert. Beide Schritte basieren auf der sequentiellen Verarbeitung der originalen Scanstreifen und nutzen die in Abschnitt 3.2.2 erläuterte Pufferstrategie. Die ermittelten Farbwerte aller Punkte lassen sich nicht ohne weiteres an die originalen Eingangsdaten anhängen. Daher wird eine Kachelung der Daten, wie in Abschnitt 3.2.3 gezeigt, durchgeführt. Anschließend werden die Scanpunktdaten, einschließlich aller ermittelten Farbwerte im flexiblen PLY Datenformat gespeichert. Auf Basis dieser Kachelstruktur verarbeiten die in den folgenden Abschnitten beschriebenen Basismodule die Punktwolkendaten. Im weiteren Verlauf der Verarbeitungskette zur Erzeugung einer eingefärbten Punktwolke werden zunächst die Punktwolkendaten einer Vorverarbeitung unterzogen, welche unter anderem essentielle Punktattribute bestimmt. Mittels dieser Attribute wird eine Segmentierung und Klassifikation durchgeführt. Diese wird für die abschließende Bestimmung des finalen Farbwertes aller Punkte der Punktwolke benötigt. Ergebnis dieser Verarbeitungskette ist eine eingefärbte Punktwolke, welche in der besagten Kachelstruktur im PLY Dateiformat gespeichert vorliegt.

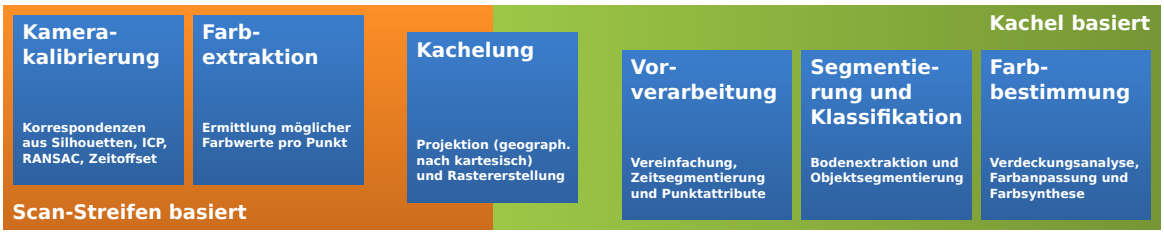


Abbildung 4.3: Modularisierte exemplarische Verarbeitungskette zur Erzeugung einer eingefärbten Punktwolke mit zugrundeliegenden Datenstrukturen.

4.3 Vorverarbeitungsmodul

Das Vorverarbeitungsmodul gehört zu den Basismodulen des Frameworks und umfasst elementare Komponenten, wie Verfahren zur Bestimmung diverser Punktattribute oder der Vereinfachung der Punktwolke, welche häufig vor etwaigen Analyse- oder Verarbeitungsschritten ausgeführt werden.

4.3.1 Vereinfachung

Bei der Erfassung der Mobile Mapping Daten kann es vorkommen, dass das Erfassungsfahrzeug anhalten muss. Je nach zu erwartender Standzeit sollte die Erfassung entsprechend unterbrochen werden. Wird dies nicht berücksichtigt oder ist die Standzeit länger als erwartet und die Erfassung wurde nicht unterbrochen, so wird dieselbe Scanzeile von beiden Scannern wiederholt erfasst. Die wiederholte Erfassung der selben Punkte führt in der resultierenden Punktwolke zu einer sehr hohen Punktdichte. Darüber hinaus wird an diesen Stellen das Rauschverhalten der Laserscanmessungen deutlich. Abbildung 4.4 zeigt eine nach Punktdichte eingefärbte Punktwolke, bei der bei längerer Standzeit die Erfassung nicht unterbrochen wurde. Blau gefärbte Punkte weisen eine geringe Dichte auf, wohingegen rot gefärbte Punkte eine hohe Dichte aufweisen. Deutlich zu erkennen sind die Regionen, z.B. links an der Fassade oder auf der Fahrbahn (Abbildung 4.4 (rechts)), mit hoher Punktdichte, die wiederholt erfasst wurden. Das typische kreuzförmige Muster wird dabei durch die Anordnung der Laserscanner auf dem Messsystem hervorgerufen.

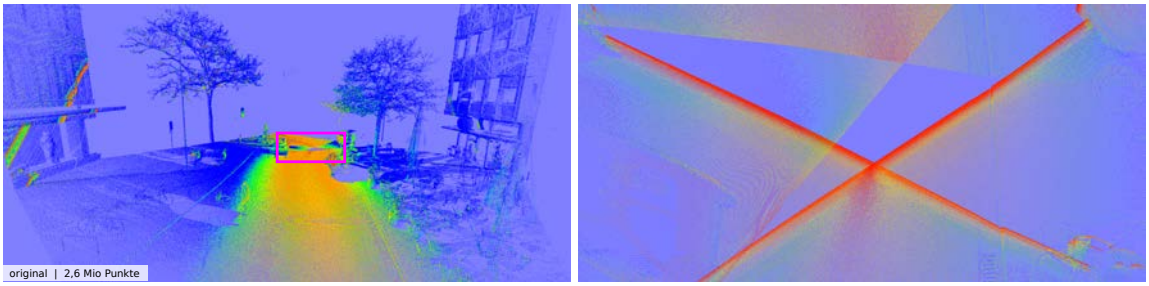


Abbildung 4.4: Nach Punktdichte eingefärbte Punktewolke (rot: hohe Dichte; blau: geringe Dichte).

Diese Regionen mit hoher Punktdichte weisen einige Nachteile auf. Bei einer Visualisierung der Daten entsteht an den entsprechenden Regionen ein deutlich sichtbarer Übergang (Kreuz), welcher unter Visualisierungsaspekten nicht erwünscht ist. Des Weiteren führt das vorhandene Rauschen zu Problemen bei der weiteren Verarbeitung der Scanpunkte. Insbesondere bei Verarbeitungsschritten (vgl. Abschnitt 4.3.3), die auf der Bestimmung der k-nächsten Nachbarn beruhen, können bei dem vorhandenen Rauschen problematische Resultate entstehen. Abschließend wird der Speicheraufwand, sei es für die persistente Speicherung oder das Vorhalten im Arbeitsspeicher unnötig erhöht.

Die gezeigten Probleme können durch eine eine Vereinfachung, bzw. Ausdünnung der Punktwolke, welche starke Punktdichteunterschiede ausgleicht, behoben werden. Dazu wird ein dreidimensionales Belegungsraster generiert und für jeden Punkt die entsprechende Rasterzelle überprüft. Ist die Zelle bereits belegt, wird der Punkt entfernt, ist die Zelle frei, so wird der Punkt behalten und die Zelle als belegt markiert. Die gewünschte Punktdichte kann mit der Zellgröße des Rasters gesteuert werden. Abbildungen 4.5 bis 4.9 zeigen die Ergebnisse von Vereinfachungen mit unterschiedlichen Zellgrößen. Bei einer Zellgröße von etwa 1cm wurden die wesentlichen Differenzen in der Punktdichte eliminiert. Bei kleineren Zellen (bspw. 0,5cm) bleiben nach wie vor signifikante Unterschiede bestehen, wohingegen bei größeren Zellen ($\geq$2cm) der Detailgrad der Punktwolke sichtbar sinkt, was für eine Visualisierung wiederum nachteilig zu bewerten ist.

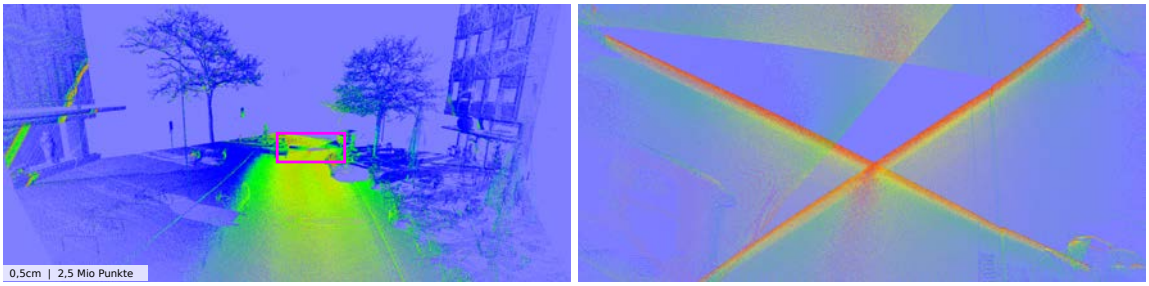


Abbildung 4.5: Nach Punktdichte eingefärbte Punktewolken bei einer Zellgröße von 0,5cm (rot: hohe Dichte; blau: geringe Dichte).

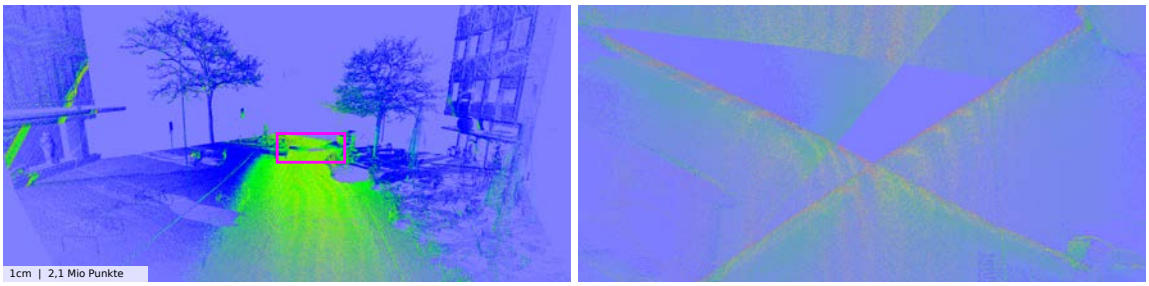


Abbildung 4.6: Nach Punktdichte eingefärbte Punktewolken bei einer Zellgröße von 1cm (rot: hohe Dichte; blau: geringe Dichte).

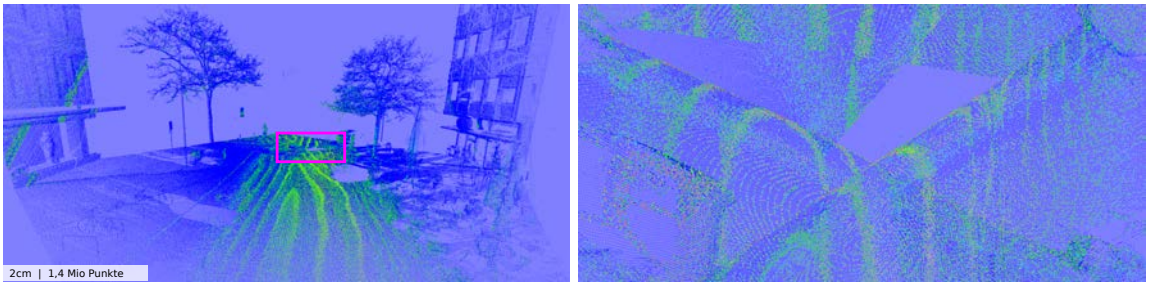


Abbildung 4.7: Nach Punktdichte eingefärbte Punktewolken bei einer Zellgröße von 2cm (rot: hohe Dichte; blau: geringe Dichte).

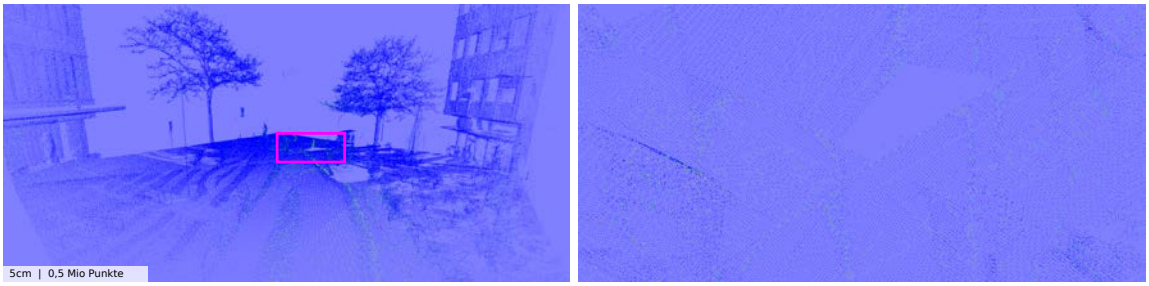


Abbildung 4.8: Nach Punktdichte eingefärbte Punktewolken bei einer Zellgröße von 5cm (rot: hohe Dichte; blau: geringe Dichte).

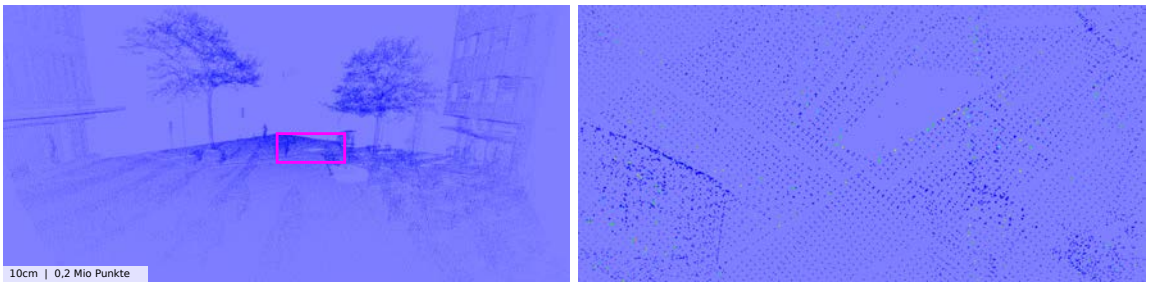


Abbildung 4.9: Nach Punktdichte eingefärbte Punktewolken bei einer Zellgröße von 10cm (rot: hohe Dichte; blau: geringe Dichte).

4.3.2 Zeitsegmentierung

Je nach Trajektorie des Erfassungsfahrzeugs kann es vorkommen, dass dasselbe Objekt bzw. dieselbe Region mehrmals in einem gewissen zeitlichen Abstand erfasst wurde. Jedoch ist beispielsweise die Höhe (im Gegensatz zur Lage) der aufgezeichneten Trajektorie nicht sehr stabil, was dazu führt, dass die zu unterschiedlichen Zeiten erfassten Punkte einen Höhenunterschied von bis zu 0,5 Metern aufweisen können. Dies kann zu Problemen bei nachfolgenden Verarbeitungsschritten führen. Abbildung 4.10 zeigt eine Punktwolke welche während einer Erfassungsfahrt zu drei Zeitpunkten erfasst wurde. Bei näherer Betrachtung wird ein sichtbarer Höhenversatz der drei Erfassungen deutlich.

Beschränkt man sich auf die Anpassung der Höhen innerhalb einer Kachel, kann dies mittels einer zeitlichen Segmentierung der Punkte und der anschließenden Höhenanpassung aller Zeitsegmente erfolgen. Bei der zeitlichen Segmentierung der Punkte muss beachtet werden, dass die Punkte bereits einen, durch die Anordnung der Laserscanner bedingten, zeitlichen Versatz aufweisen. Wie in Abschnitt 2.1.2 beschrieben erfassen die verwendeten Laserscanner die gleichen Objekte mit einem zeitlichen Versatz von einigen Sekunden, in Abhängigkeit der Geschwindigkeit des Erfassungsfahrzeugs. Dies muss beim Segmentierungsschritt berücksichtigt werden um eine Übersemgentierung zu vermeiden.

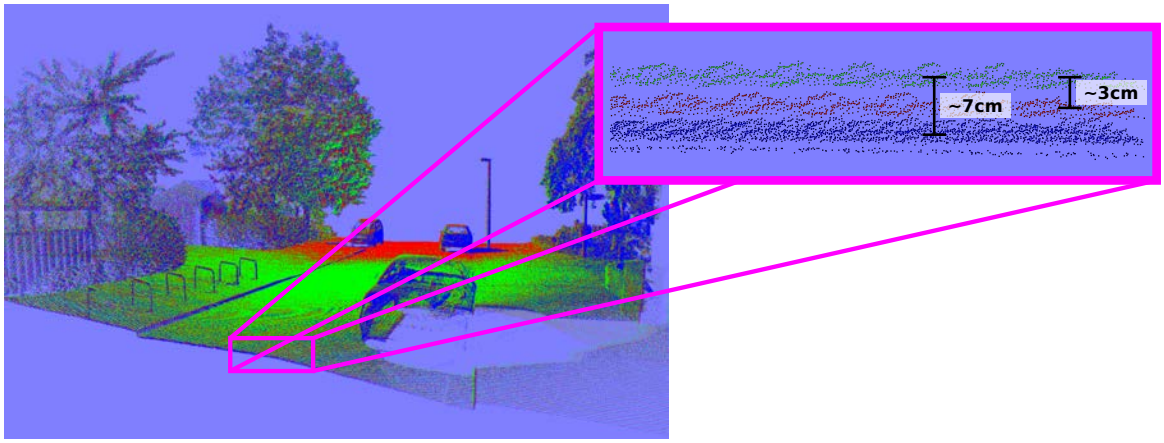


Abbildung 4.10: Nach Erfassungszeit eingefärbte Punktewolke mit sichtbarem Höhenversatz der jeweiligen Zeitcluster.

Zur Identifikation der einzelnen zeitlich zusammenhängenden Segmente werden im ersten Schritt alle Punkte nach ihrer Erfassungszeit sortiert und anschließend alle zeitlichen Lücken die einen gewissen Grenzwert überschreiten bestimmt. Dieser Grenzwert muss wie beschrieben größer als der zeitliche Versatz der beiden Laserscanner sein, bspw. $\geq$1min. Zur Bestimmung der Höhenanpassung wird zunächst das Zeitsegment mit den meisten Punkten als Referenzsegment bestimmt. Die Höhen aller weiteren Segmente werden diesem Referenzsegment angepasst. Dazu wird zu jedem Punkt eines Zeitsegments der nächstgelegene Punkt aus dem Referenz-Segment ermittelt und der jeweilige Höhenunterschied bestimmt. Als Höhenanpassung für das gesamte Zeitsegment wird der Median aller ermittelter Höhenunterschiede herangezogen.

Problematisch dabei ist jedoch die Wahl des Referenzsegments. Werden in benachbarten Kacheln unterschiedliche Referenzsegmente gewählt, so kann dies in sichtbaren Sprüngen an den Kachelgrenzen resultieren. Eine Möglichkeit wäre, die gewählten Referenzsegmente der Nachbarkacheln bei der Verarbeitung einer Kachel zu berücksichtigen. Eine weitere Möglichkeit besteht in der Anpassung der Trajektorie des Erfassungsfahrzeugs auf Basis der einzelnen Höhendifferenzen mittels einer Gesamtausgleichung.

4.3.3 Bestimmung von Punktattributen

Viele der im Rahmen dieser Arbeit vorgestellten Verfahren beruhen auf der Analyse gewisser Attribute für jeden Punkt. Diese Attribute werden in der Regel nicht vom Aufnahmesystem erfasst und müssen daher auf Basis der vorhandenen Punktdaten und -attribute ermittelt werden.

4.3.3.1 Punktnormale

Die Normale eines Punktes gibt Auskunft über die Orientierung der Oberfläche, zu der der jeweilige Punkt gehört. Die Kenntnis der Punktnormalen ist Voraussetzung des in Abschnitt 4.4 vorgestellten Segmentierungsverfahrens. Darüber hinaus wird sie für die bei der Visualisierung von Punktwolken eingesetzten Beleuchtungsmodelle benötigt. Das für diese Arbeit umgesetzte Verfahren zur Bestimmung der Punktnormalen beruht auf der Ermittlung der Normalen der ausgleichenden Ebene der Nachbarpunkte eines Punktes (PCA) und ist im Detail bereits in Abschnitt 3.1.2 beschrieben. Abbildung 4.11 zeigt die Nutzung der bestimmten Punktnormalen bei der Visualisierung von Punktwolken. Abbildung 4.11 (links) zeigt eine farblose Punktwolke ohne Punktnormalen. Nach der Bestimmung der Normalen, welche in Abbildung 4.11 (Mitte) dargestellt sind, lassen sich die Punkte über ein entsprechendes Beleuchtungsmodell schattieren. Wie Abbildung 4.11 (rechts) zeigt, ermöglicht dies eine plastischere Darstellung, selbst bei noch farblosen Punkten.

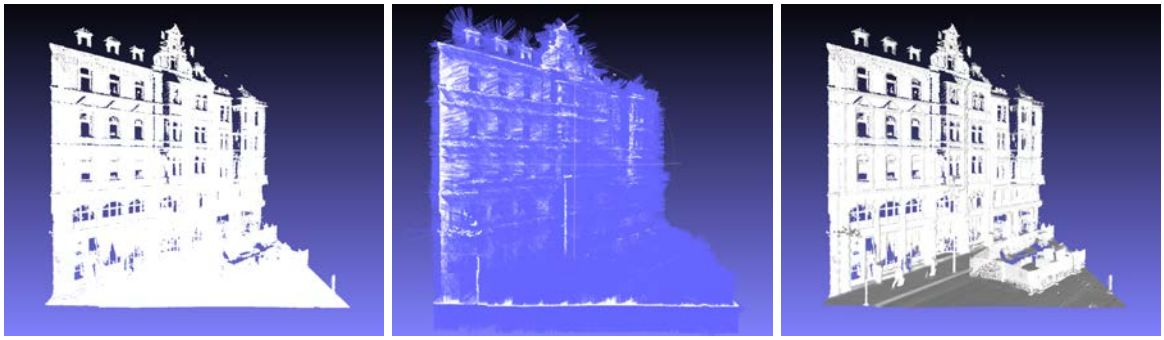


Abbildung 4.11: Farblose Punktwolke ohne Punktnormalen (links), mit gerenderten Punktnormalen (Mitte) und mittels Beleuchtungsmodell schattiert (rechts).

4.3.3.2 Dimensionalität

Ein für die weitere Verarbeitung der Daten nützliches Punktattribut ist die Dimensionalität des Punktes. Sie spiegelt grob die Art der Verteilung der Punkte in der direkten Nachbarschaft wieder. Mögliche Werte sind dabei *räumlich* (engl. scatter), *planar*, *linear* und *zylindrisch*. Die Bestimmung der Dimensionalität erfolgt nach dem in Demantke u. a. (2011) beschriebenen Verfahren über eine Hauptkomponentenanalyse (engl. principal componentent analysis, PCA). Während Demantke u. a. (2011) lediglich die Ermittlung der Werte räumlich, planar und linear beschreibt, erweitert Monnier u. a. (2012) den Ansatz um den Wert zylindrisch, was die Klassifikation von zylindrischen Objekten, bspw. Baumstämme, erleichtert.

Abbildung 4.12 zeigt zwei Beispiele bei denen die Punkte nach ihrer ermittelten Dimensionalität eingefärbt wurden. Große Teile der Fassade als auch des Boden wurden korrekt als planar (grün) erkannt. Die Laternenpfosten wurden als zylindrisch (rot) bestimmt. Abbildung 4.12 (rechts) zeigt deutliche die Dimensionalitäten eines Baumes, welche im Stammbereich zylindrisch (rot) bis planar (grün) sind und in der Baumkrone überwiegend räumlich (blau) mit einigen linearen (magenta) Bereichen. Auffällig ist dabei, dass Kanten (Fensterrahmen, etc) häufig als zylindrisch detektiert werden. Werden die Dimensionalitäts-Attribute im Rahmen von Segmentierungs- oder Klassifikationsverfahren genutzt, muss dies entsprechend berücksichtigt werden.

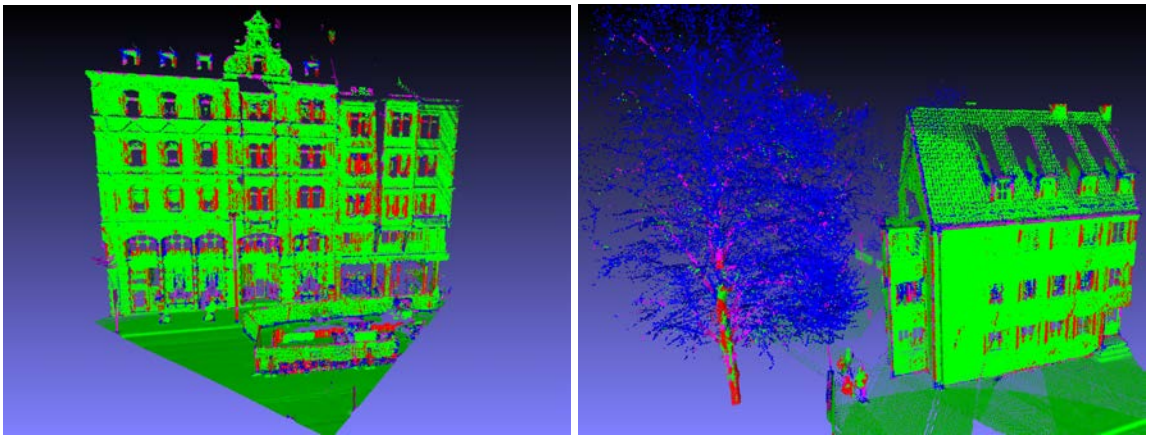


Abbildung 4.12: Punktdimensionalität - magenta:linear, grün:planar, blau:räumlich und rot:zylindrisch

4.4 Segmentierung und Klassifikation

Einige Verfahren, der später beschriebenen Verarbeitungsmodule, setzen eine segmentierte und in Teilen auch klassifizierte Punktwolke voraus. Daher wurde ein Basismodul *Segmentierung und Klas-*

sifikation entworfen und umgesetzt, welches Verfahren zur Segmentierung und Klassifikation als Komponenten bereitstellt. Bei der Umsetzung der Verfahren wurden die besonderen Eigenschaften der zu verarbeitenden Mobile Mapping Daten, wie Lücken durch Verdeckungen oder ungleichmäßige Punktdichten, berücksichtig. Die im folgenden vorgestellten Verfahren setzen das Vorhandensein diverser Punktattribute, Punktnormalen und Dimensionalitäten, voraus. Sind die benötigten Attribute nicht bereits vorhanden, müssen diese zunächst mittels des zuvor vorgestellten Vorverarbeitungsmoduls ermittelt werden. Das im folgenden Abschnitt 4.4.1 beschriebene Verfahren zur Extraktion von Bodenpunkten ist eine kombinierte Segmentierung und Klassifikation aller Punkte, welche dem Boden (Straße, Gehweg, usw.) zugehörig sind. Nach der Extraktion der Bodenpunkte kann eine Segmentierung einzelner Objekte, beschrieben in Abschnitt 4.4.2, erfolgen.

4.4.1 Bodenextraktion

Die Segmentierung bzw. Klassifikation des Bodens basiert im Kern auf einem gewöhnlichen Regionenwachstum (engl. Region-Growing) nach Muerle und Allen (1968). Dabei werden benachbarte Bereiche mit ähnlichen Eigenschaften zu Regionen zusammengefasst. Dazu müssen zunächst die entsprechenden Eigenschaften einer Bodenregion definiert werden. Ausgehend von den zu verarbeitenden Mobile Mapping Daten kann der Boden über drei Eigenschaften definiert werden:

- überwiegend planar
- Punktnormalen in Z-Richtung
- Punkte mit geringer Höhe

Die beiden erstgenannten Eigenschaften dienen als Entscheidungskriterium für das Zusammenfassen benachbarter Bereiche, wohingegen die Punkthöhe lediglich zur Bestimmung des Saatbereichs genutzt wird. Zur Bestimmung des Saatbereichs werden bis zu 1% der Punkte gewählt, welche eine eine Normale in Z-Richtung und die geringste Höhe aufweisen. Der resultierende Saatbereich ist beispielhaft in Abbildung 4.13 (links) dargestellt.

Ausgehend vom zuvor bestimmten Saatbereich wird das Region-Growing durchgeführt. Die Effizienz des Region-Growings stützt sich grundsätzlich auf das schnelle Finden der Nachbarbereiche. Dies wird wiederum durch die Nutzung einer Rasterdatenstruktur garantiert. Ausgehend von den Rasterzellen der Punkte des Saatbereichs werden für das Region-Growing alle Nachbarzellen herangezogen. Zur Überprüfung der geforderten Eigenschaften werden zwei Filter genutzt. Das Planaritätskriterum wird dabei für die gesamte Rasterzelle überprüft, dafür wird die Verteilung der Dimensionalitäten der Punkte der Zelle bestimmt. Ist nicht die Mehrheit aller Zellpunkte planar wird die gesamte Zelle nicht für das Region-Growing herangezogen. Dies garantiert zum einen, dass lediglich planare Bereiche betrachtet werden, gleicht zum anderen jedoch auch kleinere Ungenauigkeiten der Dimensionalitätsbestimmung aus. Die Überprüfung des Punktnormalen-Kriteriums findet dahingegen pro Punkt statt.

Befinden sich, bspw. durch Verdeckungen hervorgerufene Lücken in der Punktwolke, wird das durchgeführte Region-Growing entsprechend abgeschnittene Bereiche nicht der Bodenregion zuordnen. Daher wird im Anschluss an das Region-Growing eine ausgleichende Ebene zur ermittelten Punktmenge approximiert. Daraufhin werden alle Punkte zu der Menge der segmentierten Bodenpunkten hinzugefügt, welche ebenfalls dem Punktnormalen-Kriterium genügen und einen geringen Abstand zur ausgleichenden Ebene haben. Abschließend wird mit dieser Punktmenge ein erneutes Region-Growing durchgeführt. Dies ermöglicht die Segmentierung von nicht zusammenhängenden Bodenflächen, sofern keine größeren Höhenunterschiede zwischen den jeweiligen Teilstücken vorliegen. Das Ergebnis zeigt Abbildung 4.13 (rechts).

Während die definierten Eigenschaften im allgemeinen gut urbane Bodenflächen repräsentieren, lassen diese sich nicht auf beliebige Bodenflächen verallgemeinern. Je nach Bewuchs muss die Dimensionalitätsverteilung nicht mehr planar geprägt sein. Sollen Abhänge oder ähnliche Unebenheiten ebenfalls als Boden segmentiert werden, so ist das Punktnormalen-Kriterium ebenfalls aufzuweichen.

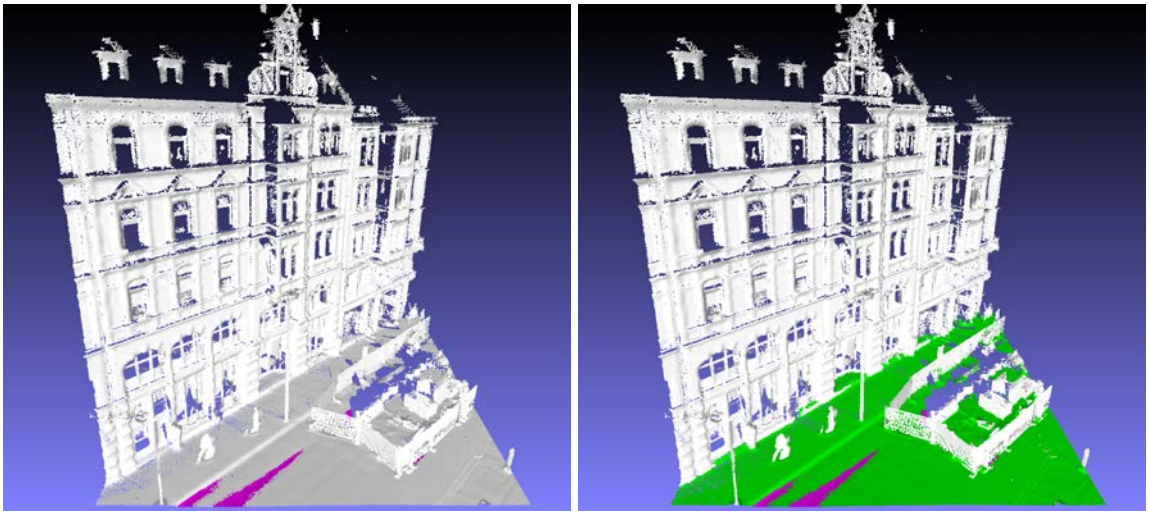


Abbildung 4.13: Region-Growing basierte Bodensegmentierung ausgehend von Saatbereich in Magenta (links) und Ergebnis (rechts).

4.4.2 Objektsegmentierung

Wie schon die zuvor beschriebene Bodenextraktion basiert die Segmentierung der einzelnen Objekte ebenfalls auf dem Region-Growing Ansatz. Das gewählte Zugehörigkeitskriterium ist dabei lediglich die Distanz der Punkte. Da nahezu alle Objekte über den Boden miteinander *verbunden* sind, muss dieser zunächst aus der Punktwolke entfernt werden. Anschließend wird ein beliebiger Punkt aus der verbliebenen Punktwolke als Saatpunkt gewählt und davon ausgehend ein Region-Growing durchgeführt. Ist das Region-Growing beendet, wurden alle zum Objekt gehörenden Punkt segmentiert und aus der Punktwolke entfernt. Dies wird wiederholt bis keine Punkte mehr in der Punktwolke vorhanden sind. Abbildung 4.14 zeigt das Resultat für die aus Abbildung 4.13 bekannte Szene. Jedes segmentierte Objekt wurde mittels einer zufälligen Farbe eingefärbt.

Je nach gewählter Rasterzellgröße kann dabei über- oder untersegmentiert werden. Grundsätzlich lassen sich beide Fälle über eine Feinabstimmung der Rasterzellgröße adressieren, jedoch nicht komplett vermeiden. Im Falle der genutzten Mobile Mapping Daten findet häufig eine Übersegmentierung statt, da Teile eines Objektes, aufgrund von Verdeckungen, gelegentlich nicht zusammenhängend erfasst wurden (z.B. die Dachgauben des Gebäudes aus Abbildung 4.14). Um solchen Übersegmentierungen entgegenzuwirken wird zu jedem Objekt die konvexe Hülle ermittelt. Schließt die konvexe Hülle eines Objektes die eines anderen vollständig ein, so werden beide Punktmenge zu einem Objekt verschmolzen. Die Hüllen der besagten Dachgauben werden jedoch nicht vollständig von der konvexen Hülle der Fassade eingeschlossen, daher wurden diese nach wie vor als eigenständige Objekte segmentiert. Dies könnte durch ein Verschmelzen von Objekten behoben werden, deren konvexe Hüllen sich nur teilweise überlappen. Da dies jedoch zu potentiellen Untersegmentierungen führen kann, werden hier nur vollständige Überlappungen verschmolzen.

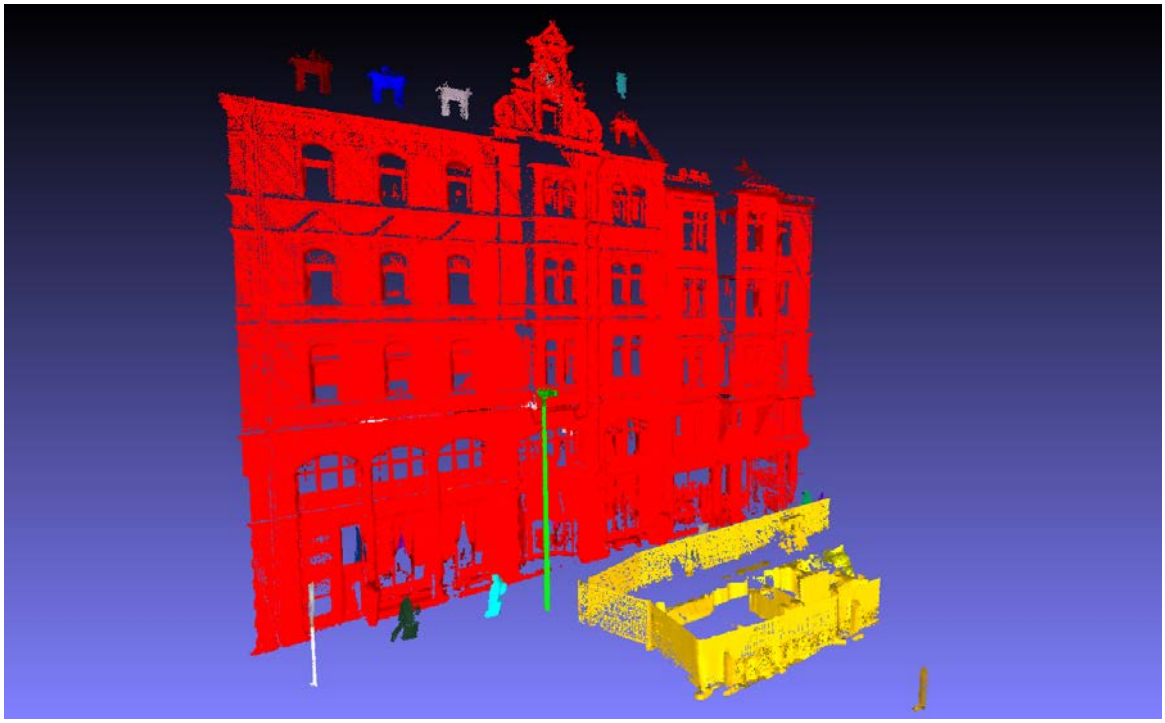


Abbildung 4.14: Ergebnis der Region-Growing basierten Objektsegmentierung. Segmentierte Objekte wurden mit zufälligen Farben eingefärbt.

5 Sensordatenintegration: Kalibrierung[1] der Kameraorientierung

Für eine Einfärbung der Scanpunkte wird die exakte Position und Orientierung aller genutzten Kameras benötigt. Die jeweiligen Kamerapositionen und -orientierungen werden jeweils manuell auf Basis von vier gewählten Punkten mittels der Riegl-Software bestimmt. Auf Grund der geringen Anzahl an genutzten Punktkorrespondenzen und Ungenauigkeiten in der Systemanordnung sind die resultierenden Parameter unter Umständen nicht sehr genau. Dies kann zu falschen Pixelkoordinaten für die jeweiligen Scanpunkte führen. Genauere und robustere Parameter können über eine signifikante Erhöhung der genutzten Punktkorrespondenzen ermittelt werden. Abbildung 5.1 gibt wie gehabt einen Überblick zur Einordnung des Moduls in das Gesamtframework. Das im folgenden beschriebene Verfahren zur Kalibrierung der Kameraparameter nutzt dabei lediglich Kernmodule zum Lesen und Schreiben der Daten, sowie entsprechende Datenstrukturen zum Zugriff auf selbige.

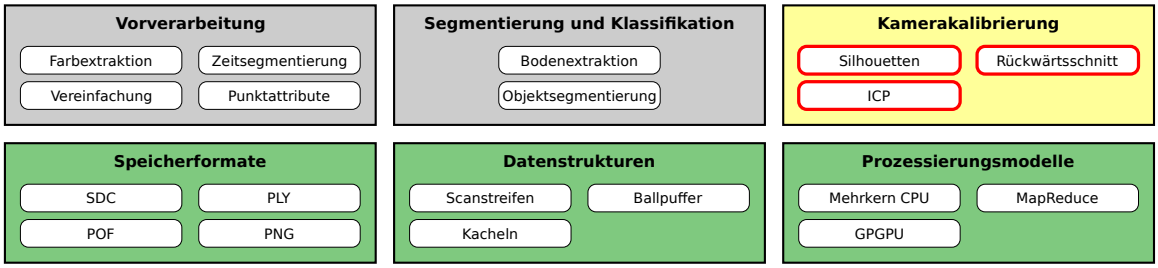


Abbildung 5.1: Einordnung des Moduls Kamerakalibrierung in das Gesamtframework.

5.1 Zeitstempelabweichung

Die Fusion der Daten des Mobile Mapping Systems erfolgt über synchronisierte Zeitstempel, welche zu jeder Messung, seien es nun Bilder oder Scanpunkte, aufgezeichnet werden. Für eine exakte Fusion der Daten sind hoch genaue Zeitstempel unabdingbar. Dies bei einem komplexen System aus mehreren autarken Elementen zu gewährleisten ist unter Umständen jedoch nicht immer möglich. Selbst bei einer perfekten Kalibrierung der Kamera führen schon minimale Zeitstempelabweichungen im Millisekundenbereich zu signifikanten Abweichungen bei der Projektion der Scanpunkte in die Kamerabilder. Abbildung 5.2 veranschaulicht den genannten Effekt bei einer Fahrgeschwindigkeit von etwa $45km/h$ und einem Zeitversatz von $\pm 10ms$, was einer Verschiebung der Kameraposition von $\pm 12,5cm$ in Fahrtrichtung entspricht. Diese Verschiebung der Kameraposition resultiert im Fall der Laterne in der rechten Bildhälfte in einer Verschiebung der Pixelkoordinaten um $25px$ und im Falle der Schilder am linken Bildrand in einer Verschiebung um $80px$.

Experimente haben gezeigt, dass beim vorliegenden Mobile Mapping System solche Zeitstempelabweichungen auftreten. Darüber hinaus hat sich gezeigt, dass diese Abweichungen über eine Messfahrt hinweg konstant sind. Eine erfolgreiche Feinkalibrierung der Kameras muss dem genannten Sachverhalt folglich Rechnung tragen.

[1] als gegenseitige Orientierung von Kamera und Laserscanner und somit keine Kamerakalibrierung im geodätisch / photogrammetrischen Sinne.

Abbildung 5.2: Auswirkung eines Zeitversatzes von -10ms (links, rot) und +10ms (rechts, blau), gegenüber korrigiertem Zeitversatz 0ms (mitte, grün) bei jeweils korrekt kalibrierter Kamera.

5.2 Ansatz

Der für die Feinkalibrierung genutzte Ansatz fußt, wie auch die herstellerspezifische Lösung, auf einer Bestimmung der Kameraparameter mittels eines Rückwärtsschnitts. Die zu ermittelnden Parameter sind dabei die Position der Kamera bzw. deren Projektionszentrum (X_0, Y_0, Z_0) im Fahrzeug-Koordinatensystem, sowie deren Orientierung (ρ, ϕ, κ). Um die zuvor aufgezeigten Zeitstempelabweichungen zu adressieren, wird über die sechs genannten Parameter hinaus ein entsprechender Parameter für einen möglichen Zeitversatz Δt eingeführt. Wie bereits in Abschnitt 2.6 skizziert, müssen, für die Durchführung des besagten Rückwärtsschnitts, zunächst korrespondierende Paare von Scan- und Bildpunkten ermittelt werden. Das bisherige Verfahren sah dafür eine manuelle Auswahl dieser Paare vor. Aufgrund des verhältnismäßig hohen manuellen Aufwandes sind allerdings nur eine geringe Anzahl an Paaren und Bildern möglich. Dies wiederum kann zu ungenauen Parametern führen. Aus diesem Grund werden beim vorgestellten Ansatz automatisiert eine hohe Anzahl an Punktkorrespondenzen ermittelt.

Der vorgestellte Ansatz besteht aus drei Schritten. Im ersten Schritt werden, in Anlehnung an Nüchter u. a. (2011) (vgl. Abschnitt 2.6), Silhouetten sowohl aus den Kamerabildern, als auch aus den Laserscandaten extrahiert. Im Anschluss werden die zuvor extrahierten Silhouetten mittels ICP (Iterative Closest Point) zur Deckung gebracht und korrespondierende Paare von Punkten der Kamerabilder und Scanpunkten bestimmt. Abschließend werden die sieben Kameraparameter über einen Rückwärtsschnitt auf Basis der gefundenen Korrespondenzpaare berechnet. Abbildung 5.3 fasst die vorgestellten Schritte als UML Aktivitätsdiagramm zusammen.

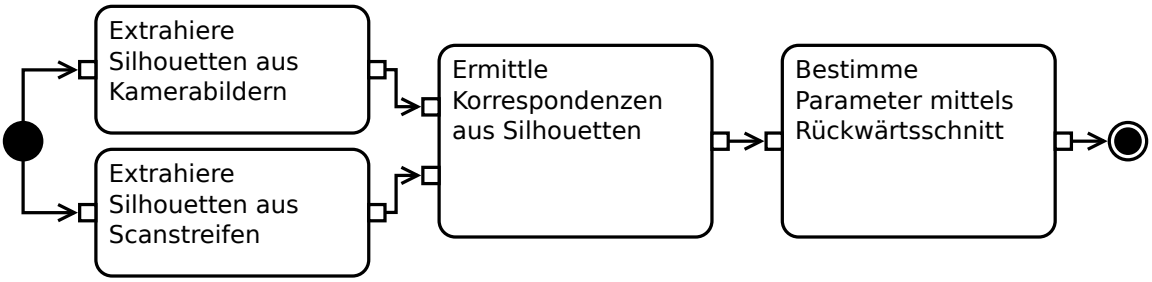


Abbildung 5.3: Ansatz zur Kalibrierung der Kameraparameter als UML Aktivitätsdiagramm.

5.3 Extraktion von Silhouetten

Für eine automatisierte Bestimmung der benötigten korrespondierenden Punktepaare müssen Scanpunkten Bildpunkte bzw. umgekehrt zugeordnet werden. Ein korrespondierendes Paar ist gefunden, wenn der Bildpunkt und eine Entfernungsmessung, also ein Scanpunkt, ein und dasselbe Objekt repräsentieren. Projiziert man den Scanpunkt also mit den richtigen Parametern, so müssen die resultierenden Bildkoordinaten dem korrespondieren Bildpunkt entsprechen. Um nun eine leichte Zuordbarkeit von Punkten aus Bildern und aus Laserscanmessungen zu erreichen, sind Strukturen nötig, welche sich sowohl in den Kamerabildern als auch in den Laserscanmessungen automatisiert finden lassen. Der vorgestellte Ansatz nutzt als solche Struktur Objektsilhouetten. Im urbanen Raum gehören diese Silhouetten in der Regel zu Gebäuden und zur Straßenmöblierung. Daher können diese Silhouetten auch als *Skyline* bezeichnet werden.

Um aus den Kamerabildern die besagte Silhouette zu extrahieren wird das Bild zunächst in ein Binärbild umgewandelt. Dies dient dazu den Himmel von den übrigen Objekten zu segmentieren. Anschließend wird mittels α-Shapes nach Edelsbrunner u. a. (1983) ein Polygon um das Objektsegment gebildet. Abschließend werden problematischen Bereiche vom Polygon entfernt. Dazu zählen Abschnitte wie der Bildrand welcher keine Trennung von Vorder- und Hintergrund repräsentiert und Vegetation. Letztere äussern sich als stark rauschende, gezackte Liniensegmente, welche die Gefahr von fehlerhaften Korrespondenzen erhöhen.

Für die Extraktion einer Silhouette aus den Laserscandaten müssen selbige zunächst in dieselbe Form wie die Kamerabilder gebracht werden. Dafür werden die in Frage kommenden Scanpunkte mittels der bekannten (u.U. groben) Kameraparametern in ein Scanpunktbild projiziert. In den drei Farbkanälen des Scanpunktbildes werden anstelle der Farbinformation die Koordinaten des jeweiligen Scanpunktes gespeichert. Analog zu den Kamerabildern wird dann mittels α-Shapes ein Polygon um den Objektbereich gebildet, welches die finale Silhouette der Laserscandaten darstellt.

5.3.1 Extraktion von Silhouetten aus Kamerabildern

Wie zuvor erläutert wird ein Kamerabild in ein Binärbild konvertiert, welches die Bildpunkte in Himmel (Hintergrund) und Objekte (Vordergrund) segmentiert. Für diese Konvertierung muss ein Schwellwert bestimmt werden, welcher die entsprechende Binärzuordnung ergibt. Anschließend wird das Silhouettenpolygon mittels α-Shapes erzeugt und von problematischen Abschnitten befreit. Abbildung 5.4 fasst die besagten Schritte in einem UML Aktivitätsdiagramm zusammen.

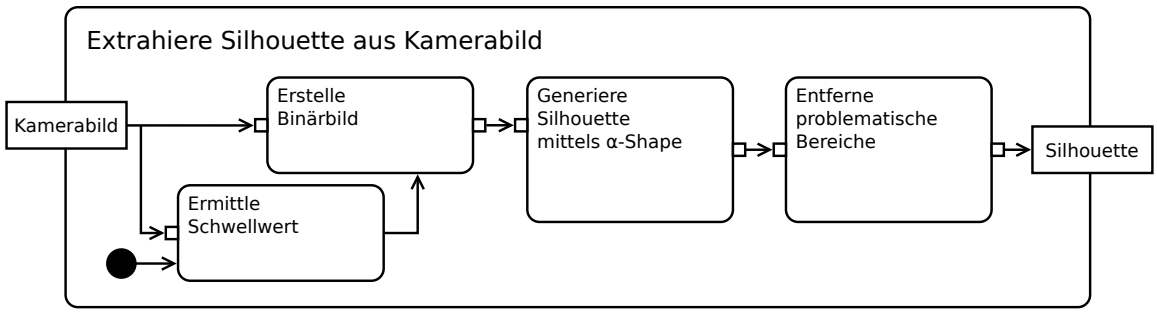


Abbildung 5.4: Extraktion einer Silhouette aus einem Kamerabild als UML Aktivitätsdiagramm.

5.3.1.1 Histogrammbasierte Schwellwertbestimmung

Davon ausgehend, dass der Himmel i.d.R. heller ist als etwaige Vordergrundobjekte, ist ein Schwellwert gesucht, welcher den *hellen* Himmel vom *dunklen* Vordergrund trennt. Dieser Schwellwert hängt jedoch von vielen Faktoren, wie Belichtungszeit, Sonnenstand, u.ä. ab. Zur automatischen Ermittlung eines geeigneten Schwellwerts wird ein Grauwerthistogramm des Bildes erzeugt, welches die

Häufigkeiten der jeweiligen Graustufen repräsentiert. In diesem Histogramm wird nun das Helligkeitsmaximum h_{max} mit $128 \leq h_{max} \leq 255$ bestimmt (maximaler Wert im Bereich 128 bis 255). h_{max} repräsentiert damit den prominentesten (hellen) Grauwert. Nach der anfänglichen Prämisse gehört dieser zum Hintergrund (Himmel). Nun muss noch ein Wert zur Trennung von Hinter- und Vordergrund ermittelt werden. Dieser muss kleiner als das gerade bestimmte h_{max} sein. Als Schwellwert s zur Trennung von Hintergrund (Himmel) und Vordergrund (Objekte) wird daher das Minimum im Bereich zwischen 128 und h_{max} gewählt. Abbildung 5.5 stellt die Histogrammbasierte Schwellwertermittlung (Mitte) und das bei Anwendung auf ein Kamerabild (links) resultierende Binärbild (rechts) dar.

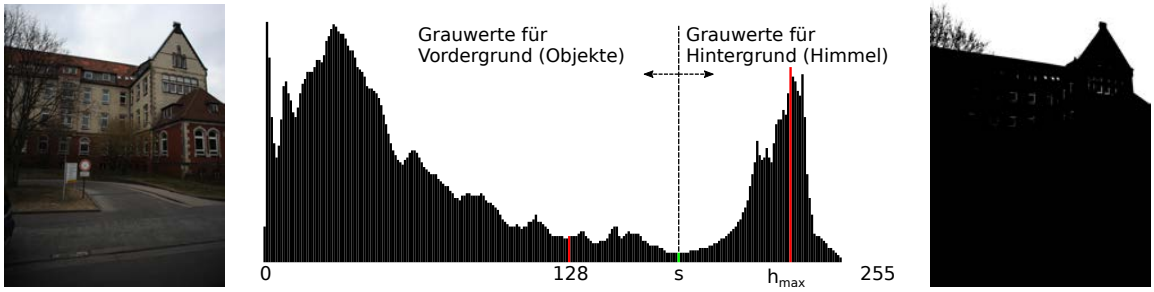


Abbildung 5.5: Histogrammbasierte Schwellwertermittlung (Mitte), sowie Anwendung des Ermittelten Schwellwertes auf Ausgangsbild (links) und resultierendes Binärbild (rechts).

5.3.1.2 Erzeugung von Silhouetten mittels α-Shapes

Im zweiten Schritt wird nun die Silhouette des Binärbildes als α-Shape Polygon des segmentierten Bereiches extrahiert. Da der segmentierte Bereich weitestgehend kompakt und geschlossen ist, kann dabei ein kleiner Radius (bspw. 2px) für die vom α-Shape Prozess genutzte Scheibe gewählt werden. Für das Beispiel aus Abbildung 5.5 (rechts) ergibt sich eine Silhouette wie in Abbildung 5.7 (links) dargestellt.

5.3.1.3 Entfernen problematischer Abschnitte

Wie Abbildung 5.7 (links) zeigt, verläuft das erzeugte Polygon auf der Grenze zwischen Vorder- und Hintergrund, jedoch ebenso entlang des Bildrandes. Bildpunkte am Bildrand trennen jedoch nicht Vorder- und Hintergrund und müssen daher von der Silhouette entfernt werden. Die wird erreicht indem das Polygon um einen Puffer um den Bildrand beschnitten wird. Das Resultat zeigt Abbildung 5.7 (Mitte).

Wie Abbildung 5.7 (Mitte) auch zeigt, können Vegetation wie Sträucher und Bäume, insbesondere Baumkronen, ebenfalls Teil der Silhouette sein. Dies ist jedoch problematisch, da die korrespondierenden Scanpunkte, für sehr dünne Äste, nicht zuverlässig ermittelt werden können. Dies erhöht potentiell die Anzahl an Fehlzuordnungen, beziehungsweise Ausreißern, signifikant. Um die durch Punktkorrespondenzen in Vegetation entstehenden Fehlzuordnungen möglichst gering zu halten, werden abschließend stark rauschende, gezackte Abschnitte aus der Silhouette entfernt. In einem zusätzlichen Homogenisierungsschritt wird abschließend versucht, neben den rauschenden Abschnitten auch glatte Zwischenstücke (Äste) zu entfernen und umgekehrt kleinere (u.U. rauschende) Details an Gebäudefassaden zu erhalten.

Zur Detektion der gezackten Bereiche des Silhouettenpfades wurde ein Fensterbasierter (Sliding Window) Ansatz gewählt. Dabei wird jeweils nur ein Teil (Fenster) des Pfades betrachtet und entschieden, ob ein gezackter Pfad vorliegt oder nicht. Um dabei nicht zu große Bereiche unmittelbar auszuschließen wird beim Vorliegen eines gezackten Bereichs lediglich die erste Hälfte des Fensters entsprechend markiert (gezackt oder nicht) und das Fenster anschließend auch nur um die halbe Fensterbreite

verschoben. Die Entscheidung, ob ein gezackter Pfad vorliegt basiert auf Analyse des vom Pfad abgeleiteten Ketten-Codes nach Freeman (1961). Dabei wird für je zwei aufeinander folgende Pfadpunkte die jeweilige Richtung in der Achter-Nachbarschaft bestimmt. Gezählt werden die aufeinander folgenden Richtungsänderungen, welche ebenfalls wieder über eine Achter-Nachbarschaft definiert werden können. Liegen innerhalb eines Fensters mehr als fünf unterschiedliche Richtungsänderungen vor, so wird, wie in Abbildung 5.6 gezeigt, die erste Fensterhälfte als *zu entfernen* markiert und das Fenster um eben diesen Bereich verschoben.

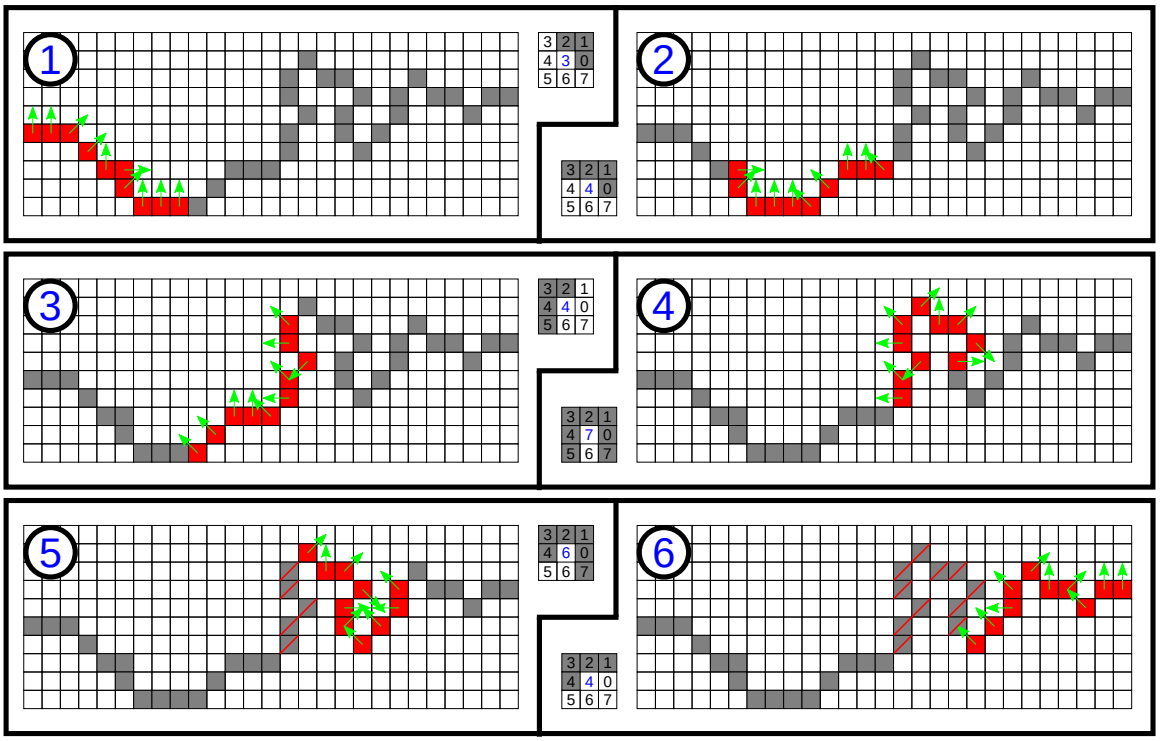


Abbildung 5.6: Veranschaulichung des entworfenen Sliding Window Verfahrens zur Identifikation gezackter Abschnitte einer Silhouette. Die jeweiligen Filtermasken (mittige Zahlenwürfel) veranschaulichen die identifizierten Richtungen und deren Summe (blaue Zahl in der Mitte) für das gezeigte Fenster.

Vor dem finalen Entfernen der markierten Bereiche wird noch eine Homogenisierung durchgeführt. Dabei werden kleine Abschnitte mit großen umliegenden Abschnitten vereint. Dies erlaubt das Entfernen von bspw. geraden Astabschnitten in Bäumen und das Behalten von kleineren Gebäudedetails, wie bspw. Schornsteinen. Im Homogenisierungsschritt wird jeweils der kürzeste Abschnitt identifiziert und mit den angrenzenden Abschnitten vereint. Liegen zwei gleich kleine Abschnitte vor, so wird der Abschnitt mit dem größeren Nachbarn bevorzugt. Der Schwellwert ab welcher Länge ein Abschnitt als *klein* eingestuft wird hängt stark von der gewählten Fenstergröße und natürlich von der Bildauflösung ab. Ein Wert vom vierfachen der genutzten Fensterbreite hat sich dabei als brauchbar herausgestellt. Das Ergebnis ist eine Silhouette mit vorwiegend geraden Abschnitten wie in Abbildung 5.7 (rechts) zu sehen.

Bei einer Mobile Mapping Messfahrt entlang einer Allee oder eines Waldes können Kamerabilder erfasst werden, welche zu einem Großteil Vegetation abbilden. Sollte der Anteil an rauschenden, gezackten Liniensegmenten 85% der Gesamtsilhouette übersteigen, so wird die gesamte Silhouette verworfen. Dies schließt das entsprechende Kamerabild, sowie das zugehörige Scanpunktbild aus der Bestimmung der Kameraparameter aus und vermeidet unnötige fehlerhafte Korrespondenzen.

Abbildung 5.7: α-Shape Polygon um Vordergrundsegmentierung (links), Beschränkung auf Skyline als Grenze zwischen Vorder- und Hintergrund (Mitte), sowie entfernte (rot) und gültige Bereiche (grün) (rechts).

5.3.2 Extraktion von Silhouetten aus Laserscandaten

Wie eingangs erläutert müssen aus den Laserscandaten zunächst Scanpunktbilder erzeugt werden. Dies erfolgt über die in Abschnitt 3.2.2 vorgestellte Scanstreifenbasierte Pufferstrategie. Abschließend erfolgt die Bestimmung der Silhouette, analog zu den Kamerabildern, mittels der α-Shape Technik im erzeugten Scanpunktbild. Abbildung 5.8 fasst die aufgezeigten Schritte wie gehabt in einem UML Aktivitätsdiagramm zusammen.

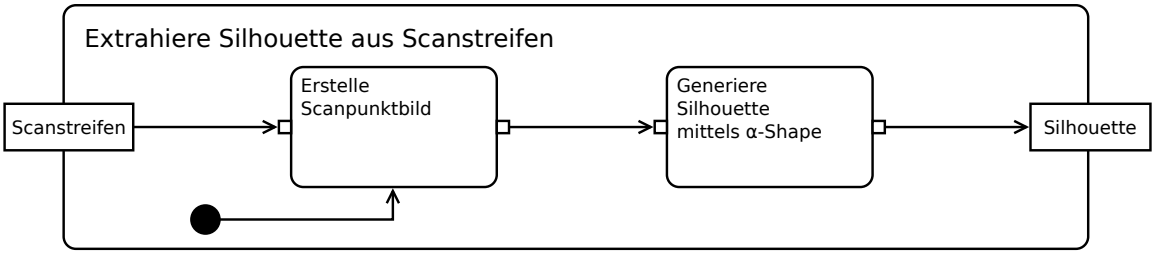


Abbildung 5.8: Extraktion von Silhouetten aus Laserscandaten mittels aus Scanstreifen erstellten Scanpunktbildern als UML Aktivitätsdiagramm.

5.3.2.1 Erstellung von Scanpunktbildern

Zur Extraktion von Silhouetten für Laserscandaten müssen diese zunächst in die gleiche Form wie die oben beschriebenen Kamerabilder gebracht werden. Dafür müssen alle Scanpunkte ermittelt werden, die für den jeweiligen Bildbereich relevant sind. Transformiert man anschließend alle in Frage kommenden Scanpunkte mittels der bekannten (groben) Kameraparamter in das entsprechende Bildkoordinatensystem, so erhält man ein Scanpunktbild. Da für die weitere Verarbeitung nicht nur die Lage der Scanpunkte im entsprechenden 2D Bildkoordinatensystem relevant sind, sondern darüber hinaus die ursprüngliche 3D Position des Scanpunktes (bspw. im Scanner Koordinatensystem zum Aufnahmezeitpunkt des Bildes) wird letztere in den zur Verfügung stehenden Farbkanälen gespeichert. Um einen Genauigkeitsverlust zu vermeiden wird hier eine Wortbreite von 32 Bit pro Farbkanal genutzt, dies ermöglicht das Speichern von drei Fließkommazahlen mit einfacher Genauigkeit in den drei RGB Farbkanälen. Ein Scanpunktbild enthält demnach genau zwei Informationen pro Punkt bzw. Pixel, zum einen die Lage im Bildkoordinatensystem (Pixelkoordinaten) und die Position des Scanpunktes im Scanner Koordinatensystem (kodiert in den drei Farbkanälen pro Pixel).

Die Erstellung der beschriebenen Scanpunktbilder erfolgt auf Basis der in Abschnitt 3.2.2 beschrieben Scanstreifenbasierten Pufferstrategie. Dabei werden die jeweiligen Scanpunkte ins Bildkoordinatensystem transformiert und wie beschrieben deren 3D Koordinaten in den Farbkanälen gespeichert. Erhalten zwei Scanpunkte die gleichen Bildkoordinaten, so wird der Scanpunkt, welcher sich dichter an der Kamera befindet, gewählt. Beispiele für während der Mobile Mapping Messfahrt erfassten Bilder und den dazu erstellten Scanpunktbildern zeigt Abbildung 5.9. Bild (1) und (2) zeigen das Kamera- und dessen Scanpunktbild der vorangegangenen Beispiele. Darüber hinaus zeigen Bilder (3) und (4) ein weiteres Paar aus Kamera- und Scanpunktbild. In den gezeigten Beispielen sind die jeweiligen Scanstreifen deutlich sichtbar. In Abhängigkeit der Punktdichte der Laserscandaten enthalten die erzeugten Scanpunktbilder in der Regel keine geschlossenen Flächen. Dies muss bei der folgenden Bestimmung der Silhouette berücksichtigt werden.

Abbildung 5.9: Beispiele für Fotos, aufgenommen während einer Mobile Mapping Messfahrt (1 und 3) und die dazugehörigen Scanpunktbilder als Binärbild (2 und 4).

5.3.2.2 Erzeugung von Silhouetten aus Scanpunktbildern mittels α-Shapes

Da die Laserscandaten keinerlei Punkte des Himmels beinhalten (vgl. Abb. 5.10 (links)), entfällt bei Scanpunktbildern der entsprechende Segmentierung in Vorder- und Hintergrund und es kann unmittelbar mit dem Bilden des α-Shape Polygons begonnen werden. In Abhängigkeit von der Geschwindigkeit, Fahrtrichtung und Objektdistanz des Erfassungsfahrzeugs sind die Scanpunktbilder weit weniger dicht als ihre Kamerabildpendants. Daher muss hier ein entsprechend größerer Scheibenradius genutzt werden. Genügte bei den Kamerabildern noch ein Radius von 2px, so wird bei den Scanpunktbildern der mindestens zehnfache Radius benötigt (abgeleitet aus den Punktabständen der sichtbaren Scanstreifen im Scanpunktbild). Wird ein zu kleiner Radius gewählt, so verläuft die Silhouette entlang einzelner Scanstreifen und nicht um das dadurch repräsentierte Objekt. Das gebildete Polygon umschließt dann unter Umständen lediglich einen oder einige wenige Scanstreifen und repräsentiert damit nicht die Silhouette der Vordergrundobjekte. In Abbildung 5.10 (Mitte) ist dieser Fall in abgeschwächter Form in der unteren linken Bildecke zu sehen. Ein großer Radius hingegen überspringt kleinere Details wie Ecken, welche wichtige Merkmale für ein robustes ICP Verfahren darstellen. Daher sollte der Radius immer so klein wie möglich und so groß wie nötig gewählt werden. Aufgrund der unterschiedlichen Punktdichten, sowohl innerhalb eines Scanpunktbildes, als auch über mehrere Bilder hinweg, fällt es schwer, einen fixen Radius für alle Bilder festzulegen. Deshalb wird lediglich ein minimaler α-Shape Scheibenradius von 25px festgelegt. Beinhaltet die gebildete Silhouette nicht hinreichend viele Punkte (hier wurde ein Schwellwert von mindestens 200 Punkten genutzt), so wird eine weitere Silhouette unter Verwendung eines Radius von 35px generiert. Beinhaltet die entstandene Silhouette nach wie vor ungenügend viele Punkte, so wird das Bildpaar verworfen. Ist die Silhouette hinreichend lang, wie in Abbildung 5.10 (Mitte) der Fall, kann die Identifikation der Korrespondenzen erfolgen.

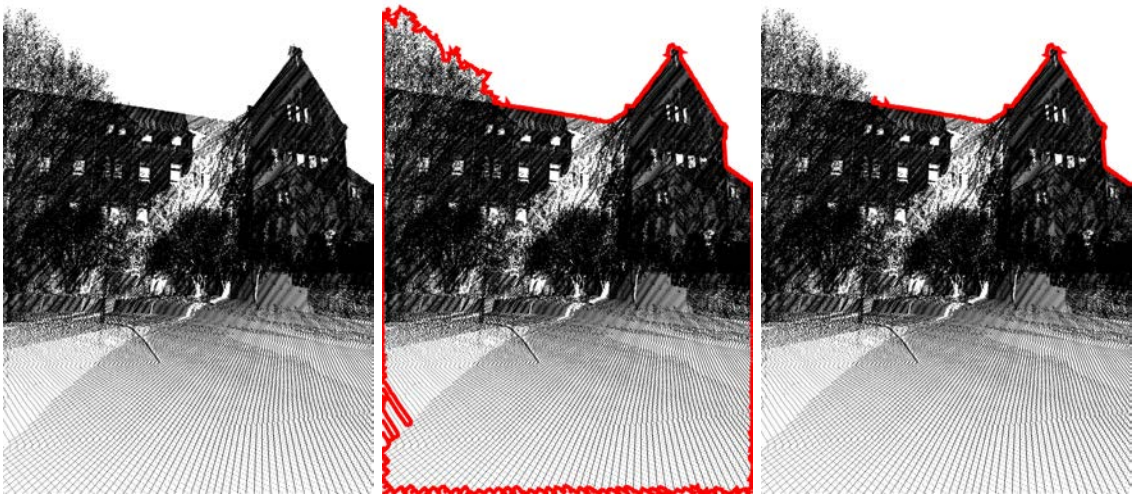


Abbildung 5.10: Scanpunktbild (links), α-shape Polygon um Scanpunkte (Mitte), Beschränkung auf Korrespondenzen aus Kamerabild Silhouette (rechts).

5.4 ICP-basierte Identifikation der Korrespondenzen

Wurden sowohl aus dem Kamerabild, als auch aus dem Scanpunktbild erfolgreich Silhouetten extrahiert, können diese zur Deckung gebracht und korrespondierende Punkte identifiziert werden. Die gefunden Paare gehen anschließend als Eingangsdaten in die Berechnung des Rückwärtsschnitts ein. Als erstes wird die Silhouette des Scanpunktbildes auf Abschnitte beschränkt, welche eine grobe Korrespondenz in der Silhouette des Kamerabildes besitzen. Da der Versatz der Scanpunkte unter Anderem von der Entfernung abhängt, wird die Silhouette des Scanpunktbildes zunächst in Abschnitte mit ähnlichen Entfernungen zur Kamera gruppiert, was anschließend zu robusteren Zuordnungsergebnissen führt. Abschließend werden die Korrespondenzpaare mittels ICP ermittelt. Um die Anzahl möglicher Fehlzuordnungen zu reduzieren, werden bei der Ermittlung der korrespondierenden Punkte deren Normalen berücksichtigt. Abbildung 5.11 fasst die beschriebenen Schritte als UML Aktivitätsdiagramm zusammen.

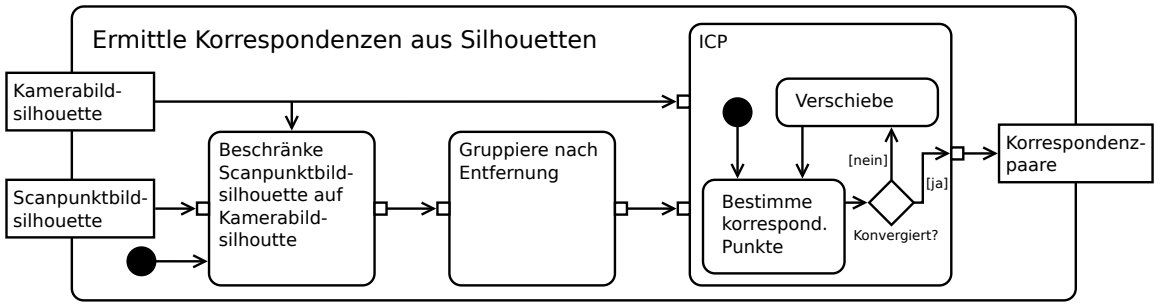


Abbildung 5.11: ICP-basierte Identifikation der Korrespondenzpaare aus Kamerabild- und Scanpunktsilhouetten als UML Aktivitätsdiagramm.

5.4.1 Beschränkung der Scanpunktbildsilhouette

Aufgrund der Beschaffenheit (sichtbares Muster der Scanstreifen) des Scanpunktbildes, weist dessen Silhouette viele Abschnitte auf, welche nicht genutzt werden können, da es keinerlei Korrespondenzen in der Silhouette des Kamerabildes gibt. Daher werden zunächst alle Abschnitte aus der Silhouette des Scanpunktbildes entfernt, die keine Korrespondenz in der Silhouette des Kamerabildes besitzen.

Das Ergebnis ist in Abbildung 5.10 (rechts) gezeigt. Keine Korrespondenz bedeutet dabei, dass der nächstgelegene Punkt in der Silhouette des Kamerabildes weiter als ein definierter Schwellwert MCD (Maximum Correspondance Distance) entfernt ist. Ein kleiner Wert für MCD schließt somit weit entfernte Korrespondenzen aus, was die Anzahl an potentiellen Fehlzuordnungen verringert. Andererseits können die gegebenen Kameraparameter so schlecht sein, dass durchaus weit von einander entfernte Punktkorrespondenzen existieren, daher sollte MCD auch nicht zu klein gewählt werden. Wie eingangs beschrieben, kann ein Versatz von bis zu 80px vorliegen (vgl. Abschnitt 5.1). Daher wurde ein MCD von 80 gewählt.

5.4.2 Gruppierung der Scanpunktdaten

Der auszugleichende Versatz zwischen Bild- und Scanpunkten hängt von den bei der Erstellung der Scanpunktbilder genutzten Kameraparameter ab. Da es sich bei den Scanpunktbildern um eine 3D nach 2D Projektion handelt, ist der entsprechende Versatz nicht konstant. Er hängt sowohl von der Entfernung, als auch von den Winkeln des Scanpunktes gegenüber der Kamera ab. Um möglichst korrekte Punktkorrespondenzen zu erhalten muss diesem Sachverhalt Rechnung getragen werden.

Grundsätzlich wird beim ICP Verfahren für jeden Punkt aus dem Quelldatensatz der nächstgelegene Punkt aus dem Zieldatensatz gewählt. Anschließend werden die Transformationen bzw. Verschiebungen aller Korrespondenzen gemittelt und an die Punkte des Quelldatensatzes angebracht. Dies wird iterativ wiederholt bis der Prozess konvergiert, d.h. die angebrachte Änderung konstant bleibt. Im Falle der Silhouetten führt dieses Vorgehen zu einer starren Verschiebung der gesamten Silhouette, was dem oben geschilderten Problem nicht gerecht wird. Alternativ könnten die einzelnen Korrespondenzen unabhängig von einander aus den jeweils nächstgelegenen Punkten gebildet werden. Dies erhöht jedoch die Anzahl an potentiellen Fehlzuordnungen signifikant und kann unter Umständen zu einem Divergieren des Prozesses führen.

Aus diesem Grund wird die Silhouette des Scanpunktbildes in zusammenhängende Punktgruppen unterteilt. Die Punkte innerhalb einer Gruppe besitzen dabei eine ähnliche Distanz zur Kamera und sind darüber hinaus kompakt, d.h. jeweils zwei aufeinander folgende Punkte sind auch Nachbarpunkte innerhalb der Silhouette. Dies unterteilt die Silhouette im Wesentlichen jeweils an den Tiefensprüngen. Abbildung 5.12 zeigt beispielhaft die entstandene Gruppierung der Silhouette. Zu kleine Gruppen werden abschließend entfernt, da diese die Gefahr von potentiellen Fehlzuordnungen erhöhen. Für jede dieser Gruppen werden nun separat die Punktkorrespondenzen mittels ICP ermittelt.

5.4.3 ICP unter Berücksichtigung der Punktnormalen

Wie Abbildung 5.13 (Mitte) zeigt, können einzelne Scanpunktgruppen jedoch falsche Korrespondenzen mittels ICP zugeordnet werden. Dies geschieht besonders häufig an schmalen Objekten wie dem in Abbildung 5.13 gezeigten Laternenmast. Da sich die Scanpunkte des linken und rechten Teil des Masts in unterschiedlichen Gruppen befindet und sich beide Gruppen gleich nah an der linken Seite der Silouette des Kamerabildes befinden, werden beide Gruppen der gleichen Seite des Mastens zugeordnet.

Um dies zu verhindern, werden die Punktnormalen der jeweiligen Silhouettenpunkte bei den ICP Zuordnungen ebenfalls berücksichtigt. Dabei werden keine Korrespondenzen von Punkten zugelassen, deren Normalen um mehr als 90 Grad von einander abweichen. Abbildung 5.13 (rechts) zeigt die folglich korrekte Zuordnung der Scanpunktgruppen zur Kamerabildsilhouette.

Abschließend werden alle Gruppen entfernt, welche ein schlechtes Korrespondenzverhältnis aufweisen. Das Korrespondenzverhältnis bildet das Verhältnis aus der Anzahl der Punkte einer Gruppe, zu denen ein korrespondierender Punkt innerhalb der MCD gefunden werden konnte und der gesamt Anzahl der Punkte einer Gruppe. Liegt dieses Verhältnis unterhalb von 50 % so wird die Gruppe verworfen, um potentielle Fehlzuordnungen auszuschließen.

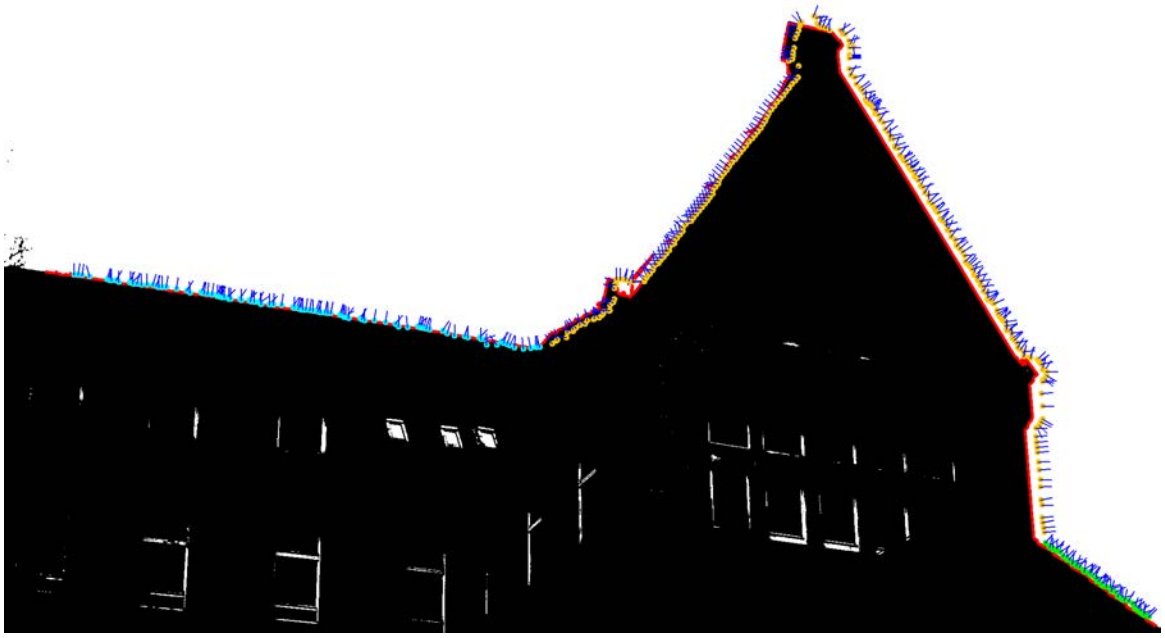


Abbildung 5.12: Distanzbasierte Scanpunktgruppen, Gruppe 1 (grün) bestehend aus 70 Punkten mit einer mittleren Distanz von 25m; Gruppe 2 (orange) bestehend aus 294 Punkten mit einer mittleren Distanz von 35m; Gruppe 3 (cyan) bestehend aus 122 Punkten mit einer mittleren Distanz von 45m; Punktnormalen (blau) und Kamerabild Silhouette (rot).

Alle verbliebenen Gruppen bzw. die zugehörigen korrespondierenden Paare aus Kamerabildpunkten und Scanpunkten bilden die Ausgangsdaten für die abschließende Berechnung des Rückwärtsschnitts zur Ermittlung der Kameraparameter.

5.5 Bestimmung der Kameraparameter mittels Rückwärtsschnitt

Zur finalen Bestimmung der Kameraparameter werden alle zuvor ermittelten Korrespondenzpaare (aller Kamera- und Scanpunktbilder) herangezogen. Zur Ermittlung der Kameraparameter wird folgendes funktionales Modell herangezogen:

$$\begin{aligned} x' &= x_0 + c\frac{r_{11}(X - X_0) + r_{12}(Y - Y_0) + r_{12}(Z - Z_0)}{r_{31}(X - X_0) + r_{32}(Y - Y_0) + r_{33}(Z - Z_0)}\Delta x' \\ y' &= y_0 + c\frac{r_{21}(X - X_0) + r_{22}(Y - Y_0) + r_{23}(Z - Z_0)}{r_{31}(X - X_0) + r_{32}(Y - Y_0) + r_{33}(Z - Z_0)}\Delta y' \end{aligned} \tag{5.1}$$

Die Pixel Koordinaten x' und y', welche aus der Silhouette der Kamerabilder extrahiert wurden, werden dabei als Beobachtungen herangezogen. Wohingegen die Rotation r_{ij} und der Kameraursprung X_0, Y_0, Z_0 die Unbekannten darstellen. Wie eingangs beschrieben, liegt jedoch ein zeitlicher Versatz Δt zwischen den Zeitstempeln der Laserscan- und Kamerabildbaten vor. Eine Anpassung des Bild Zeitstempels wirkt sich in einer virtuellen Verschiebung der Kameraposition entlang der Trajektorie des Erfassungsfahrtzeugs aus. Um nun den zeitlichen Versatz Δt in das funktionale Modell zu in-

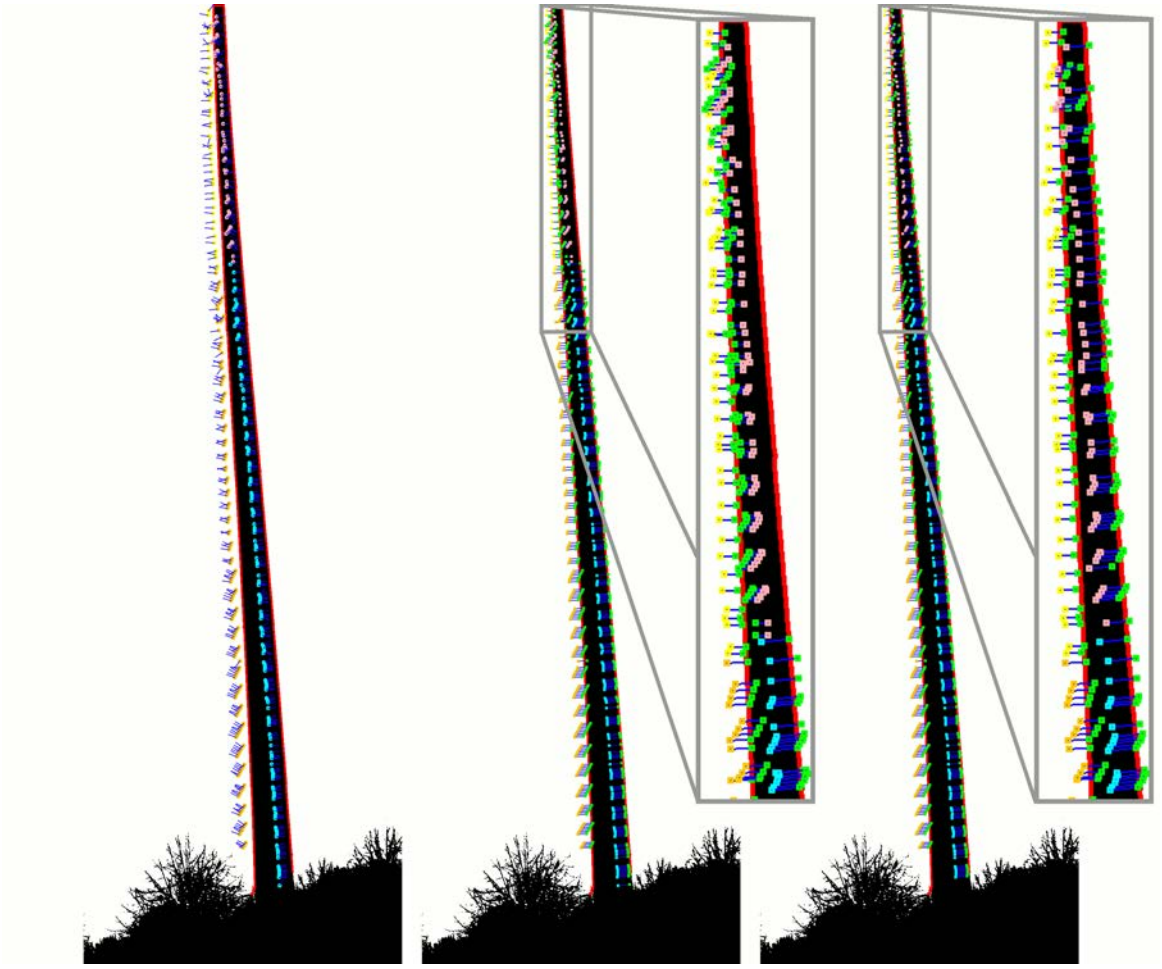


Abbildung 5.13: Vier Distanz-basierte Scanpunkt-Gruppen an einer Stange (links); ICP Ergebnis ohne Berücksichtigung der Punkt-Normalen: Gruppen `orange, gelb, cyan` *korrekt zugeordnet, Gruppe* `pink` *wird jedoch der falschen Seite zugeordnet (Mitte); ICP Ergebnis unter Berücksichtigung der Punkt-Normalen: alle Gruppen korrekt zugeordnet (rechts).* `Grüne` *Punkte stellen jeweils transformierten Punkte dar.*

tegrieren, muss die daraus resultierende lokale Verschiebung $\Delta X_t, \Delta Y_t, \Delta Z_t$ an die Kameraposition angebracht werden:

$$\begin{aligned} x' &= x_0 + c\frac{r_{11}(X - X_0 - \Delta X_t) + r_{12}(Y - Y_0 - \Delta Y_t) + r_{12}(Z - Z_0 - \Delta Z_t)}{r_{31}(X - X_0 - \Delta X_t) + r_{32}(Y - Y_0 - \Delta Y_t) + r_{33}(Z - Z_0 - \Delta Z_t)}\Delta x' \\ y' &= y_0 + c\frac{r_{21}(X - X_0 - \Delta X_t) + r_{22}(Y - Y_0 - \Delta Y_t) + r_{23}(Z - Z_0 - \Delta Z_t)}{r_{31}(X - X_0 - \Delta X_t) + r_{32}(Y - Y_0 - \Delta Y_t) + r_{33}(Z - Z_0 - \Delta Z_t)}\Delta y' \end{aligned} \tag{5.2}$$

$$mit \begin{pmatrix} \Delta X_t \\ \Delta Y_t \\ \Delta Z_t \end{pmatrix} = \begin{pmatrix} X_{t_0+\Delta t} \\ Y_{t_0+\Delta t} \\ Z_{t_0+\Delta t} \end{pmatrix} - \begin{pmatrix} X_{t_0} \\ Y_{t_0} \\ Z_{t_0} \end{pmatrix} \tag{5.3}$$

Dabei repräsentiert $\Delta X_t, \Delta Y_t, \Delta Z_t$ den Verschiebungsvektor zwischen den Trajektoriepunkten X_{t_0}, Y_{t_0}, Z_{t_0} zum Zeitpunkt t_0 und $X_{t_0+\Delta t}, Y_{t_0+\Delta t}, Z_{t_0+\Delta t}$ zum Zeitpunkt $t_0 + \Delta t$.

Je nach Anzahl der Bilder können mehrere Hunderttausend Beobachtungen in die Berechnung einfließen. Dies macht das separate Aufstellen der Design Matrix A und deren Transponierte A^t sehr

speicherintensiv, da für jede Beobachtung zwei Zeilen bzw. Spalten benötigt werden. Unter der Annahme, dass die Gewichtung aller Beobachtungen unkorreliert ist, kann die Matrix A^tA, deren Größe nicht mehr von der Anzahl der Beobachtungen abhängt, sondern lediglich von der Anzahl der Unbekannten, direkt aufgestellt werden.

Obgleich bei der Bestimmung der Korrespondenzen darauf geachtet wurde, Fehlzuordnungen zu vermeiden, muss nach wie vor von einer gewissen Anzahl an Ausreißern ausgegangen werden. Iterative Verfahren, welche bei jeder Iteration den größten Ausreißer eliminieren, wie das Data-Snooping nach Baarda (1968) bringen bei der gegebenen Menge an Beobachtungen erhebliche Ressourcen- und Laufzeitnachteile mit sich. Hier wird zum einen die komplette Design Matrix A benötigt, was sich wie erwähnt Nachteilig auf den Speicherbedarf auswirkt, zum anderen wird in der Regel bei jeder Iteration nur ein einziger Ausreißer entfernt, was sich wiederum extrem nachteilig auf die Laufzeit auswirkt. Daher wurde hier ein RANSAC (RANdom SAmple Consensus) basierter Ansatz nach Fischler und Bolles (1981) gewählt.

Der gewählte Ansatz bestimmt dabei auf Basis einer zufälligen Stichprobe an Beobachtungen das gesuchte Modell (die gesuchten Kameraparameter). Für die Ermittlung eines zuverlässigen Modells sollte die gewählte Stichprobe jedoch gewissen Randbedingungen genügen (vgl. Abschnitt 5.5.1 Anschließend wird die Menge der Punkte (Consensus) ermittelt, die zum Modell der Stichprobe passen. Der LO-RANSAC (local optimized) Variante nach Chum u. a. (2004) folgend werden mittels dieser Consensus Menge abermals die Modellparameter bestimmt und abschließend wieder die zugehörige Consensus Menge ermittelt. Dies wird entsprechend einer vorher bestimmten Anzahl an Iterationen oft wiederholt, wobei jedes neu gefundene Modell mit dem bisher besten Modell verglichen wird. Das nach Ablauf aller Iterationen beste Modell repräsentiert abschließend das Ergebnis.

Das beschriebene Verfahren bietet diverse Steuerparameter, mit denen Anpassungen an die gegebenen Daten und Laufzeitumgebungen möglich sind. Großen Einfluss auf die Qualität des Modells hat die initiale Auswahl der Stichprobe, welche daher nicht gänzlich zufällig gezogen werden, sondern gewissen Bedingungen genügen sollte. Ein weiterer für die Qualität resultierender Modelle relevanter Parameter stellt die Metrik dar, welche für die Bewertung eines gefunden Modells herangezogen wird. Der für die Laufzeit des Verfahrens maßgebende Parameter stellt die Anzahl der durchgeführten Iterationen dar. Einfluss und Optimierung der aufgezeigten Parameter werden in den folgenden Abschnitten diskutiert.

5.5.1 Wahl der Stichprobe

Eine wichtiger Steuerparameter des Verfahrens ist die Auswahl der Stichprobe. Dabei wird eine minimale Anzahl an Beobachtungen gewählt. Im Fall der sieben Unbekannten sind dies vier Korrespondenzpaare (ein Korrespondenzpaar steuert jeweils zwei Beobachtungen bei, eine für x und und eine für y). Gänzlich zufällig gewählte Korrespondenzpaare sind unter Umständen jedoch für eine zuverlässige Parameterbestimmung ungeeignet. Liegen beispielsweise die jeweiligen Scanpunkte auf einer Geraden oder konzentrieren sich die Kamerabildpunkte auf einen kleinen Bereich, können möglicherweise keine verlässlichen Parameter bestimmt werden. Daher werden für eine gewählte Stichprobe folgende Randbedingungen gefordert:

Distanz zur Kamera: *Mindestens ein Scanpunkt der Stichprobe muss nahe an der Kamera liegen.*

Entfernungsverteilung: *Alle gewählten Scanpunkte müssen einen gewissen (räumlichen) Mindestabstand von einander besitzen.*

Kamerabildpunktverteilung: *Alle gewählten Kamerabildpunkte müssen einen gewissen Mindestabstand von einander besitzen.*

Geschwindigkeitsverteilung: *Jeweils zwei der gewählten Beobachtungen müssen bei geringen, als auch bei höheren Geschwindigkeiten erfasst worden sein.*

Die gezeigten Bedingungen tragen der Tatsache Rechnung, dass die Lage der projizierten Scanpunkte von der Entfernung und dem Winkel zur Kamera abhängt. Ersteres wird über die Bedingungen

Distanz zur Kamera und *Entfernungsverteilung* adressiert, wohingegen die *Kamerabildpunktverteilung* letztes berücksichtigt. Der beschriebene Zeitversatz wirkt sich in Abhängigkeit der Geschwindigkeit des Erfassungsfahrzeugs unterschiedlich stark aus, was über die Bedingung *Geschwindigkeitsverteilung* abgedeckt wird. Erfüllt eine Stichprobe die geforderten Bedingungen nicht, so wird sie verworfen und eine weitere Stichprobe gezogen.

5.5.2 Anzahl an Iterationen

Die Anzahl der durchgeführten Iterationen hat großen Einfluss auf die Qualität der final bestimmten Modellparameter und ist maßgebend für die Laufzeit des Verfahrens. Ist die Anzahl zu gering, ist die Wahrscheinlichkeit hoch, dass nicht das beste Modell gefunden wurde, da beispielsweise in allen herangezogenen Stichproben Ausreißer enthalten waren. Eine hohe Anzahl an Iterationen schlägt sich hingegen in einer hohen Laufzeit wieder. Daher sollte die Anzahl an Iterationen etwa so groß gewählt werden, dass die Wahrscheinlichkeit, dass mindestens eine ausreißerfreie Stichprobe gewählt wurde genügend hoch, bspw. $\geq 99\%$, ist. Bei vier zu wählenden Korrespondenzpaaren (zur Bestimmung der sieben Unbekannten) und einem Ausreißer Verhältnis von ϵ ergibt sich die Wahrscheinlichkeit $P_{mindEinAusreisser}$, dass bei i Iterationen immer mindestens ein Ausreißer gewählt wird nach Gleichung 5.4.

$$P_{mindEinAusreisser} = (1 - (1 - \epsilon)^i)^4 \tag{5.4}$$

Soll nun die Wahrscheinlichkeit, dass mindestens einmal eine ausreißerfreie Stichprobe gezogen wurde bspw. $99\% => P_{mindEinAusreißer} = 0.01$ betragen, so ergibt sich eine Mindestanzahl an Iterationen nach Gleichung 5.5.

$$i_{min} = \frac{log(P_{mindEinAusreisser})}{log(1 - (1 - \epsilon)^4)} \tag{5.5}$$

Geht man von einem pessimistischen Ausreißerverhältnis von bspw. 57% ($\epsilon = 0.57$) aus, ergibt sich eine Mindestanzahl an Iterationen i_{min} von:

$$i_{min} = \left\lceil \frac{log(0.01)}{log(1 - (1 - 0.57)^4)} \right\rceil = 42 \tag{5.6}$$

Wie Gleichungen 5.5 und 5.6 zeigen, hängt die Anzahl der durchzuführenden Iterationen lediglich vom gewählten Ausreißerverhältnis ab. Somit steht die benötigte Laufzeit im direkten Zusammenhang mit der Qualität der Beobachtungen. Dies wurde wie bereits gezeigt bei der Ermittlung der Korrespondenzpaare berücksichtigt und nur jene gewählt welche potentiell sichere Korrespondenzen darstellen.

5.5.3 Bewertung der gefundenen Modelle

Zur Bewertung der gefunden Modelle können diverse Metriken herangezogen werden. Üblich sind bspw. die Größe der Consensus Menge oder die Standardabweichung der zur Consensus Menge gehörenden Beobachtungen im Bezug auf das gefundene Modell. In der vorliegenden Arbeit wurden beide Metriken kombiniert und die Standardabweichung mit der Größe der Consensus Menge zusätzlich gewichtet. Somit werden auch Modelle akzeptiert, welche eine schlechtere Standardabweichung besitzen, jedoch mehr Beobachtungen einschließen.

5.6 Ergebnisse

Das beschriebene Verfahren zur Feinkalibrierung der Kameraparameter (X_0, Y_0, Z_0) in Metern, sowie (ρ, ϕ, κ) in ° und Δt in Millisekunden ergab für zwei Beispielprojekte die in Tabelle 5.1 und Tabelle 5.2 aufgelisteten Resultate. Ein Projekt stellt dabei eine unabhängige Mobile Mapping Messfahrt dar.

Projekt A		Kamera III	Kamera IV
Position (X_0, Y_0, Z_0)	Riegl	$(0.424, -0.396, 0.082)$	$(0.347, 0.425, 0.011)$
	Fein	$(0.256, -0.481, 0.056)$	$(0.284, 0.399, 0.043)$
	Δ	$(-0.168, -0.085, -0.026)$	$(-0.063, -0.026, 0.032)$
Orientierung (ρ, ϕ, κ)	Riegl	$(-100.045°, -13.671°, 92.136°)$	$(100.193°, -0.977°, -90.193°)$
	Fein	$(-100.015°, -14.135°, 91.971°)$	$(100.223°, -1.270°, -90.163°)$
	Δ	$(0.029°, -0.463°, -0.164°)$	$(0.030°, -0.293°, 0.030°)$
Zeitversatz Δt	Riegl	$0ms$	$0ms$
	Fein	$3.175ms$	$-5.231ms$
	Δ	$3.175ms$	$-5.231ms$

Tabelle 5.1: Ermittelte Kameraparameter für Beispiel-Projekt A.

Projekt B		Kamera III	Kamera IV
Position (X_0, Y_0, Z_0)	Riegl	$(0.424, -0.396, 0.082)$	$(0.347, 0.425, 0.011)$
	Fein	$(0.350, -0.447, 0.061)$	$(0.341, 0.434., 0.029)$
	Δ	$(-0.074, -0.051, -0.021)$	$(-0.006, -0.009, 0.018)$
Orientierung (ρ, ϕ, κ)	Riegl	$(-100.045°, -13.671°, 92.136°)$	$(100.193°, -0.977°, -90.193°)$
	Fein	$(-99.999°, -14.300°, 92.059°)$	$(100.404°, -1.299°, -90.244°)$
	Δ	$(0.046°, -0.629°, -0.077°)$	$(0.211°, -0.321°, -0.051°)$
Zeitversatz Δt	Riegl	$0ms$	$0ms$
	Fein	$-12.702ms$	$-9.881ms$
	Δ	$-12.702ms$	$-9.881ms$

Tabelle 5.2: Ermittelte Kameraparameter für Beispiel-Projekt B.

Bei näherer Betrachtung beträgt die Änderung der Orientierung gegenüber der groben Kalibrierung weniger als ein Grad. Im Gegensatz dazu weist die Position eine Verschiebung von mehreren Zentimetern gegenüber den Initialwerten auf, was als durchaus signifikant bezeichnet werden kann. Der interessanteste und für die Qualität der Kalibrierung ausschlaggebenste Wert stellt jedoch der ermittelte Zeitversatz Δt der jeweiligen Kamera dar. Dieser ist nicht nur bei den beiden untersuchten Kameras für die jeweilige Messfahrt unterschiedlich, sondern auch über die beiden betrachteten Messfahrten hinweg. Der ermittelte Zeitversatz reicht dabei von -12ms bis zu +3ms. Was eine wie in Abschnitt 5.1 gezeigte Abweichung der ermittelten Pixelkoordinaten von über 80 Pixeln zur Folge haben kann.

Neben einer rein visuellen Beurteilung der Resultate, lassen sich die Ergebnisse auch anhand der Residuen, als Abstand zwischen den Soll- und Ist-Positionen der transformierten Scanpunkte, bewerten. Abbildungen 5.14 und 5.15 zeigen die Histogramme der Residuen vor und nach der Feinkalibrierung für die Beispielprojekte A und B. Die Mediane der Abstände der Soll- und Ist-Positionen ist deutlich von 13-17 auf 3-5 gesunken.

Abbildungen 5.16 und 5.17 zeigen die Auswirkung der verbesserten Kamerakalibrierung bei der Bestimmung der Farbwerte der Scanpunkte anhand unterschiedlicher Geschwindigkeiten. Die rot eingefärbten Scanpunkte wurden mit den manuellen Parametern (ohne Berücksichtigung des Zeitversatzes) projiziert, wohingegen die blau und grün eingefärbten Scanpunkte mittels der vorgestellten automatisierten Feinkalibrierung in das Kamerabild transformiert wurden. Um den Einfluss des Zeitversatzes deutlich zu machen, wurde im Falle der blauen Scanpunkte der ermittelte Zeitversatz

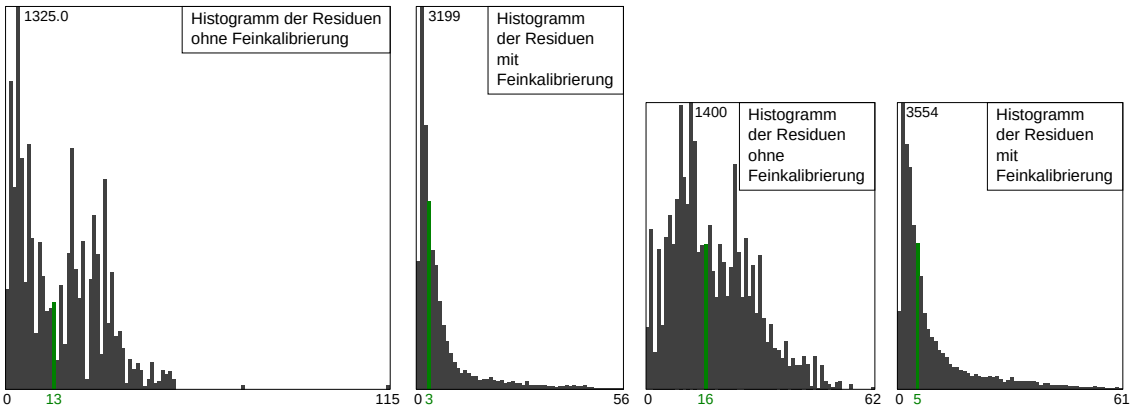


Abbildung 5.14: Histogramme der Residuen (Pixel-Abstand von Soll- und Ist-Koordinaten) für Projekt A (Kamera III - links; Kamera IV - rechts): Vor der Feinkalibrierung mit einem Median von 13 und 16 Pixeln und nach der Feinkalibrierung mit einem Median von 3 und 5 Pixeln.

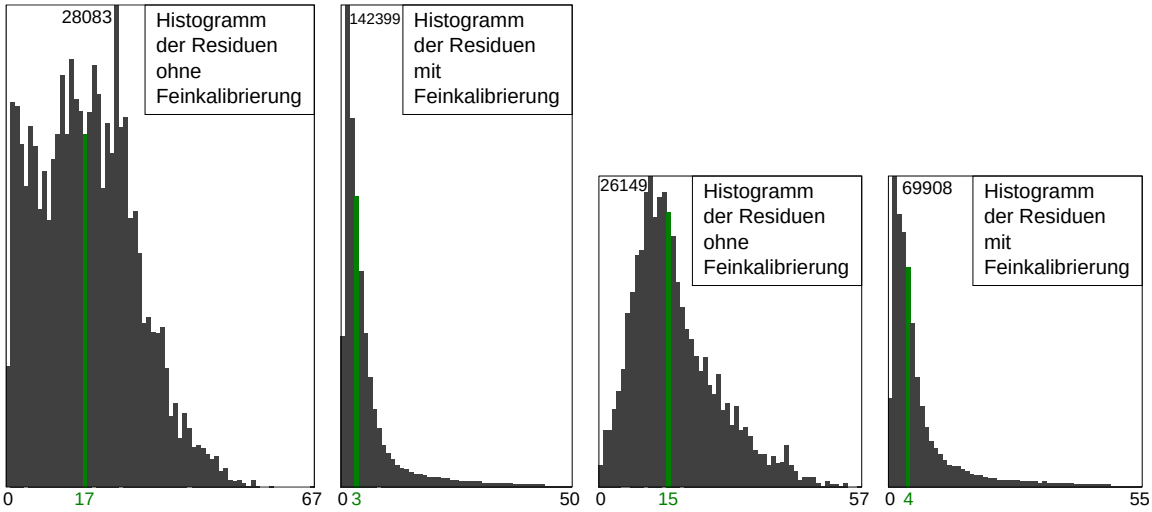


Abbildung 5.15: Histogramme der Residuen (Pixel-Abstand von Soll- und Ist-Koordinaten) für Projekt B (Kamera III - links; Kamera IV - rechts): Vor der Feinkalibrierung mit einem Median von 17 und 15 Pixeln und nach der Feinkalibrierung mit einem Median von 3 und 4 Pixeln.

ignoriert (vgl. Abbildung 5.2). Im Falle der grün dargestellten Scanpunkte wurde der ermittelte Zeitversatz entsprechend berücksichtigt, was somit das finale Ergebnis repräsentiert. Abbildung 5.16 (rot, links) zeigt einen nur leichten Versatz, welcher mittels des vorgestellten Verfahrens erfolgreich korrigiert werden konnte (grün, rechts). Die Geschwindigkeit des Erfassungsfahrzeugs betrug in diesem Fall etwa 30 km/h, was lediglich eine marginale Auswirkung des ermittelten Zeitversatzes (blau, Mitte) zeigt. Im Gegensatz dazu betrug die Geschwindigkeit bei der Aufnahme der Daten aus Abbildung (Abb. 5.17) etwa 70 km/h. Sowohl die manuellen (rot, links), als auch die automatisiert ermittelten Parameter ohne Berücksichtigung des Zeitversatzes (blau, Mitte) zeigen einen signifikanten Versatz auf. Erst die vorgestellte Verfahren unter Einschluss des Zeitversatzes führt zu Parametern welche keinen sichtbaren Versatz mehr erkennen lassen (grün, rechts).

5.7 Verbesserungspotential und Probleme

Aufgrund der Komplexität des aufgezeigten Verfahrens gibt es auch hier Potential für Verbesserungen, sowie mögliche Problemstellen.

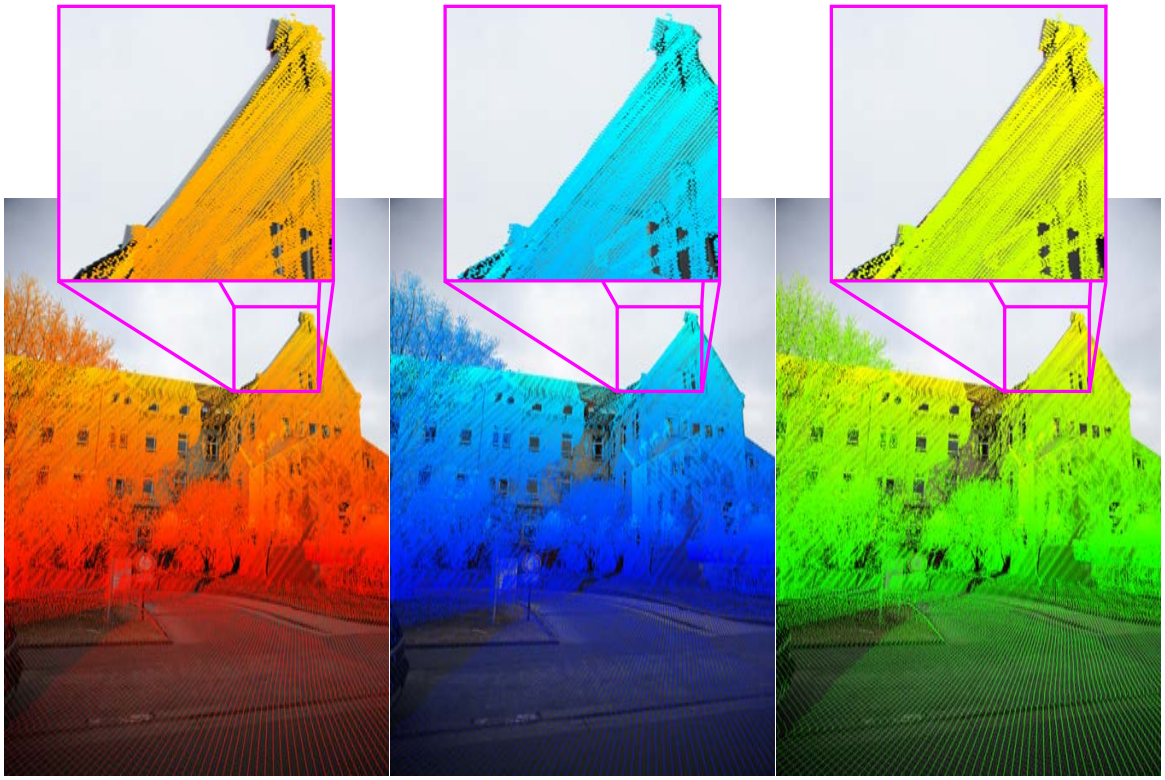


Abbildung 5.16: Marginaler Einfluss des ermittelten Zeitversatzes bei geringen Geschwindigkeiten: Transformierte Scanpunkte unter Verwendung der original Kalibrierparameter (rot, links), der Feinkalibrierparameter ohne Zeitversatz (blau, Mitte) und der Feinkalibrierparameter mit Zeitversatz (grün, rechts) bei einer Geschwindigkeit von 30 km/h.

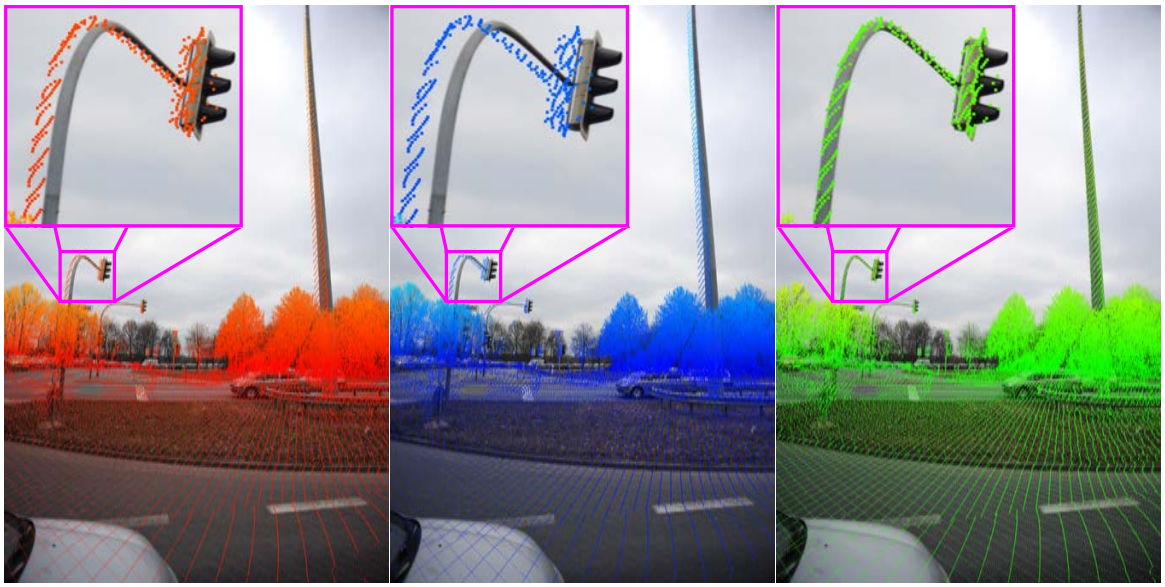


Abbildung 5.17: Deutlicher Einfluss des ermittelten Zeitversatzes bei höheren Geschwindigkeiten: Transformierte Scanpunkte unter Verwendung der original Kalibrierparameter (rot, links), der Feinkalibrierparameter ohne Zeitversatz (blau, Mitte) und der Feinkalibrierparameter mit Zeitversatz (grün, rechts) bei einer Geschwindigkeit von 70 km/h.

5.7.1 Laufzeiten

Ein kritischer Aspekt ist die Laufzeit des Verfahrens. Die Laufzeiten der Hauptverarbeitungsschritte, *Extraktion der Silhouetten*, *Ermittlung der Korrespondenzen* und die *Bestimmung der Parameter mittels Rückwärtsschnitt* sind dabei höchst unterschiedlich. Die zuvor gezeigten Optimierungen erlauben eine sehr schnelle Ausführung der finalen Parameterberechnung mittels des Rückwärtsschnitts. Dieser Schritt benötigt, wie gezeigt, unabhängig von der Anzahl der eingehenden Beobachtungen, nur wenige Sekunden auf einem aktuellen System. Die Laufzeit der Ermittlung der Korrespondenzen hängt im Wesentlichen von der Anzahl der extrahierten Silhouetten und von der Auflösung der Kamerabilder ab. Eine höhere Auflösung der Kamerabilder resultiert dabei in detaillierte Silhouetten. Daher gilt, je mehr Silhouetten extrahiert wurden und je höher die Auflösung, desto rechenintensiver ist die Ermittlung der Korrespondenzen. Die Laufzeiten für diesen Schritt lagen für ein Mobile Mapping Projekt mit jeweils etwa 1000 Silhouetten aus Kamerabildern und Laserscanmessungen im einstelligen Minutenbereich. Der erste Schritt, die Extraktion der Silhouetten, hat im Gegensatz zu den beiden darauf folgenden Schritten eine wesentlich höhere Laufzeit. Die Laufzeit der Extraktion der Silhouetten aus den Kamerabildern hängt dabei zum Großteil von den Ladezeiten der Kamerabilder ab und lässt sich daher kaum optimieren. Hier muss pro Kamerabild mit einer Laufzeit von mehreren Sekunden gerechnet werden, was bei mehreren Tausend Bildern zu einer Gesamtlaufzeit im Stundenbereich führt. Ähnlich verhält es sich bei der Extraktion der Silhouetten aus den Laserscandaten. Hier stellen wieder die Ladezeiten der Laserscandaten, obgleich optimiert durch die verwendete Pufferstrategie, bei der Erstellung der Scanpunktbilder den maßgeblichen Flaschenhals dar. Die Laufzeit liegt auch hier wie bei den Kamerabildern wieder im Stundenbereich.

Soll die Laufzeit des gesamten Verfahrens auf interaktive Bereiche (wenige Sekunden) reduziert werden, so muss die Extraktion der Silhouetten signifikant beschleunigt werden. Da, wie geschildert, die Laufzeit stark von den jeweiligen Ladezeiten und damit vom Datenumfang abhängt, ist eine Beschleunigung lediglich über eine Reduktion des Datenumfangs erreichbar. Eine Möglichkeit dies zu erreichen, stellt die Verringerung der Anzahl der verwendeten Kamerabilder dar. Das beschriebene Verfahren sah die Verwendung aller zur Verfügung stehenden Kamerabilder vor. Für eine zuverlässige Bestimmung der Parameter sind jedoch nur wenige Korrespondenzpaare notwendig. Hier könnten, ähnlich zur Wahl der Stichprobe im Rahmen des genutzten RANSAC Verfahrens, stichprobenartig Bilder unter der Einhaltung gewisser Randbedingungen ausgewählt werden. Die Extraktion der Silhouetten würde sich dann lediglich auf diese Untermenge an Bildern beschränken, was eine, je nach Umfang der Stichprobe, erhebliche Laufzeiteinsparung ergibt. Mit einer Reduktion der Bildanzahl reduziert sich ebenfalls die Anzahl an erzeugten Scanpunktbildern, sowie die Anzahl an zu verarbeitenden Silhouetten bei der Ermittlung der Korrespondenzen.

5.7.2 Robustheit des Verfahrens und Qualität der Ergebnisse

Die Extraktion der Silhouetten aus den Kamerabildern nutzt zur Trennung von Hinter- und Vordergrundobjekten einen Grauwertschwellwert. Dies basiert auf der Annahme, dass der Hintergrund immer heller ist als etwaige Vordergrundobjekte. Obgleich dies in der Regel der Fall ist, gibt es natürlich auch Ausnahmen, bzw. Problemsituationen, bspw. Gebäude mit sehr heller Fassade oder dunkle Gewitterwolken. Die genannten Situationen führen dann zu potentiell falschen Silhouetten und zu möglichen Fehlzuordnungen und sollten daher auf geeignete Art und Weise adressiert werden.

Ein weiterer potentiell problematischer Aspekt stellt der verwendete α-Shape Scheibenradius bei der Extraktion der Scanpunktbildsilhouette dar. Dieser sollte in Abhängigkeit von der vorliegenden Punktdichte ermittelt, anstatt fest vorgegeben werden. Hier muss jedoch die Vergleichbarkeit der damit erstellten Silhouetten gewährleistet werden, was bei unterschiedlichen Radien unter Umständen problematisch sein könnte.

Wie gezeigt sind die Ermittlung der Korrespondenzen und die Berechnung der gesuchten Parameter als eigenständige Schritte umgesetzt. Dies führt jedoch dazu, dass die im Rahmen der Korrespondenzsuche mittels ICP ausgeführten Transformationen lediglich im 2D Bildraum stattfinden und keine erneute Projektion der Scanpunkte anhand temporärer Kameraparameter erfolgt. Die ein-

geführte Entfernungsgruppierung der Scanpunktsilhouette adressiert dieses Problem. Da der vorliegende Versatz der Scanpunkte nicht ausschließlich von der Entfernung des Scanpunktes zur Kamera abhängt, löst die genutzte Entfernungsgruppierung das Problem nicht in Gänze. Für die Ermittlung von Kameraparametern während der Korrespondenzermittlung muss die Trennung der beiden Schritte aufgehoben bzw. zumindest aufgeweicht werden. Dies führt jedoch unter Umständen dazu, dass pro Silhouette Kameraparameter ermittelt und zur Transformation genutzt werden, welche lokal ein möglicherweise korrektes Ergebnis liefern, jedoch global falsche Zuordnungen erzeugt, welche in der globalen Ermittlung der Kameraparameter über alle bzw. mehrere Bilder hinweg zu falschen Ergebnissen führen. Darüber hinaus gestaltet sich die durchzuführende Transformation dadurch komplexer und damit in der Regel fehleranfälliger und weniger robust, was zum einen die Laufzeit erhöht und zum anderen zu einem Divergieren des ICP Prozesses führen kann. Insgesamt gesehen sollte dies jedoch zu qualitativ besseren und genaueren Zuordnungen von Kamera- und Scanpunkten führen und damit zu qualitativ besseren Kameraparametern.

6 Farbbestimmung

Die im vorherigen Kapitel aufgezeigte Ermittlung der Kameraparameter Position, Orientierung und Zeitversatz ermöglichen eine möglichst genaue Zuordnung von Farben zu Scanpunkten. Die bestimmten Farbwerte können anschließend im Rahmen einer Visualisierung der Laserscandaten verwendet werden. Darüber hinaus stellen die besagten Farben zusätzliche Eigenschaften der Scanpunkte dar, welche beispielsweise in komplexeren (verglichen mit den in Abschnitt 4.4 gezeigten) Segmentierungs- und Klassifikationsverfahren genutzt werden können.

Das hier vorgestellte Verfahren zur Bestimmung der Farbe der Scanpunkte besteht aus vier Schritten. Jeder Scanpunkt kann in der Regel in mehrere Kamerabilder projiziert werden. Daher werden zunächst im Rahmen eines Vorverarbeitungsschritts alle in Frage kommenden Farbwerte für jeden Scanpunkt aus den Kamerabildern extrahiert. Anschließend wird für jeden Scanpunkt und Farbwert die Verdeckungssituation analysiert, um mögliche, durch potentielle Verdecker verursachte, Fehlzuordnungen von Farbwerten auszuschließen. Zur effizienten Analyse der Verdeckungssituation jedes Scanpunktes wurde eine Ballpuffer Datenstruktur entwickelt. Unabhängig von der Verdeckungsanalyse entstehen auf diesem Wege immer Farbzuordnungen von räumlich benachbarten Scanpunkten, welche aus unterschiedlichen Kamerabildern extrahiert wurden. In Abhängigkeit der Beleuchtungssituation und der Lage des projizierten Scanpunktes im jeweiligen Kamerabild kann dies zu stark unterschiedlichen Helligkeiten der extrahierten Farben und somit zu deutlich sichtbaren Farb- bzw. Helligkeitssprüngen führen. Dieses Problem wird durch eine Farbanpassung adressiert, welche die Helligkeit der Farbwerte benachbarter Scanpunkte anpasst und somit die sichtbaren Helligkeitssprünge ausgleicht. Je nach Anordnung der Kameras des Mobile Mapping Systems und etwaigen Verdeckungssituationen kann es vorkommen, dass keine Farbe für einen Scanpunkt bestimmt werden kann. Um dennoch einen Farbwert für jeden Scanpunkt zur Verfügung zu stellen, wird im letzten Schritt ein Farbwert für die besagten Scanpunkte synthetisiert.

Einen Überblick zur Einordnung in das Gesamtframework gibt Abbildung 6.1. Die eingeführte Ballpuffer Datenstruktur ist dabei dem Kernmodul Datenstrukturen zuzuordnen, wohingegen die Extraktion der möglichen Farbwerte zum Basismodul Vorverarbeitung zählt. Die Komponenten Verdeckungsanalyse, Farbanpassung und -synthese sind im Verarbeitungsmodul Farbbestimmung zusammengefasst.

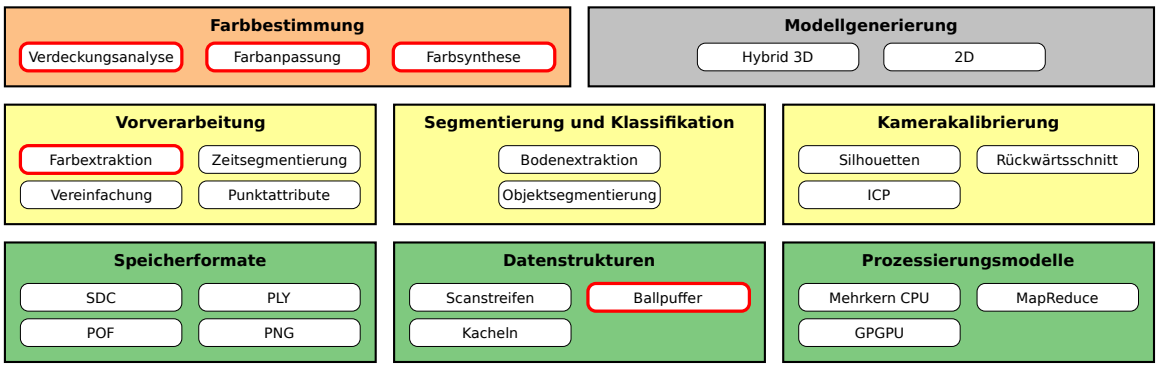


Abbildung 6.1: Einordnung des Verarbeitungsmoduls Farbbestimmung in das vorgestellte Framework.

6.1 Farbextraktion

Im Falle des vorliegenden Mobile Mapping Systems können die Farbwerte der einzelnen Scanpunkte aus den während der Aufnahmefahrt aufgenommen Kamerabildern extrahiert werden. Grundsätzlich wird dazu jeder Scanpunkt in das Kamerakoordinatensystem zum Aufnahmezeitpunkt der Fotos transformiert und dem jeweiligen Scanpunkt der Farbwert an den resultierenden Pixelkoordinaten des entsprechenden Fotos zugeordnet. Dabei kann nicht jeder Punkt in jedes Foto transformiert werden, z.B. wenn sich der Punkt hinter der Kamera befindet. Daher muss hier zunächst ermittelt werden, welcher Punkt in welches Foto transformiert werden kann. Da es sehr ineffizient ist, jeden Punkt in jedes Foto zu projizieren, werden lediglich Punkte mit Bildern getestet, welche eine gewisse zeitliche und damit auch räumliche Nähe aufweisen. Dafür wird, wie schon bei der Erstellung der Scanpunktbilder in Kapitel 5, die in Abschnitt 3.2.2 beschriebene Scanstreifenbasierte Pufferstrategie genutzt. Dabei besteht die Möglichkeit, dass ein Punkt in mehrere Fotos transformiert werden kann und daher entschieden werden muss, welche der gefundenen Farben genutzt werden soll. In diesen Fällen sind Varianten vorstellbar, welche genau einen gefundenen Farbwert auswählt oder auch eine Farbe aus mehreren Farbwerten kombiniert. Ein gängiges Verfahren liegt in der Wahl der Farbe, welche aus dem Kamerabild extrahiert wurde, das die kleinste räumliche Distanz zum jeweiligen Scanpunkt aufweist. Die Distanz ergibt sich dabei aus der Position des Scanpunktes und der Position der Kamera zum Aufnahmezeitpunkt des entsprechenden Bildes. Diese Auswahlstrategie wird mit einigen Erweiterungen auch im Rahmen dieser Arbeit angewendet. Zunächst werden jedoch die Farbwerte aus allen in Frage kommenden Kamerabilder, in Form eines Vorverarbeitungsschritts, extrahiert. Zur persistenten Speicherung der extrahierten Farbwerte wird die Möglichkeit des PLY Datenformats genutzt, beliebige Punktattribute zu definieren. Zusätzlich zum eigentlichen Farbwert wird darüber hinaus noch eine Bild-ID gespeichert. Über diese ID kann später das Kamerabild, aus dem der Farbwert stammt, zugeordnet werden.

6.2 Verdeckungsanalyse

Wie einleitend beschrieben wird zur Einfärbung der Punkte der Farbwert des Fotos gewählt, welches die kleinste räumliche Entfernung zum jeweiligen Punkt aufweist. Diese Auswahlstrategie berücksichtigt jedoch nicht etwaig auftretende Verdeckungen zwischen Vorder- und Hintergrundobjekten. Die vorliegende Anordnung von Scannern und Kameras führt jedoch dazu, dass Punkte bzw. Objekte zum einen von unterschiedlichen Positionen und darüber hinaus zu unterschiedlichen Zeitpunkten erfasst werden. Abbildung 6.2 zeigt die beschriebe Erfassungssituation. Die Fassade wird wie in Abbildung 6.2 (links) gezeigt zum Zeitpunkt T_1 von Position P_1 aus vom Laserscanner erfasst. Die Farbinformation zu diesem Punkt wird von der Kamera erst zum Zeitpunkt T_2 von Position P_2 aus erfasst (rechts). Zu diesem Zeitpunkt und von dieser Position aus ist jedoch der Fassadenpunkt von einem anderen Vordergrundobjekt, in diesem Fall einem parkenden PKW, verdeckt. Wird nach der zuvor beschriebenen Methode der Fassadenpunkt eingefärbt, so erhält dieser den Farbwert des davor parkenden PKWs.

Abbildung 6.3 zeigt häufig auftretende Fälle von Verdeckungssituationen anhand der daraus resultierenden falschen Farbzuordnungen. Dabei zeigt (1) die Projektion einer Laterne mit Ampel auf die dahinter liegende Fassade, wohingegen (2) das Bild eines Radfahrers auf der Fahrbahn zeigt. Fall (3) zeigt, dass nicht nur Verkehrsteilnehmer und Straßenmöblierung für Verdeckungen verantwortlich sind, sondern auch die Fassade eines Hauses eine andere Fassade desselben Hauses verdecken kann. Beispiel (4) zeigt abschließend eine großflächige Farbfehlzuordnung aufgrund eines großen und nahe am Erfassungsfahrzeug befindlichen Transporters.

Geht man von der Einteilung in Vorder- und Hintergrundobjekte aus, so lässt sich Fall (3) dort nicht eindeutig einordnen, da sich das Gebäude als Objekt hier quasi selbst verdeckt. Die zugehörige Erfassungssituation ist in Abbildung 6.4 dargestellt.

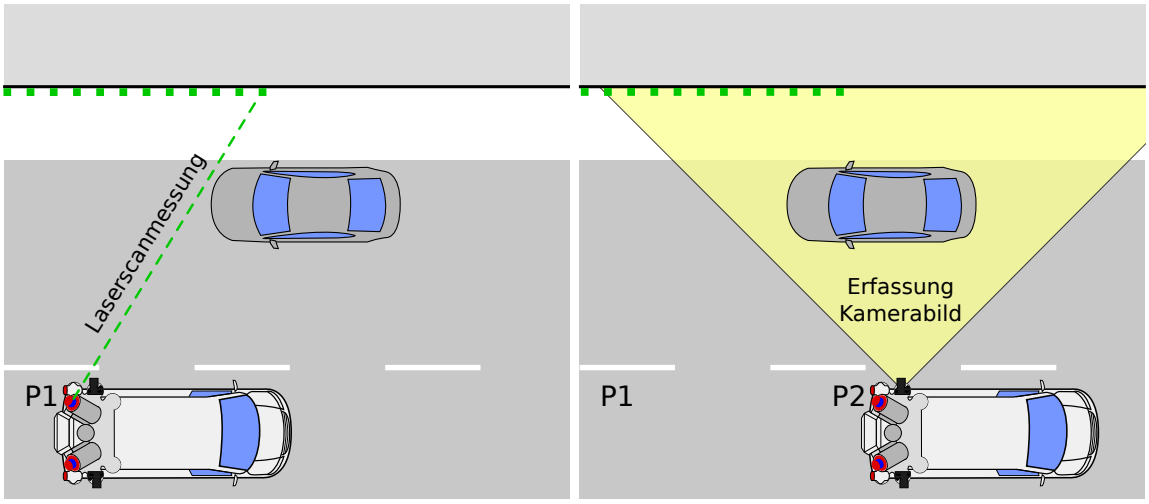


Abbildung 6.2: Erfassung von Objekten mittels Laserscanner (links, Zeitpunkt T_1 und Position P_1) und Kamera (rechts, Zeitpunkt T_2 und Position P_2) eines Mobile-Mapping Systems.

Abbildung 6.3: Häufig auftretende Verdeckungssituationen: (v.l.n.r.) (1) Laterne mit Ampel auf Fassade, (2) Radfahrer auf Straße, (3) Vorderfassade auf Seitenfassade (vgl. Abbildung 6.4), (4) Transporter auf Zaun und Fassade

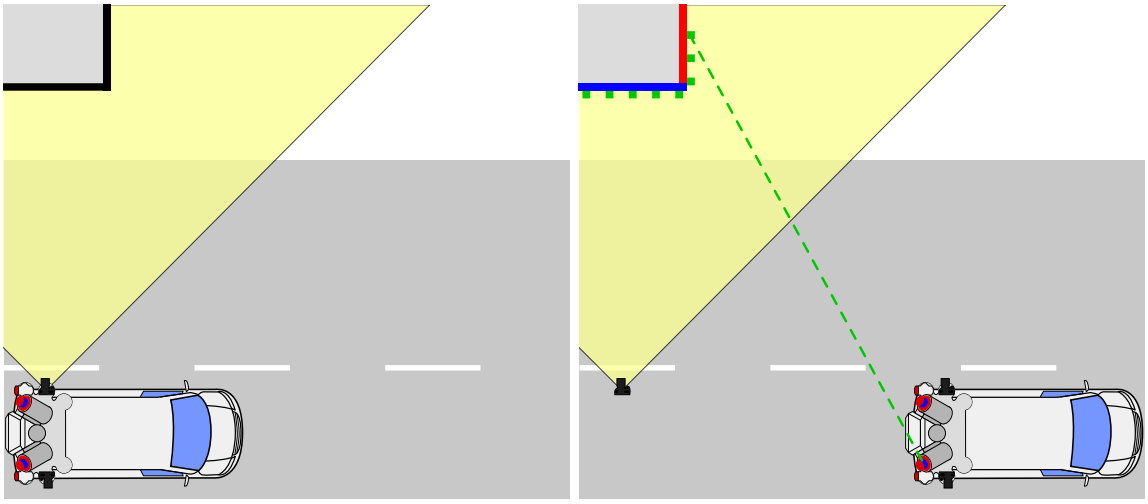


Abbildung 6.4: Selbstverdeckungssituation: Kein separates Vordergrundobjekt verdeckt die Fassade, sondern eine Fassade des Gebäudes (blau) verdeckt eine andere Fassade (rot) desselben Gebäudes im Kamerabild.

6.2.1 Geometrische Verdeckungsanalyse

Der semantische Ansatz von Hammoudi u. a. (2012), wie in Abschnitt 2.5 beschrieben, hat zwei entscheidende Nachteile. Der vorgeschlagene Ansatz sieht vor, die Daten zunächst zu segmentieren und anschließend in verdeckende und verdeckte Objekte zu klassifizieren. Je nach gewähltem Verfahren ist dies zum einen verhältnismäßig aufwendig und zum anderen gilt es die üblichen Segmentierungs- und Klassifizierungsprobleme, wie Über- oder Untersegmentierung und Falschklassifizierungen zu adressieren. Darüber hinaus werden als verdeckte Objekte lediglich Fassaden angenommen. Wie die Beispiele in Abbildung 6.3 zeigen, sind neben Fassaden auch alle weiteren Objekte, wie zum Beispiel Straßen, von potentiellen Verdeckungen betroffen. Die geschilderte Selbstverdeckung eines Objektes (vgl. Abb. 6.3 (3) und Abb. 6.4) wird ebenfalls nicht berücksichtigt.

Eine rein geometrische Verdeckungsanalyse vereinfacht das universelle Erkennen von beliebigen Verdeckungssituationen. Darüber hinaus adressiert sie sämtliche Nachteile der semantischen Verdeckungsanalyse. Für eine rein geometrische Verdeckungsanalyse muss für jeden Punkt entschieden werden, ob sich ein weiterer Punkt zwischen ihm und der jeweiligen Kamera befindet. Der naheliegendste Ansatz besteht in einem Ray-tracing Verfahren. Dabei wird für den Sichtstrahl (ray) von der Kamera zum zu testenden Scanpunkt überprüft, ob sich ein weiterer Scanpunkt zwischen ihnen befindet. Ist dies der Fall, ist der zu testende Punkt verdeckt. Die Ermittlung aller auf einem Sichtstrahl befindlichen Scanpunkte ist verhältnismäßig aufwendig, da prinzipiell für jeden Scanpunkt der Abstand zum besagten Sichtstrahl berechnet werden muss. Darüber hinaus muss dies für alle Kamerastandpunkte (m) und alle Scanpunkte (n) durchgeführt werden. Geht man von $m \ll n$ aus, so ergibt sich eine quadratische Laufzeit von $\mathcal{O}(m * n^2) = \mathcal{O}(n^2)$. Die effektive Laufzeit kann durch geeignete Datenstrukturen und Heuristiken reduziert werden, sie bleibt jedoch quadratisch.

Bei den zu verarbeitenden Punktmengen ist eine lineare Laufzeit wünschenswert. Diese kann durch die Nutzung eines Tiefenpuffers erreicht werden. Ein Tiefenpuffer, auch Z-Puffer genannt, erfasst Tiefen- bzw. Entfernungsangaben, häufig in Raster- oder Matrixform. Dafür wird für jedes Kamerabild, d.h. für jeden Aufnahmestandpunkt, ein Tiefenpuffer generiert, indem die Entfernungen aller relevanten Scanpunkte zum Aufnahmestandpunkt ermittelt und in der Tiefenpuffermatrix gespeichert werden. Soll für einen beliebigen Scanpunkt die Verdeckungssituation in einem Kamerabild ermittelt werden, so muss lediglich die Entfernung des Punktes zum Aufnahmestandpunkt mit dem Wert aus dem Tiefenpuffer überprüft werden. Ist der im Tiefenpuffer gespeicherte Wert größer oder gleich der Entfernung des Scanpunktes zum Kamerastandpunkt, so ist der besagte Scanpunkt nicht verdeckt. Damit ist die Laufzeit der Überprüfung auf Verdeckung (Abfrage des Wertes aus dem Tiefenpuffer) pro Scanpunkt konstant. Über alle Scanpunkte ergibt sich somit eine lineare Laufzeit. Die Laufzeit der Erstellung der Tiefenpufferstrukturen beträgt $\mathcal{O}(n)$. Der eigentliche Test auf Verdeckung benötigt abermals $\mathcal{O}(n)$, was eine lineare Laufzeit von $\mathcal{O}(2n) = \mathcal{O}(n)$ ergibt.

Die Erstellung der Tiefenpuffer kann im vorliegenden Framework über zwei Verfahren umgesetzt werden. Da es sich wieder um eine *pro Kamerabild* Operation handelt, kommt die Scanstreifenbasierte Pufferstrategie aus Abschnitt 3.2.2 in Frage. Andererseits kann der Tiefenpuffer auch über die in Abschnitt 3.2.3 vorgestellte Rasterdatenstruktur in Kombination mit der gezeigten Cachingoptimierung aus Abschnitt 3.2.4 erstellt werden. Letztere Variante hat den Vorteil, dass schnell Tiefenpuffer für beliebige Kamerabilder erstellt werden können. Die Verwendung der Scanstreifenbasierten Pufferstrategie erfordert dagegen eine streng sequentielle Verarbeitung und ist damit etwas weniger flexibel. Daher wurde zur Erstellung der Tiefenpuffer die besagte Rasterdatenstruktur genutzt.

Großen Einfluss auf die Genauigkeit und Qualität der Verdeckungsanalyse mittels Tiefenpuffer hat die Art und Weise wie die Datenstruktur des Tiefenpuffers die Position der Scanpunkte abbildet und speichert. Da es sich um Richtungsangaben bezüglich des Aufnahmestandpunktes handelt, werden hier die Polarkoordinaten des Scanpunktes genutzt. Aus den Winkeln der Polarkoordinaten wird die Position im Raster bzw. der Matrix ermittelt, wobei der Distanzwert an der besagten Stelle gespeichert wird. Jedoch bildet eine Matrix die Winkelkoordinaten nur ungenügend ab, bspw. ist die Winkelauflösung am Äquator wesentlich kleiner als an den Polen. Die Pole selbst können dabei in der Regel gar nicht abgebildet werden. Für eine zuverlässige und genaue Verdeckungsanalyse wird daher

keine Matrixstruktur als Tiefenpuffer, sondern eine, den Polarkoordinaten gerechtere, ballbasierte Struktur genutzt.

6.2.2 Ballbasierter Tiefenpuffer

Wie bereits beschrieben kann ein Tiefenpuffer über verschiedenste Datenstrukturen umgesetzt werden. Akkumulatoren, wie sie bei der Hough-basierten Ebenendetektion angewendet werden, haben ähnliche Charakteristika, nur dass sie keinen Distanzwert, sondern eine Anzahl speichern. Borrmann u. a. (2011) diskutiert die entsprechenden Vor- und Nachteile verschiedenster Akkumulatortypen. Dabei wurde ein ballbasiertes Design vorgeschlagen, welches den entscheiden Vorteil gegenüber den übrigen aufweist, dass sämtliche Winkelpositionen die gleiche Fläche auf der Einheitskugel einnehmen. Abbildung 6.5 veranschaulicht das vorgeschlagene ballbasierte Design. Die Breitenkreise verteilen sich dabei gleichmäßig als Streifen vom Nord- zum Südpol. Jeder (Breitenkreis-)Streifen hat eine variable Anzahl an Feldern, welche dem jeweiligen Längengrad entsprechen. Die Anzahl der Felder ergibt sich dabei aus dem Umfang des Breitenkreises, welcher vom entsprechenden Streifen repräsentiert wird. Die Pole erhalten dabei einen Streifen mit genau einem Feld. Dieses Design wurde im Rahmen dieser Arbeit aufgegriffen und für einen ballbasierten Tiefenpuffer angepasst.

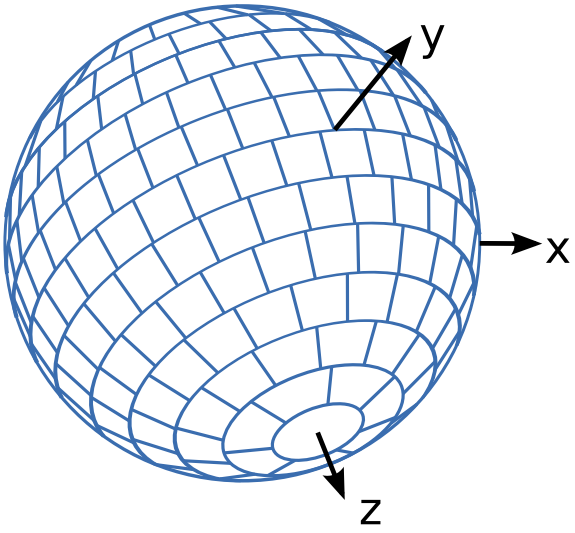


Abbildung 6.5: Ballbasierter Tiefenpuffer mit gleichgroßen Flächen für alle Winkelpositionen in Anlehnung an Borrmann u. a. (2011).

Abbildung 6.6 zeigt einen abgewickelten Tiefenpuffer auf Basis des beschriebenen Balldesigns. Die gespeicherten Entfernungswerte wurden über die Helligkeit kodiert. Dunkel bedeutet eine geringe Entfernung, wohingegen hell eine große Entfernung widerspiegelt. Auffällig ist der LKW in der linken Bildhälfte. Während die Scanpunkte des Hauses im Hintergrund eine geschlossene Oberfläche bilden, ergibt die punktuelle Erfassung des im Vordergrund befindlichen LKWs keine geschlossene Oberfläche. Dies ist der geringen Entfernung zum Erfassungsfahrzeug geschuldet, woraus sich für die erfassten Scanpunkte deutlichere Winkelunterschiede in den Polarkoordinaten ergeben.

Eine Verdeckungsanalyse auf Basis eines solchen Tiefenpuffers ergäbe lediglich die Verdeckung einiger weniger Punkte der Hintergrundobjekte, obgleich das Vordergrundobjekt (LKW) das gesamte Hintergrundobjekt (Gebäude) in der Realität, sowie im entsprechenden Kamerabild verdeckt. Diesem Problem lässt sich über eine entfernungsabhängige Größe des Scanpunktes begegnen. Dazu wird eine gewisse Punktgröße $size_{unit}$ (bspw. 0,1 Meter) in einer definierten Entfernung (bspw. 1 Meter) angenommen und linear mit der tatsächlichen Entfernung des Punktes verkleinert, wobei die kleinste Punktgröße von der Rasterauflösung abhängt. Die tatsächliche Punktgröße $size_{dist}$ (bzw.

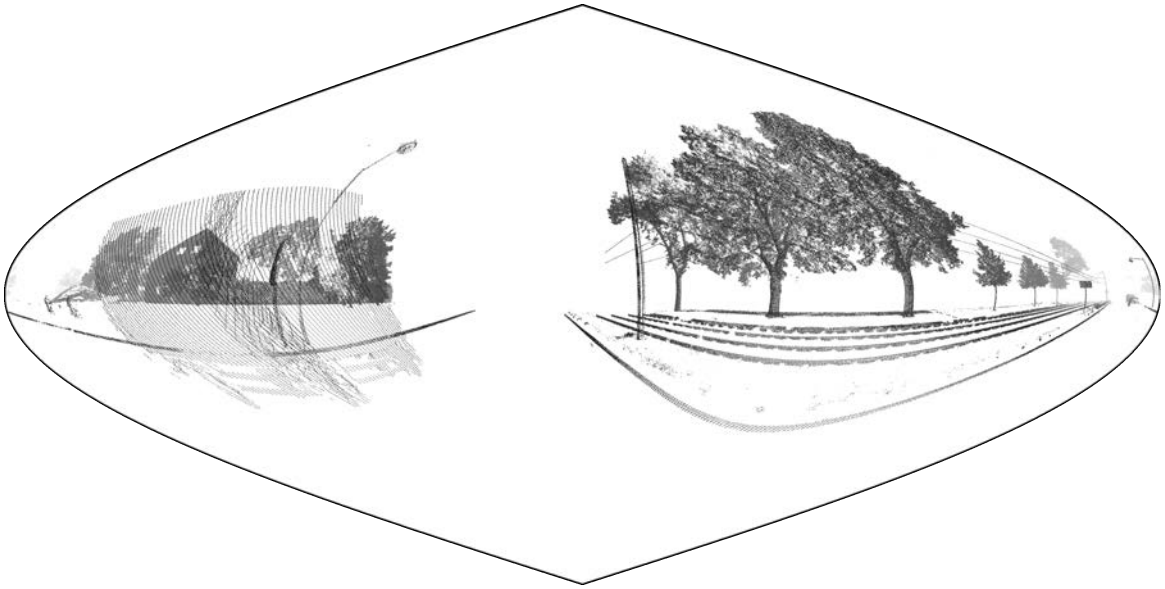


Abbildung 6.6: Darstellung eines abgewickelten, ballbasierten Tiefenpuffers.

deren Winkelausdehnung) bezüglich der Distanz zur Kamera $dist_{cam}$ kann somit, wie in Gleichung 6.1 gezeigt, als Arcus-Tangens von $dist_{cam}$ und $size_{unit}$ modelliert werden.

$$size_{dist} = \arctan(size_{unit}, dist_{cam}) \tag{6.1}$$

Der Zusammenhang zwischen Entfernung, Punktgröße und resultierender Winkelausdehnung ist in Abbildung 6.7 noch einmal veranschaulicht. Auf diese Weise nimmt die Winkelausdehnung (α_1) bei Punkten mit geringem Abstand zur Kamera ($Distanz_1$) zu und fällt mit zunehmendem Abstand zur Kamera (α_2 und $Distanz_2$). Das Ergebnis zeigt Abbildung 6.8. Durch die entfernungsabhängige Punktgröße wird im Tiefenpuffer das im Hintergrund befindliche Gebäude vollständig vom davor befindlichen LKW verdeckt.

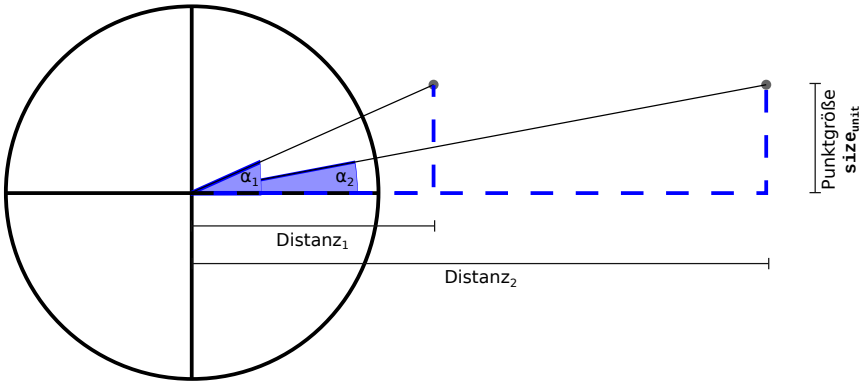


Abbildung 6.7: Veranschaulichung des Zusammenhangs zwischen Entfernung und Winkelausdehnung eines Punktes.

Im Gegensatz zur gezeigten geringen Punktdichte eines beliebigen Vordergrundobjektes ist die Punktdichte auf dem Boden sehr hoch. Da die Scanpunkte der Fahrbahn nicht nur sehr nahe an der Kamera liegen (i.d.R. ca. 2 Meter) sondern auch noch sehr dicht erfasst wurden, kann dies zu Problemen beim Tiefenpuffer führen. Die Nähe zur Kamera ergibt im beschriebenen Modell eine relativ hohe Punktgröße, so dass bereits wenige Scanpunkte des Bodens genügen um selbigen im Tiefenpuffer komplett zu verdecken. Beim anschließenden Test, ob ein Scanpunkt verdeckt ist, kann es nun aufgrund der

hohen Punktdichte vorkommen, dass ein Scanpunkt der Fahrbahn als verdeckt identifiziert wird und daher nicht eingefärbt werden kann. Da Punkte auf dem Boden ohnehin nur minimales Verdeckungspotential besitzen (die Fahrbahn selbst verdeckt üblicherweise keine anderen Objekte), werden diese bei der Erstellung des Tiefenpuffers nicht berücksichtigt. Dazu werden wie in Abschnitt 4.4.1 beschrieben sämtliche zum Boden gehörige Scanpunkte segmentiert und vor der Erstellung des Tiefenpuffers aus der Punktwolke entfernt. Dies hat darüber hinaus den Vorteil, dass somit viel weniger Werte im Tiefenpuffer gespeichert werden und dieser sich daher stärker komprimieren lässt, was wiederum deren Ladegeschwindigkeit erhöht.

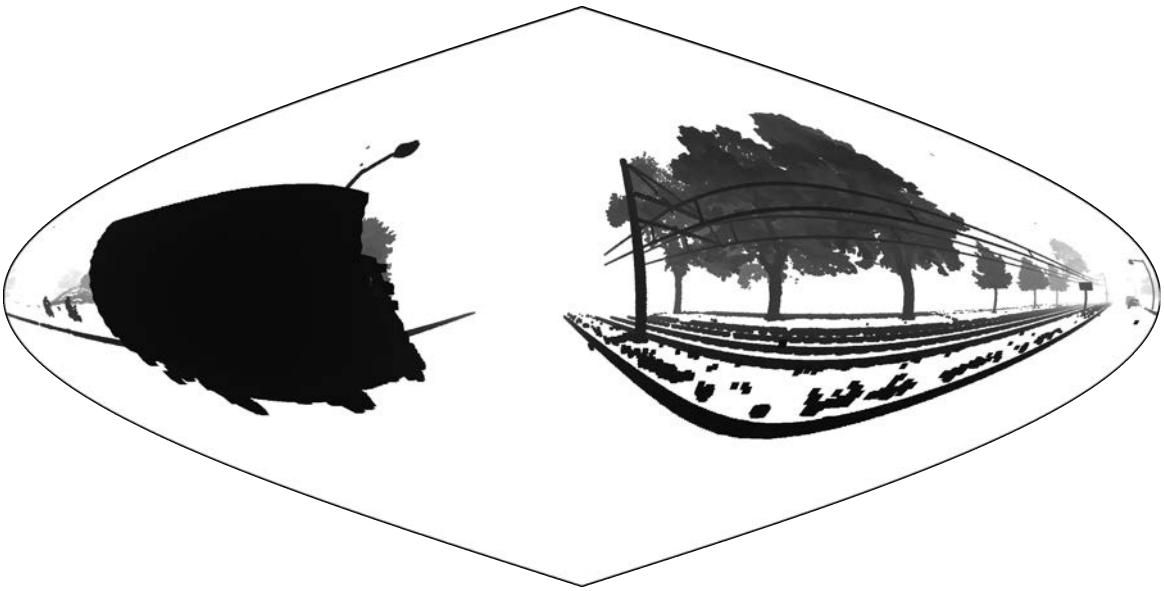


Abbildung 6.8: Darstellung eines abgewickelten, ballbasierten Tiefenpuffers mit entferungsbasierter Punktgrößenmodellierung (vgl. Abb. 6.6).

6.2.3 Ergebnisse

Abbildung 6.9 zeigt die Verdeckungssituation aus Abbildung 6.3 (1). Bild (1) zeigt das Kamerabild aus dem die Farbwerte für die Fassade extrahiert wurden. Bild (2) zeigt den zum Kamerastandpunkt gehörigen ballbasierten Tiefenpuffer, wohingegen Bild (3) eine nach Verdeckung eingefärbte Punktwolke zeigt. Magenta eingefärbte Punkte repräsentieren verdeckte und weiß eingefärbte repräsentieren nicht verdeckte Punkte. Dabei wird deutlich, dass die Boden- und Fassadenpunkte die von der Laterne verdeckt werden, als solche erkannt wurden und nicht mit Farbwerten aus Bild (1) eingefärbt werden dürfen. Bild (4) zeigt schließlich die eingefärbte Punktwolke unter Berücksichtigung der Verdeckungssituation. Die Farbwerte verdeckter Punkte wurden aus Kamerabildern mit anderen Aufnahmestandpunkten extrahiert.

Abbildung 6.10 zeigt die Situation Selbstverdeckung aus Abbildung 6.3 (3). Bild (1) zeigt analog zum vorherigen Beispiel das Foto für die Farbwert-Extraktion. Wie deutlich zu erkennen ist, ist die rechte Fassade komplett verdeckt und darf nicht aus diesem Foto eingefärbt werden. Darüber hinaus verdeckt die Laterne im Vordergrund die dahinter liegende Fassade ebenfalls. Bild (3) zeigt wieder unter Verwendung des Tiefenpuffers aus Bild (2) die nach Verdeckung eingefärbten Punkte. Wie man sieht ist der Bereich hinter der Laterne, als auch ein Großteil (vgl. Abschnitt 6.2.4) der seitlichen Fassade als verdeckt markiert. Abschließend zeigt Bild (4) die eingefärbte Punktwolke als Resultat der Verdeckungsanalyse.

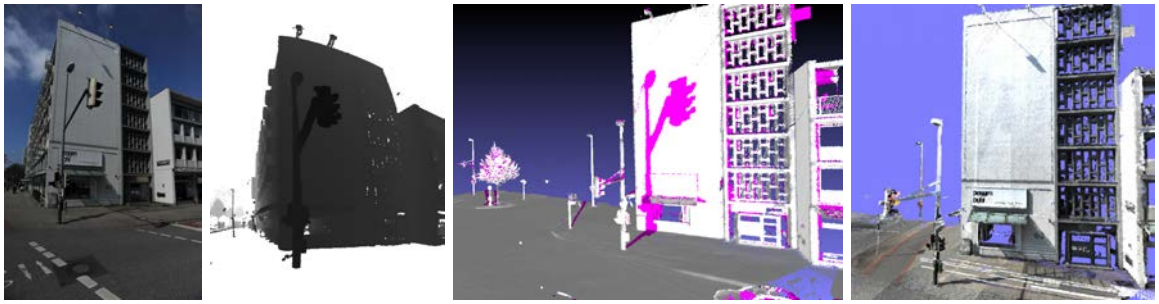


Abbildung 6.9: Verdeckungssituation und Ergebnis der Verdeckungsanalyse (v.l.n.r.): (1) Kamerabild zur Farbwertextraktion; (2) Ballbasierter Tiefenpuffer für Kamerastandpunkt; (3) nach Verdeckung eingefärbte Punktwolke; (4) eingefärbte Punktwolke unter Berücksichtigung der Verdeckung (vgl. Abb. 6.3 (1)).

Abbildung 6.10: Verdeckungssituation und Ergebnis der Verdeckungsanalyse (v.l.n.r.): (1) Kamerabild zur Farbwertextraktion; (2) Ballbasierter Tiefenpuffer für Kamerastandpunkt; (3) nach Verdeckung eingefärbte Punktwolke; (4) eingefärbte Punktwolke unter Berücksichtigung der Verdeckung (vgl. Abb. 6.3 (3)).

6.2.4 Nicht erfasste und dynamische Objekte

Bei genauer Betrachtung des Beispiels aus Abbildung 6.10 werden einige Probleme des Ansatzes deutlich. Da der Tiefenpuffer zur Verdeckungsanalyse aus den Laserscandaten generiert wird, werden auch nur Verdeckungen abgebildet die auch tatsächlich erfasst werden konnten. Werden Objektteile nicht erfasst, so entsteht eine Lücke im Tiefenpuffer und Punkte werden fälschlicherweise als nicht verdeckt eingestuft. Die nach Verdeckung eingefärbte Punktwolke aus Abbildung 6.10 (3) zeigt eine solche Situation. Da Fenster den Laserstrahl selten bis gar nicht reflektieren, fehlen an dieser Stelle Scanpunkte auf der Fassade und der Tiefenpuffer bildet dementsprechend nicht die korrekte Verdeckungssituation ab, so dass nicht die komplette seitliche Fassade des Gebäudes als verdeckt markiert wird.

Eine weitere Ursache für nicht erfasste Objekte bzw. Objektteile sind dynamische Objekte. Wie die Abbildungen der Erfassungssituationen 6.2 und 6.4 zeigen, wird ein und dasselbe Objekt zu drei unterschiedlichen Zeitpunkten erfasst. Beispielsweise zum Zeitpunkt T_1 vom ersten Laserscanner, zum Zeitpunkt T_2 von der Kamera und schließlich zum Zeitpunkt T_3 vom zweiten Laserscanner. Bewegt sich das Objekt innerhalb dieser Zeitspanne, so kann die Verdeckungssituation zum Zeitpunkt T_2 nicht korrekt aus den erfassten Scanpunkten der Zeitpunkte T_1 und T_3 abgeleitet werden. Dabei werden üblicherweise Regionen als verdeckt markiert die zum Aufnahmezeitpunkt des Fotos T_2 nicht mehr verdeckt sind bzw. waren und im Gegenzug Regionen welche eigentlich verdeckt sind als nicht verdeckt markiert. Ein Beispiel dafür stellt der Radfahrer aus Abbildung 6.3 (2) dar. Können die Scanpunkte einem konkreten Objekt zu unterschiedlichen Erfassungszeitpunkten zugeordnet werden, so könnte die Position des Objektes zum Aufnahmezeitpunkt des Fotos interpoliert und entsprechend korrekt abgebildet werden. So muss zunächst der besagte Radfahrer als eigenständiges Objekt sowohl aus den Punkten des ersten (Zeitpunkt T_1), als auch des zweiten Laserscanners (Zeitpunkt T_3), identifiziert werden. Anschließend muss erkannt werden, dass es sich um ein und denselben Radfahrer handelt, welcher sich in der Zwischenzeit bewegt hat. Aus den Positionen des Radfahrer zum Zeitpunkt T_1 und T_3 ließe sich dann die Position des Radfahrer zum Zeitpunkt der Aufnahme des

Kamerabildes T_2 interpolieren und die Scanpunkte des Radfahres an entsprechender Stelle in den Tiefenpuffer schreiben.

6.3 Farbanpassung

Wie eingangs beschrieben, werden die Scanpunkte über das jeweils nächstgelegene Kamerabild eingefärbt. Das Ergebnis zeigt Abbildung 6.11. Hier sind deutliche Farb- bzw. Helligkeitssprünge zu erkennen. Diese resultieren in der Regel daraus, dass räumlich benachbarte Scanpunkte aus unterschiedlichen Kamerabildern eingefärbt wurden. Je nach Kameraausstattung des Mobile Mapping Systems sind unterschiedliche Einfärbesituationen möglich. Die resultierende Einfärbung hängt im Wesentlichen davon ab, ob benachbarte Punkte aus Kamerabildern von identischen oder unterschiedlichen Kameras eingefärbt wurden und welche zeitliche bzw. räumliche Nähe die Aufnahmestandpunkte der Kameras aufweisen. Tabelle 6.1 veranschaulicht die sich aus den genannten Faktoren ergebenen Einfärbesituationen.

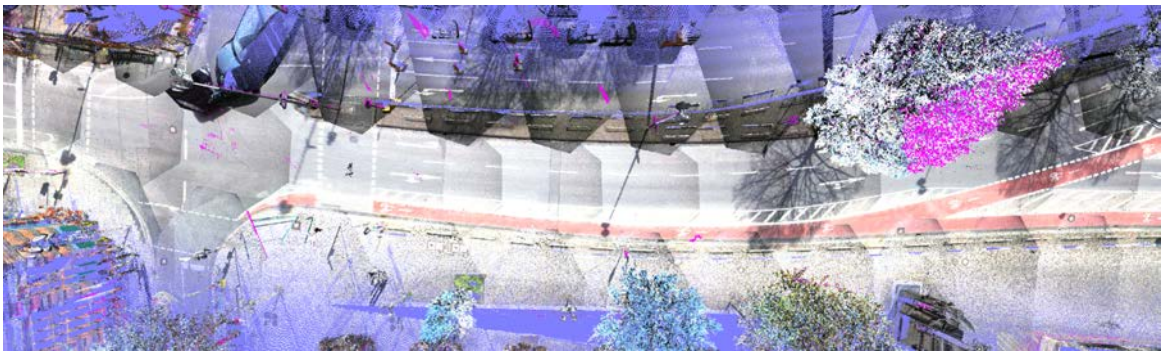


Abbildung 6.11: Scanpunkte eingefärbt über Farbwerte aus dem jeweils nächstgelegenen Kamerabild. Deutlich zu Erkennen sind die Übergänge zwischen den jeweiligen zum Einfärben genutzten Kamerabildern.

6.3.1 Einfärbesituationen benachbarter Scanpunkte

Wie Abbildung 6.12 verdeutlicht treten Einfärbesituation I und II sehr häufig auf und lassen sich auch nicht vermeiden. Situationen III und IV können bei Verdeckungen auftreten (Abb. 6.12 Baumstamm) oder wenn Scanpunkte bspw. ober- oder unterhalb des Bildausschnittes liegen (Abb. 6.12 Laterne). Werden benachbarte Punkte aus Bildern unterschiedlicher Kameras eingefärbt, so spiegeln sich die unterschiedlichen Kameraparameter (Belichtungszeit, ISO-Filmempfindlichkeit, Weißabgleich, usw.) deutlich in der Farbgebung wieder. Mit größer werdendem Abstand (zeitlich und räumlich) zwischen den Aufnahmestandpunkten der Kameras nehmen Umgebungseinflüsse (Sonnenstand, Abschattungen, usw.) deutlich zu und beeinflussen wiederum die Farbgebung der Scanpunkte.

Um eine für Visualisierungszwecke ansprechende Einfärbung der Scanpunkte zu erreichen müssen die aufgezeigten Problemsituationen entweder vermieden oder die Symptome, wie sichtbare Übergänge, kaschiert werden. Wie zuvor beschrieben lassen sich die Situationen I und II nicht vermeiden. Jedoch können große Abstände zwischen den Kamerastandpunkten vermieden werden. Dazu werden

		Kamera	
		identisch	unterschiedlich
Abstand	klein	I	II
	groß	III	IV

Tabelle 6.1: Übersicht über Einfärbe-Situationen benachbarter Scanpunkte in Abhängigkeit von der Kamera und der Abstände der Aufnahmestandpunkte.

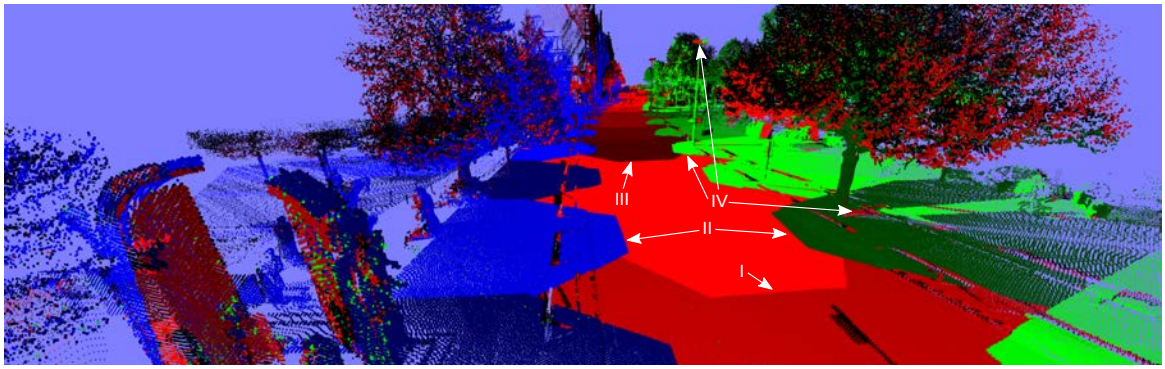


Abbildung 6.12: Punktwolke eingefärbt nach Bildquellen, Rot entspricht Kamera 1, Grün Kamera 3 und Blau Kamera 4, wobei unterschiedliche Helligkeiten unterschiedliche Bilder bedeuten. Beispiele für Einfärbe-Situationen aus Tabelle 6.1 sind markiert.

im Rahmen der Ermittlung der Punktfarben eine zulässige maximale Entfernung, sowie ein Entfernungsfenster definiert. Ersteres begrenzt den maximalen Abstand zwischen Aufnahmestandpunkt der Kamera und Scanpunkt. Im Rahmen dieser Arbeit wurde ein Grenzwert von 100 Metern genutzt. Das heißt für die Bestimmung der Scanpunktfarben werden nur Kamerabilder berücksichtigt, deren Aufnahmestandpunkte weniger als 100 Meter vom jeweiligen Scanpunkt entfernt sind. Darüber hinaus ist dieser Grenzwert maßgebend für die benötigten Puffergrößen der bei der Farbbestimmung verwendeten Scanstreifenbasierten Pufferstrategie aus Abschnitt 3.2.2. Mit wachsendem zulässigen Maximalabstand steigt die notwendige Puffergröße des Verfahrens. Dies hat zum einen starken Einfluss auf die Laufzeit (vgl. Abschnitt 3.2.2) und zum anderen können derart große Puffer benötigt werden, dass diese nicht mehr im Arbeitsspeicher gehalten werden können. Da ohnehin die Punktdichte bei großen Entfernungen stark abnimmt, stellt der 100 Meter Grenzwert einen guten Kompromiss zwischen Prozessierbarkeit, Laufzeit und Umfang der Punkte ohne Farbinformation dar.

Der für die Vermeidung der Problemsituationen III und IV ausschlaggebende Grenzwert ist jedoch die Größe des Entfernungsfensters. Dieser beschränkt den Entfernungsbereich, aus welchem Farben für einen Scanpunkt in Betracht kommen. Bei einer maximalen Entfernung von 100 Metern könnte die Farbe eines Punktes aus einer Vielzahl von Bildern extrahiert werden. Beispielsweise könnten für einen Scanpunkt Farben aus Kamerabildern mit folgenden Entfernungen extrahiert werden: $10m, 15m, 80m, 90m$. Da grundsätzlich immer das nächstgelegene Kamerabild verwendet wird scheint dies zunächst unerheblich, nicht jedoch wenn die Verdeckungsanalyse ergibt, dass der Scanpunkt in den Kamerabildern mit den Entfernungen $10m$ und $15m$ verdeckt ist. Demnach würde der Scanpunkt dann mit der Farbe aus dem 80m entfernten Kamerabild eingefärbt. Da jedoch, wie beschrieben, die Umgebungseinflüsse mit steigender Entfernung zunehmen und das Kamerabild unter Umständen sogar von einer anderen Kamera erfasst wurde, als die Kamerabilder aus 10 und 15 Meter Entfernung, wird sich diese Stelle farblich stark von den umgebenden nicht verdeckten Punkten abheben. Die Einführung des Entfernungsfensters beschränkt diesen Effekt. Bei zu kleinem Fenster kann unter Umständen kein Ersatz für einen verdeckten Farbwert gefunden, wohingegen bei zu großem Fenster die Farbabweichung durch die Umgebungseinflüsse zu stark werden. In dieser Arbeit wurde bei der Einfärbung eine Größe von 30 Metern für das Entfernungsfenster gewählt. Für das beschriebene Beispiel würde dies jedoch bedeuten, dass diesem Scanpunkt keine Farbe zugeordnet werden kann.

Neben der Vermeidung der Problemsituationen bleibt häufig nur noch das Kaschieren der auftretenden Farbsprünge. Eine solche Farbangleichung wird in der Photogrammetrie als radiometrische Anpassung bezeichnet, vgl. Kraus und Jansa (1996). Dabei werden die Grauwerte überlappender Bildbereiche einander angeglichen um sichtbare Bildübergänge zu entfernen. Ein solcher auf 3D Scanpunkte angepasster Ansatz zur radiometrischen Farbanpassung sollte demnach auch die sichtbaren Farb- bzw. Helligkeitssprünge (vgl Abb. 6.11) benachbarter Scanpunkte beseitigen. Da in diesem Kontext die menschliche Farbwahrnehmung maßgeblich ist, finden alle Anpassungsoperationen im HSV Farbmodell (vgl. Abschnitt 2.3) statt.

6.3.2 Objektweise Farbanpassung

Aufgrund von dynamischen Helligkeitsverläufen lassen sich keine globalen Helligkeitsanpassungen pro Bild definieren. Beispiele für diese Helligkeitsverläufe zeigt Abbildung 6.11, sowie in vereinfachter qualitativer Form Abbildung 6.13. Betrachtet man einen Bereich b_1 auf der Fahrbahn, welcher komplett aus einem Kamerabild eingefärbt wurde, so existieren Helligkeitsunterschiede zu vorausgegangenen Kamerabildern b_0 (jeweils links) von hell (b_1) nach dunkel (b_0). Im Gegensatz dazu ist der Helligkeitsverlauf zu folgenden Kamerabildern b_2 (jeweils rechts) genau umgekehrt, also dunkel (b_1) nach hell (b_2). Versucht man nun über diese Differenzen die Helligkeit für b_1 auszugleichen, so mitteln sich die jeweils entgegengesetzten Differenzen quasi zu Null. Somit lässt sich eine globale Farbanpassung wie sie El-Hakim u. a. (1998) vorschlagen (vgl. Abschnitt 2.3.1) nicht durchführen.

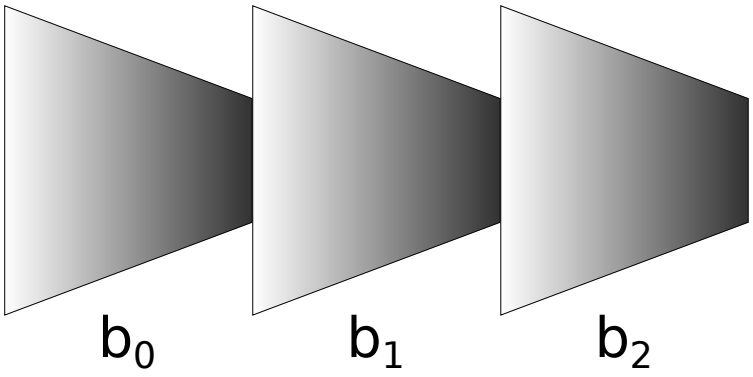


Abbildung 6.13: Qualitative Helligkeitsverläufe benachbarter Regionen b_0, b_1 und b_2.

Um die genannten Helligkeitsverläufe korrekt auszugleichen, muss also eine von der Lage des Punktes abhängige Funktion gefunden werden, welche den vorliegenden Verlauf kompensiert. Dies ist bei drei räumlichen Freiheitgraden jedoch sehr fehleranfällig und kann schnell zu stark schwingenden Funktionen und somit zu entsprechend schlechten oder gar falschen Helligkeitsanpassungen führen. Daher wird die Punktwolke zunächst nach dem in Abschnitt 4.4 beschriebenen Verfahren in einzelne Objekte segmentiert. Anschließend wird eine separate Farbanpassung für jedes Objekt durchgeführt. Dies hat den Vorteil, dass der Punktumfang, sowie die räumliche Ausdehnung eher gering sind, was die Laufzeit verbessert und sich darüber hinaus etwaige Fehler lediglich lokal auswirken. Sollten optisch nicht ansprechende Farbanpassungen ermittelt werden, so beschränkten sich diese auf einzelne Objekte und nicht auf die gesamte Szene. Da im Rahmen der Objektsegmentierung ebenfalls der Boden als eigenständiges Objekt segmentiert wird, findet eine getrennte Farbanpassung von Boden und übrigen Objekten statt. Dies hat den Vorteil, dass im Falle des in der Regel flachen Bodens eine lediglich zweidimensionale Anpassungsfunktion bestimmt werden kann.

6.3.3 IDP-Interpolierte radiometrische Helligkeitsanpassung von Bodenpunkten

Wie beschrieben ist es kaum möglich einen statischen Wert für die Helligkeitsanpassung ein Bildes zu finden. Zur bestmöglichen Anpassung muss daher der funktionale Zusammenhang der Helligkeitsunterschiede in Abhängigkeit zur Position der Punkte ermittelt werden. Da die Bodenpunkte quasi auf einer Ebene liegen, genügt es hier eine Funktion in Abhängigkeit von der Lage der Punkte zu bestimmen, die Höhe wird dabei ignoriert. Dies entspricht grob der radiometrischen Helligkeitsanpassung bei der Erstellung eines Mosaiks aus digitalen Orthophotos, vgl. Kraus und Jansa (1996). Jedoch unterscheidet sich die Form der Bildausschnitte und der jeweiligen Überlappungsbereiche im Falle der Mobile Mapping Daten stark von denen des Orthophoto-Bildverbandes. Den in der Regel rechteckigen Orthophotos und rechteckigen Überlappungenbereichen stehen beliebige Ausschnitte und Überlappungsbereiche der Mobile Mapping Daten gegenüber. Daher können die meisten Ansätze aus diesem Bereich nur eingeschränkt angewandt werden.

Als erstes müssen die jeweiligen Bild- und Überlappungsbereiche identifiziert werden. Dazu werden die Bodenpunkte zunächst auf ein zweidimensionales Raster projiziert. Da sowohl die Punktdichte, als auch -verteilung nicht konstant sind, wird anschließend die morphologische Operation *close* angewendet, um die Lücken zwischen den Scanpunkten zu schließen. Abbildung 6.14 zeigt beide Schritte an einem Beispiel, wobei links die Pixel nach Bildquelle eingefärbt und rechts die aus den jeweiligen Bildern extrahierten Farbwerte verwendet wurden.

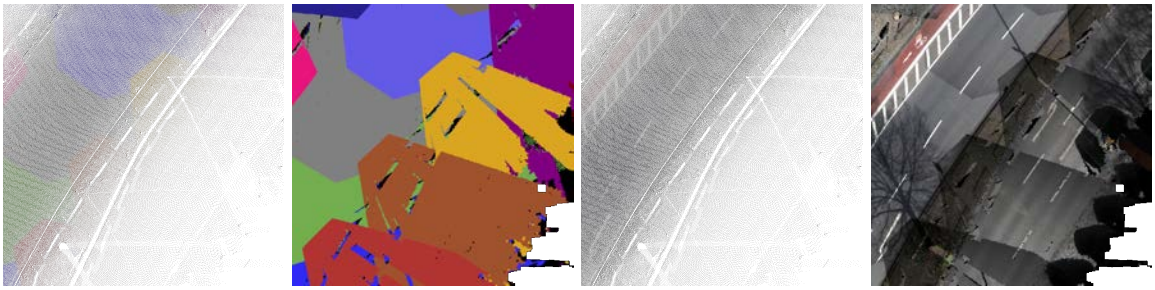


Abbildung 6.14: Projizierte Scanpunkte mit anschließender morphologischer close *Operation. Pixel farbcodiert nach Bildquelle (links). Punkte eingefärbt mit aus Bildern extrahiertem Farbwert (rechts).*

Wie beschrieben sind die Überlappungsbereiche der Mobile Mapping Daten nicht so klar abgegrenzt, wie bei der Verarbeitung von Orthophotos. Ein Bild einer nach hinten gerichteten Kamera hat beispielsweise einen Überlappungsbereich mit allen davor aufgenommenen Bildern bei geradem Streckenverlauf, was bei der Ermittlung der Überlappungsbereiche berücksichtigt werden muss. Für die Beseitigung der Farb- bzw. Helligkeitssprünge ist lediglich der grenznahe Überlappungsbereich relevant. Die Nutzung des gesamten Bereiches hingegen kann die Ermittlung einer Anpassungsfunktion sogar negativ beeinflussen. Daher ist es notwendig, die grenznahen Überlappungsbereiche zu identifizieren. Angenommen es soll der Überlappungsbereich der Region B aus Abbildung 6.15 (links) mit den angrenzenden Regionen bestimmt werden. Unter der Prämisse, dass die Farbe eines Punktes immer vom nächstgelegenen Kamerabild (hier B) stammt, unterscheiden sich die Überlappungssituationen zu den Bereichen A und C. Im Falle der Überlappung von B und A liegt diese lediglich in Region A, wohingegen die Überlappung von B und C in Region B liegt. Damit definiert sich der Überlappungsbereich nicht einfach über alle Grenzpunkte der Region B, sondern unter Umständen auch über Grenzpunkte der Nachbarregionen.

Wurde der Überlappungsbereich, wie beispielhaft als rote Linie in Abbildung 6.15 (links) skizziert, identifiziert, kann auf Basis der Helligkeitsdifferenzen der jeweiligen Punkte eine Anpassungsfunktion ermittelt werden. Grundsätzlich liegt hier die Bestimmung einer ellipsoiden Anpassungsfunktion nahe, welche als Approximationsfunktion über die Kleinste Quadrate-Methode durch die Helligkeitsdifferenzen gelegt wird. Bei genauer Betrachtung der Verteilung der Helligkeitsdifferenzen, wie in Abbildung 6.15 (rechts) dargestellt, wird jedoch klar, dass eine Flächenfunktion mit einem sehr hohen Polynomgrad notwendig ist, um die Differenzen ausreichend gut anzunähern. Da Polynome mit hohem Grad zu starken Schwingungen tendieren, wird ein alternativer Ansatz zu Ermittelung einer Anpassungsfunktion verfolgt. Hierzu werden zunächst Schlüsselpunkte identifiziert. Schlüsselpunkte sind, wie in Abbildung 6.15 (links) in grün skizziert, Punkte in denen eine Überlappung von drei oder mehr Regionen vorliegt. Ausgehend von diesen Schlüsselpunkten werden anschließend die Helligkeitsdifferenzen des Überlappungsbereichs geglättet, so dass sich die Helligkeitsdifferenzen benachbarter Punkte nur minimal von einander unterscheiden.

Ausgehend von den geglätteten Helligkeitsdifferenzen des Überlappungsbereichs werden die Helligkeitsanpassungen der Punkte innerhalb der betrachteten Region interpoliert. Dabei hat sich die Abstandsfunktion IDP als robuste Alternative zu anderen Verfahren (bspw. mittels Delaunay Vermaschung) herauskristallisiert. Für jeden Punkt p werden in der Region die dichtesten Überlappungspunkte in der Achter-Nachbarschaft ermittelt. Die Helligkeitsanpassung für p ergibt sich dann als gewichtetes Mittel der gefundenen Überlappungspunkte. Als Gewichte wurden die quadrierten Distanzen der Überlappungspunkte herangezogen. Abbildung 6.16 veranschaulicht die durchgeführte Interpolation mittels Abstandsfunktion IDP an einem Beispiel.

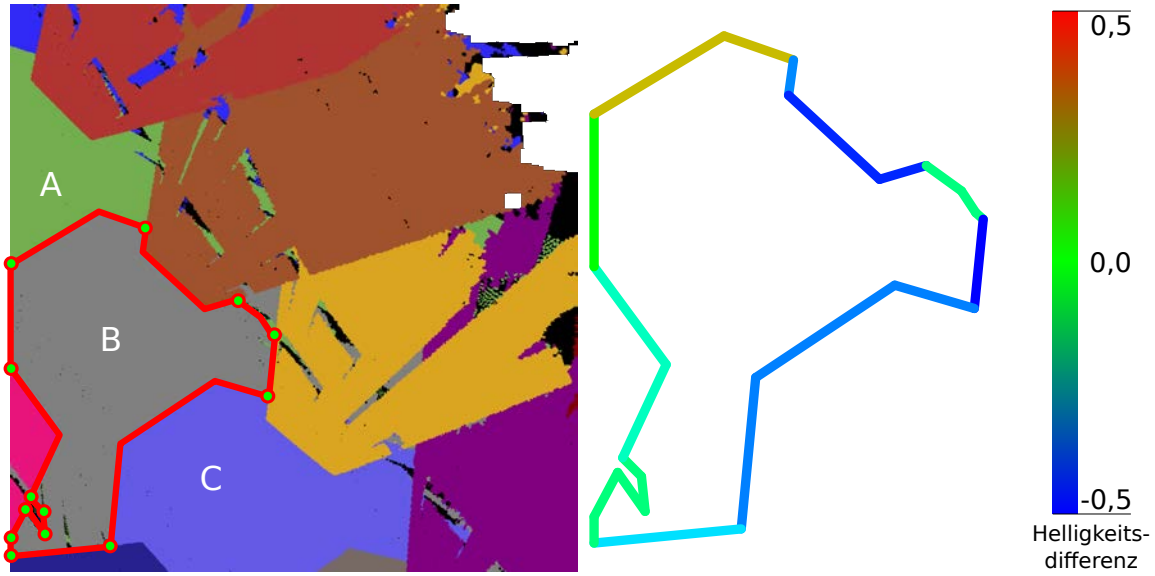


Abbildung 6.15: Grenznaher Überlappungsbereich (rot) mit markierten Schlüsselpunkten (grün) (links) und Helligkeitsdifferenzen im Überlappungsbereich (rechts)

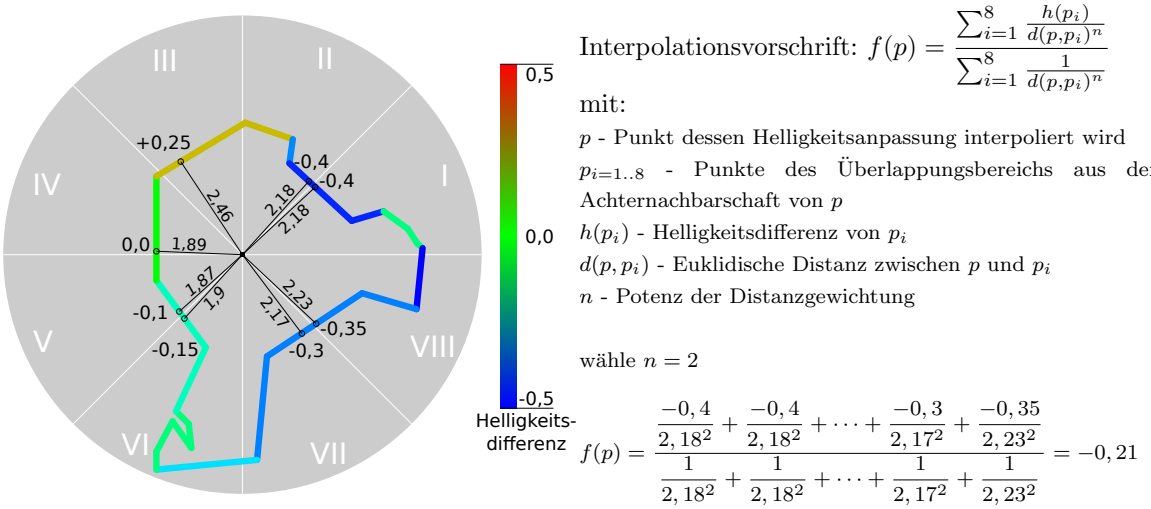


Abbildung 6.16: Beispiel für IDP Interpolation auf Basis der grenznahen Überlappungspunkte.

Da die Helligkeitsanpassung exakt den Helligkeitsdifferenzen der Grenzregion entspricht (bis auf die durchgeführte Glättung), werden die vorhandenen Helligkeitssprünge komplett beseitigt. Sämtliche verbliebenen sichtbaren Differenzen resultieren entweder aus der durchgeführten Glättung oder aus etwaigen Sättigungs- und Farbwertdifferenzen. Das Ergebnis des beschriebenen Verfahrens ist ein zweidimensionales Raster mit den entsprechenden Helligkeitsanpassungen. Zur Anpassung der Helligkeiten aller Scanpunkte wird der zur Lage des Punktes im Raster gehörende Anpassungswert ermittelt. Auf Basis des gefundenen Wertes wird die Helligkeit des Punktes entsprechend angepasst. Das Ergebnis ist in Abbildung 6.17 darstellt. Es wird deutlich, dass die meisten Helligkeitssprünge beseitigt wurden. Unter Umständen sind nach wie vor sichtbare Differenzen vorhanden, welche nicht durch die Helligkeitsanpassung beseitigt werden konnten. Diese können durch nicht beseitigte Verdeckungen oder stärkere Unterschiede in den Sättungswerten hervorgerufen werden.

Abbildung 6.17: Ergebnis der vorgestellten IDP interpolierten Farbanpassung für Bodenpunkte (vgl. Abb. 6.11)

6.3.4 Radiometrische Helligkeits- und Sättigungsanpassung von Objektpunkten

Im Gegensatz zum Boden sind alle übrigen Objektpunktwolken nicht notwendigerweise eben. Daher muss hier ein anderer Ansatz zur Farbanpassung verfolgt werden. Wie eingangs beschrieben, kommen räumliche (dreidimensionale) Anpassungsfunktionen nicht in Frage, da hier die Gefahr von starken Schwingungen das Ergebnis negativ beeinflussen kann. Darüber hinaus ist die Anzahl an beteiligten Fotos und Kameras, aus denen die Punkte eines Objektes eingefärbt werden, wesentlich geringer als bei den zuvor diskutierten Bodenpunkten. Während der Boden aus mehreren Dutzend Fotos und prinzipiell aus allen Kameras eingefärbt wird, liegt die Anzahl der beteiligten Fotos bei Objekten wie Fassaden oder Autos im eher einstelligen Bereich. Aus diesen Gründen wird bei der Farbanpassung einzelner Objekte ein Ansatz verwendet, welcher einen (für das jeweilige Objekt) globalen Anpassungswert für Helligkeit und Sättigung pro Bild bestimmt (vgl. 2.3.1).

Abbildung 6.18 zeigt die bekannte Ausgangssituation bei der Einfärbung für ein segmentiertes Objekt (Transporter). Die nach Bildquelle farbcodierte Darstellung (links) zeigt, dass das Objekt im Wesentlichen aus drei Bildern (hellgrün, dunkelgrün und dunkelrot) eingefärbt wird. Wie zuvor repräsentieren die grünen Farbtöne Bilder von Kamera 3 und die roten Farbtöne Bilder von Kamera 1. Wie das Ergebnis ohne Farbanpassung (rechts) zeigt, heben sich die Bereiche die von Bildern unterschiedlicher Kameras eingefärbt wurden (grün-rot), deutlich ab (Einfärbe-Situation III und IV nach Tabelle 6.1). Ein Unterschied zwischen aufeinander folgenden Bildern derselben Kamera (grün-grün) ist dahingegen kaum auszumachen (Einfärbe-Situation I nach Tabelle 6.1).

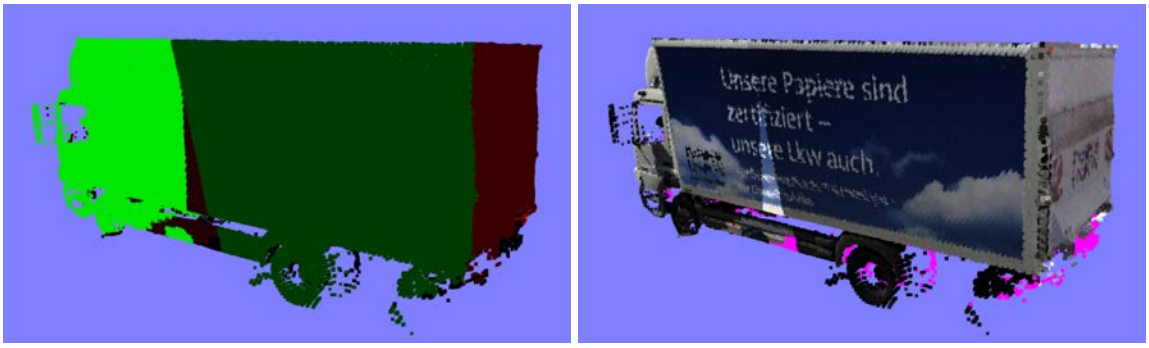


Abbildung 6.18: Einfärbesituation für ein segmentiertes Objekt (Transporter), farbcodiert nach Bildquelle (links) und das Einfärbe-Ergebnis ohne Farbanpassung (rechts).

Um eine bestmögliche radiometrische Anpassung, besonders zwischen Bildern unterschiedlicher Kameras, zu erreichen wird nicht nur die Helligkeit, sondern darüber hinaus auch die Sättigung angeglichen. Dazu wird zunächst wieder der grenznahe Überlappungsbereich ermittelt. Im Gegensatz zur zuvor beschriebenen Farbanpassung des Bodens muss dies im 3D Raum erfolgen. Dazu wird die euklidische Distanz aller Punkte eines Einfärbebereichs zu jeweils allen anderen Einfärbebereichen ermittelt. Liegt dieser Abstand unter einem zuvor festgelegten Grenzwert für grenznahe Punkte

(bspw. 0,1 Meter), so zählt der entsprechende Punkt zum grenznahen Überlappungsbereich. Abbildung 6.19 veranschaulicht den gefundenen grenznahen Überlappungsbereich für das Beispielobjekt aus Abbildung 6.18.

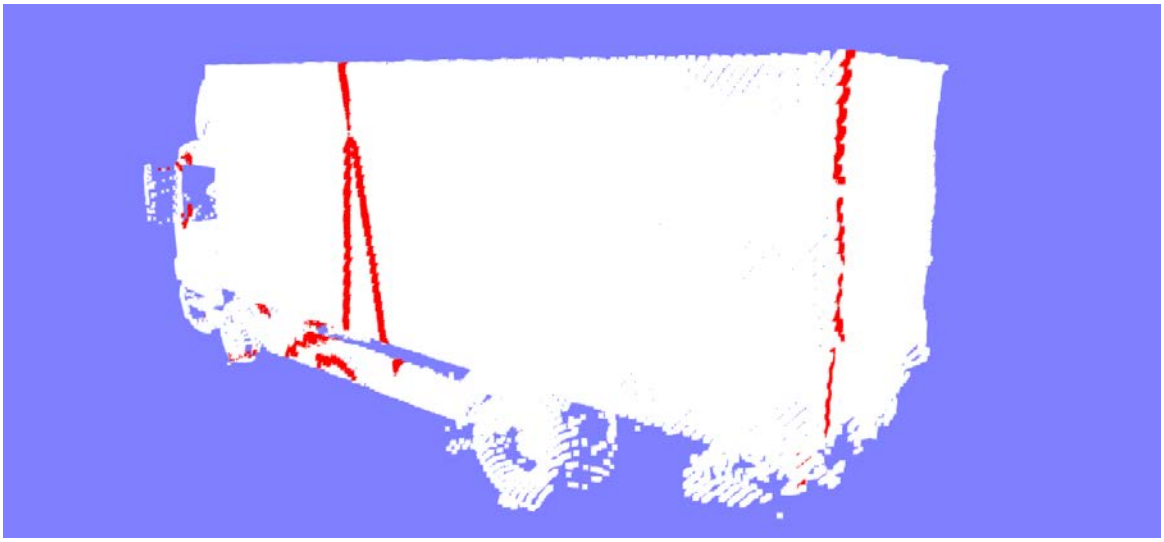


Abbildung 6.19: Ermittelter grenznaher Überlappungsbereich (rot) für den Transporter aus Abbildung 6.18.

Die vorliegenden Helligkeits- und Sättigungsdifferenzen der Punkte des ermittelten Überlappungsbereichs gehen anschließend in die Bestimmung der jeweiligen Anpassungswerte für jedes Bild ein. Die entsprechenden Differenzen werden dann mittels der *Kleinste Quadrate-Methode* minimiert. Somit ergibt sich für jedes Objekt ein Anpassungswert für Helligkeit und Sättigung für jedes bei diesem Objekt zum Einfärben genutzte Bild. Abbildung 6.20 zeigt das Ergebnis der durchgeführten radiometrischen Anpassung. Wie Abbildung 6.20 (links) zeigt, bleibt ein sichtbarer Unterschied zwischen Bereichen die aus unterschiedlichen Kameras eingefärbt wurden bestehen, wenn lediglich eine Anpassung der Helligkeit durchgeführt wird. Deutlich zu sehen an der Heckklappe des LKWs und am dreieckigen Bereic in der Mitte der Seitenwand. Bei gleichzeitiger Anpassung der Sättigung werden die sichtbaren Unterschiede weiter reduziert und sind nahezu unauffällig.

Abbildung 6.20: Ergebnisse unter Anpassung der Helligkeit (links) und unter Anpassung von Helligkeit und Sättigung (rechts).

Nicht konstante Übergänge wie bei der IDP-interpolierten Helligkeitsanpassung der Bodenpunkten können auf diesem Wege leider nicht ausgeglichen werden. Jedoch sind diese an Fassaden und anderen Objekten, im Gegensatz zum Boden, ohnehin zum einen seltener und zum anderen weniger ausgeprägt vorhanden.

6.4 Farbsynthese

In Abhängigkeit von der Erfassungssituation kann es passieren, dass Punkte erfasst wurden, denen keine Farbe zugeordnet werden kann. Dies kann mehrere Ursachen haben. Zum einen besteht die Möglichkeit, dass ein Punkt in sämtlichen Fotos verdeckt ist, was gerade bei großen Vordergrundobjekten häufig passiert. Ein weiterer Grund ist der beschränkte Öffnungswinkel der Kameras. Befindet sich die Kamera zum Zeitpunkt der Aufnahme sehr nahe am jeweiligen Objekt, bspw. einer Hausfassade, dann ist der Öffnungswinkel unter Umständen nicht groß genug um die gesamte Fassade zu erfassen. Daher können die obersten Punkte hoher Gebäude, an denen während der Erfassung dicht vorbeigefahren wird, häufig nicht eingefärbt werden. Abbildung 6.21 zeigt Beispiele beider Ursachen. Der obere Teil der Fassade wurde nicht von den seitwärts gerichteten Kameras erfasst. Farblose Punkte im unteren Bereich sind verdeckungsbedingt.

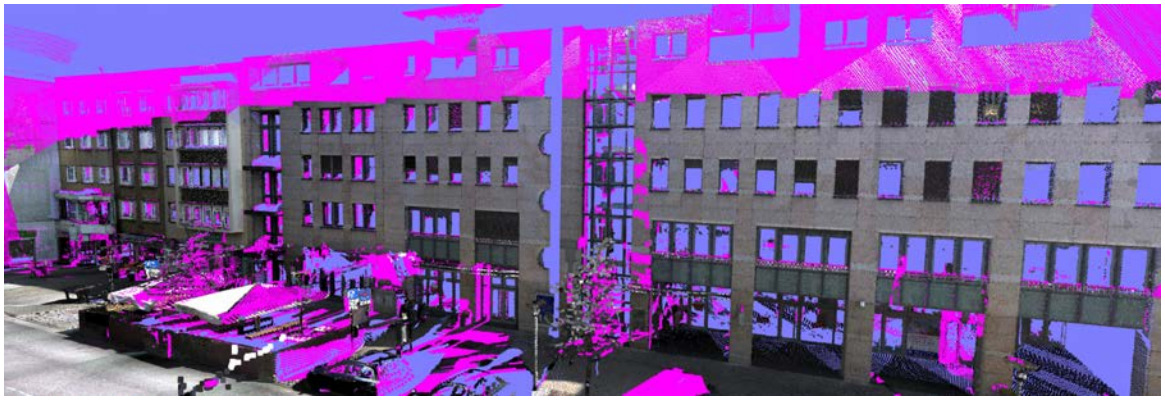


Abbildung 6.21: Für Magenta-farbene Punkte konnte kein Farbwert ermittelt werden, da der jeweilige Punkt in allen Kamerabildern verdeckt war (vorwiegend Punkte aus der untere Bildhälfte) oder sich außerhalb des Kameraausschnitts befindet (vorwiegend Punkte aus der oberen Bildhälfte).

Sollen die Mobile Mapping Daten für eine Visualisierung genutzt werden, so muss entschieden werden, wie mit farblosen Punkten zu verfahren ist. Die einfachste Option wäre, sie einfach aus der Punktwolke zu entfernen. Dies führt jedoch zu unschönen Löchern, was je nach Art der Visualisierung nicht akzeptabel ist. Die zweite Option stellt die Zuweisung einer einheitlichen, vordefinierten Farbe, bspw. Schwarz oder Weiß, dar. So bleibt zumindest die Geometrieinformation in den betroffenen Bereichen erhalten. Je nach gewählter Farbe und der Farbe der umgebenen Region führt dies unter Umständen wieder zu einer visuell nicht akzeptablen Darstellung. Die dritte Möglichkeit besteht in der Interpolation des Farbwertes anhand der Farbe der nächstgelegenen eingefärbten Nachbarpunkte. Gerade bei kleineren Regionen ohne Farbe, führt dies zu visuell ansprechenden Ergebnissen.

6.4.1 Histogrammbasierte Farbinterpolation

Im Rahmen dieser Arbeit wurde ein Histogrammbasierter Ansatz zur Farbinterpolation gewählt. Der interpolierte Farbwert setzt sich dabei aus dem häufigsten Farbton (Hue) und den mittleren Werten für Helligkeit und Sättigung der Nachbarpunkte zusammen.

Dafür werden zunächst alle Punkte in zwei Kategorien gruppiert. Gruppe 1 enthält alle Punkte ohne Farbwert, wohingegen Gruppe 2 alle Punkte mit erfolgreich ermittelten Farbwerten enthält. Im nächsten Schritt werden für jeden Punkt aus Gruppe 1 alle Nachbarn in Gruppe 2 mit einem zuvor festgelegten maximalen Abstand d_{max} ermittelt. Wurden genügend eingefärbte Nachbarpunkte gefunden, so kann eine Farbinterpolation erfolgen. Damit steuert d_{max} die Interpolationsweite. Bei der Wahl eines kleinen Wertes für d_{max} (bspw. einige Zentimeter) werden nur für farblose Punkte in kleinen Regionen Farben interpoliert. Soll die Farbe für größere Regionen interpoliert werden, so muss für d_{max} ein entsprechend großer Wert gewählt werden. Jedoch gilt, je größer die Interpolationsweite ist, desto schlechter bzw. auffälliger ist das Ergebnis.

Stehen für einen farblosen Punkt genügend eingefärbte Nachbarpunkte im definierten Bereich zur Verfügung, so kann für diesen eine Farbinterpolation durchgeführt werden. Dafür wird ein Histogramm der Farbtonwerte (Hue-Kanal des HSV Farbraums) erstellt. Das Histogramm gibt Aufschluss über den am häufigsten vorkommenden Farbwert. Dieser Wert wird für die zu interpolierende Farbe genutzt. Abschließend müssen noch die Sättigungs- und Helligkeitswerte ermittelt werden. Dafür wird aus allen Punkten mit dem gewählten Hue Wert der Median der Sättigungs- und Helligkeitswerte bestimmt. Um einen natürlicheren Eindruck zu erreichen wird auf beide Medianwerte ein kleines Rauschen addiert. Letzteres verhindert, dass farblose Punkte mit identischen Nachbarn die genau gleiche Farbe erhalten.

6.4.2 Ergebnis

Abbildung 6.22 zeigt Ergebnisse der beschriebenen Farbsynthese mit unterschiedlichen Interpolationsweiten d_{max} (20cm, 50cm, 1m und 2m). Im gezeigten Beispiel sind zwei wesentliche Ursachen für die farblosen Punkte zu unterscheiden. Der Grund für die farblosen Punkte in der unteren Hälfte der Szene ist bedingt durch verdeckende Straßenmöblierung. Die farblosen Punkte aus der oberen Hälfte der Szene lagen außerhalb des Kamerabildes der seitwärtsgerichteten Kamera. Einige der Punkte konnten jedoch Farben der rückwärtsgerichteten Kamera zugeordnet werden. Jedoch ist der Aufnahmewinkel der Fassade bezüglich der rückwärtsgerichteten Kamera sehr spitz. Dies führt wiederum dazu, dass große Teile der Fassade im Rahmen der Verdeckungsanalyse als verdeckt identifiziert werden und daher nicht allen Punkten Farben aus diesen Kamerabildern zugeordnet werden.

Die meisten der kleineren farblosen Bereiche der unteren Szenenhälfte sind durch die beschriebene Farbsynthese unauffällig kaschiert. Dies gilt ebenfalls für die größeren farblosen Bereiche der oberen Szenenhälfte. Der markante Helligkeitsunterschied liegt in diesem Fall in einer fehlgeschlagenen Farbanpassung.

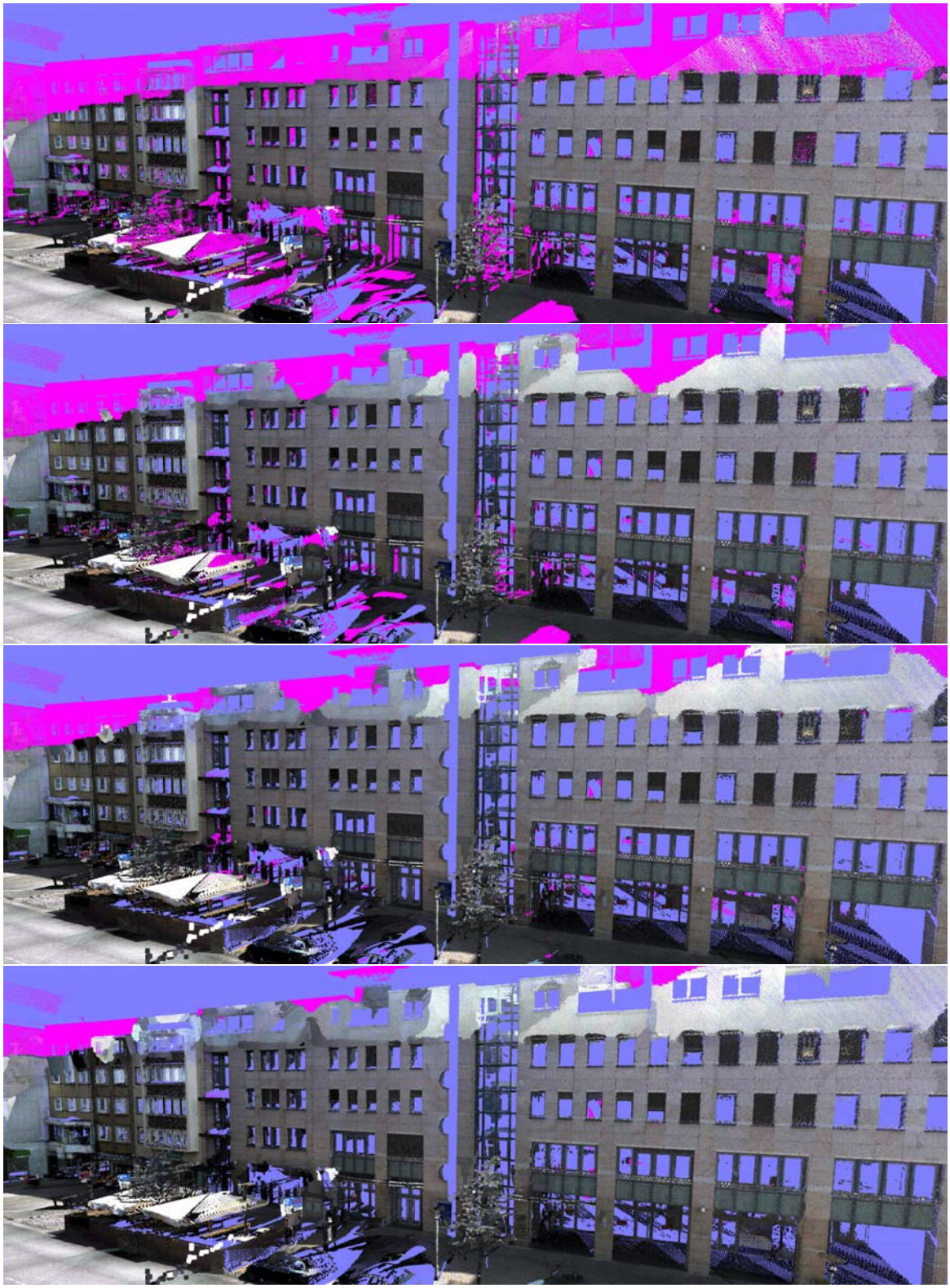


Abbildung 6.22: Ergebnisse der Farbsynthese mit unterschiedlichen Interpolationsweiten (v.o.n.u.): 20cm, 50cm, 1m und 2m.

7 Aus Punktwolken abgeleitete Modelle

Die Darstellung der Mobile Mapping Daten kann auf verschiedenste Weise erfolgen. Die Möglichkeiten reichen dabei von der Visualisierung einzelner Scanstreifen über die der eingefärbten Punktwolke bis hin zu abgeleiteten Modellen für spezielle Darstellungen. Bei der im folgenden beschriebenen Modellgenerierung wurde auf folgende Punkte wert gelegt. Zunächst sollen Modelle für eine effiziente Visualisierung erzeugt werden. Effizienz umfasst in diesem Kontext mehrere Aspekte. Zum einen ist das Ziel ein möglichst kompaktes, also speichereffizientes Modell zu generieren, welches darüber hinaus effizient gerendert, also mit möglichst hoher Bildwiederholrate dargestellt, werden kann. Neben allen Effizienzkriterien ist die Erstellung von qualitativ hochwertigen Modellen, im Sinne der Darstellungsqualität, ebenfalls eine der Zielgrößen. Darüber hinaus wurde besonderes Augenmerk auf die Entwicklung effizienter Modellgenerierungsalgorithmen gelegt. Wie gehabt stellt Abbildung 7.1 eine Einordnung der in diesem Kapitel behandelten Komponenten dar. Neben den Komponenten des Moduls zur Modellgenerierung wird in Abschnitt 7.1.4 auch auf das PNG Speicherformat und dessen Optimierungsmöglichkeiten eingegangen.

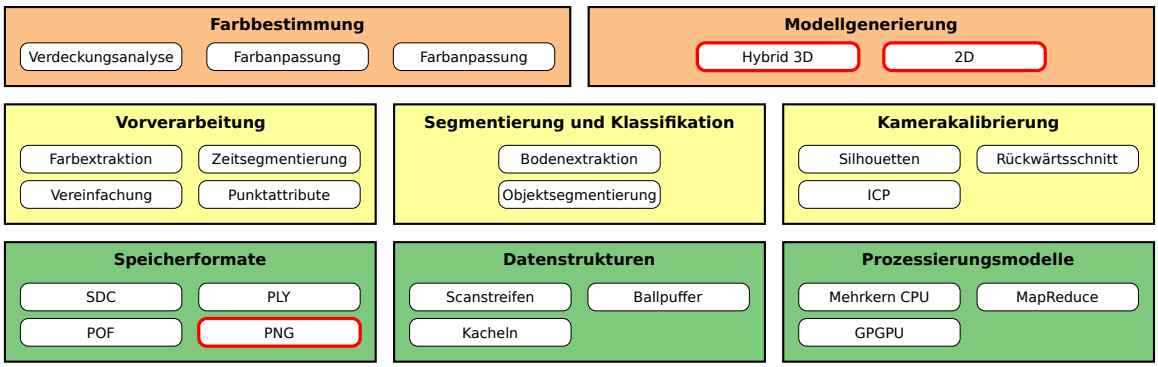


Abbildung 7.1: Einordnung des Moduls Modellgenerierung in den Gesamtkontext.

7.1 3D Modelle

Für eine Darstellung der kompletten Punktwolke einer Mobile Mapping Messfahrt kommt, aufgrund des Datenumfangs, einzig eine Out-Of-Core Variante in Frage. Out-Of-Core heißt dabei, dass lediglich Teile der Punktwolke im Speicher vorgehalten und dargestellt werden. Ein für dieses Konzept erstelltes Modell muss entsprechend in separaten Teilen vorliegen, die unabhängig voneinander geladen werden können. Dafür wird die ohnehin bereits in Kacheln partitionierte Rasterdatenstruktur weiter unterteilt und um ein Level-Of-Detail Konzept erweitert. Zur Optimierung der Speicher- als auch Rendereffizienz wird aus der eingefärbten Punktwolke wie im folgenden beschrieben ein hybrides Modell als Mischung aus 2D Texturen und den originalen Scanpunkten erstellt. Die Erstellung der besagten Texturen setzt die Identifikation planarer Bereiche (Abschnitt 7.1.1) voraus. Die gefundenen Ebenen müssen dabei gewissen Randbedingungen, wie Umfang, Ausdehnung, Planarität und Punktdichte genügen, um die besagten Ansprüche an Qualität und Effizienz zu erfüllen. Wurden entsprechende Ebenen identifiziert, werden die zugehörigen Punkte auf selbige projiziert und somit das Texturbild erzeugt. Texturen aus einzelnen projizierten Punkten weisen jedoch aufgrund vieler transparenter Bereiche optische Nachteile auf, was eine Nachbearbeitung (Abschnitt 7.1.2) notwendig macht. Für ein effizientes Rendering des erzeugten Modells gilt es abschließend die erstellten Texturen optimal zu verwalten und zu speichern (Abschnitt 7.1.3, 7.1.4), als auch ein hybrides LOD Konzept, welches sowohl Punktwolken und als auch Texturen unterstützt, zu entwickeln (Abschnitt 7.1.5).

7.1.1 Identifikation planarer Bereiche

Für die Erstellung von hybriden Modellen müssen zunächst planare Bereiche innerhalb der Punktwolke identifiziert und anschließend durch eine entsprechende Textur ersetzt werden. Einen allgemeinen Ansatz dazu zeigt Wahl u. a. (2005). Ein sehr großer Teil planarer Bereiche in urbanen Gebieten besitzt eine senkrechte Orientierung (bspw. Fassaden, Schilder, Zäune, ...), also die Normale der jeweiligen Ebenen ist senkrecht zum Vektor $\vec{Z} = (0, 0, 1)$. Sollen nur senkrechte Ebenen identifiziert werden, so reduziert sich die Suche von Ebenen in $\mathbb{R}^3$ auf Linien in $\mathbb{R}^2$. Diese Einschränkung gegenüber dem allgemeinen Fall erhöht die Laufzeiteffizienz um etwa eine Größenordnung und ermöglicht damit die effiziente Modellerzeugung bei umfangreichen Mobile Mapping Daten. Einziger Nachteil dieser Beschränkung ist, dass keine beliebig orientierten Ebenen gefunden und durch Texturen ersetzt werden. Prominente Vertreter nicht senkrecht orientierter Ebenen stellen der Boden, als auch Dächer dar. Aufgrund der Scananordnung ist die erfasste Punkdichte auf Dächern bzw. Dachschrägen naturgemäß sehr gering. Daher hat eine Vernachlässigung selbiger bei der Identifikation planarer Bereiche kaum Auswirkung auf die Erhöhung der Speichereffizienz. Die Punktdichte des Bodens ist jedoch signifikant, daher ist eine gesonderte Erstellung von Bodentexturen sinnvoll. Auf diese wird am Endes dieses Abschnittes eingegangen. Darüber hinaus wird anhand der Beispiele aus den Abbildungen 7.4 und 7.5 gezeigt, dass die bloße Identifikation von Geraden im $\mathbb{R}^2$ nicht genügt um relevante Ebenen zu ermitteln und speichereffiziente Texturbilder zu generieren. Zur Identifikation der optimalen Bereiche werden Clusteringverfahren sowohl für Punkte auf Geraden, als auch für Punkte auf Ebenen herangezogen. Abbildung 7.2 gibt einen Überblick über die durchgeführten Schritte, welche im folgenden im Detail erläutert werden.

Um einen visuell ansprechenden Ersatz der Punkte zu repräsentieren und darüber hinaus die Speichereffizienz nicht zu verringern, bzw. sogar zu erhöhen, sollte ein zu ersetzender planarer Bereich folgende Eigenschaften besitzen:

1. **Umfang:** der Bereich sollte eine Mindestanzahl an Punkten besitzen
2. **Ausdehnung:** der Bereich sollte eine Mindestausdehnung besitzen
3. **Planarität:** die Mehrzahl der Punkte eines Bereichs sollten planar (vgl. Abschnitt 4.3.3.2) sein
4. **Punktdichte:** der Bereich sollte eine Mindestpunktdichte besitzen

Die erste Bedingung (Umfang) stellt sicher, dass zum einen relevante Ebenen gefunden werden und die Speichereffizienz nicht leidet. Grundsätzlich werden für Ebenen im Raum nur drei unabhängige Punkt benötigt. So ließen sich beliebig viele Ebenen mit nur drei Punkten finden und durch eine Textur ersetzten. Dies sind geometrisch gesehen Ebenen, sie sind jedoch nicht relevant im gegeben Kontext. Die zweite Bedingung (Ausdehnung) verhindert die Erstellung sehr kleiner Texturen, welche in der Regel keine relevanten Ebenen repräsentieren. Die Planaritätsbedingung gewährleistet, dass die Punkte, welche durch die Textur ersetzt werden auch tatsächlich zu einer Ebene gehören und nicht Teile anderer Strukturen darstellen. Die letzte Bedingung (Punktdichte) stellt ähnlich der Umfangsbedingung sicher, dass die resultierende Textur nicht mehr Speicher benötigt, als die damit ersetzten Punkte (vgl. Abschnitt 7.1.4).

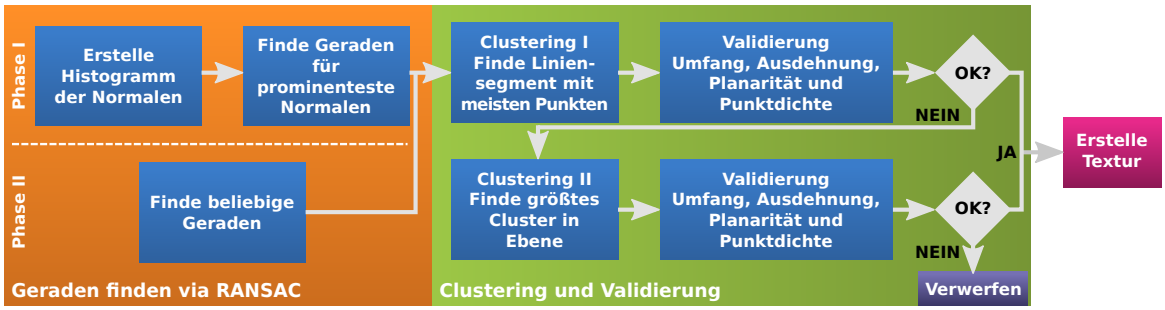


Abbildung 7.2: Ablauf Identifikation planarer Bereiche.

Die Suche von Linien mittels RANSAC (RANdom SAmple Consensus) liefert bei gewöhnlichen Fassaden wie in Abbildung 7.3 (links) gute Ergebnisse. Bei komplexeren Fassaden, wie in Abbildung 7.3 (rechts) gelingt dies jedoch nur mit genauer Anpassung der RANSAC Parameter, was wiederum negativen Einfluss bei anderen Situationen haben kann.

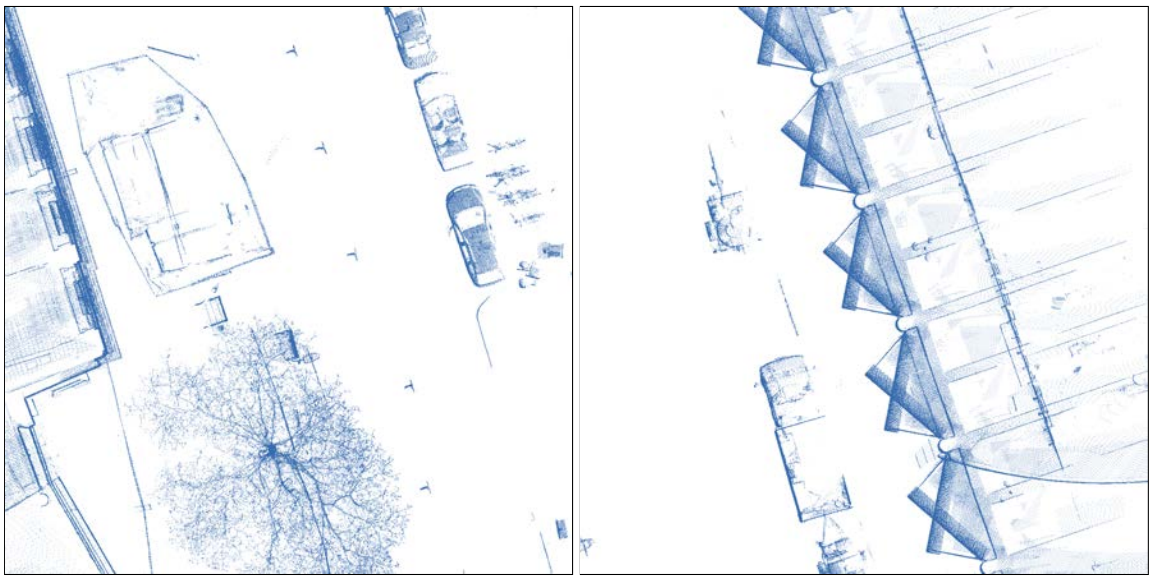
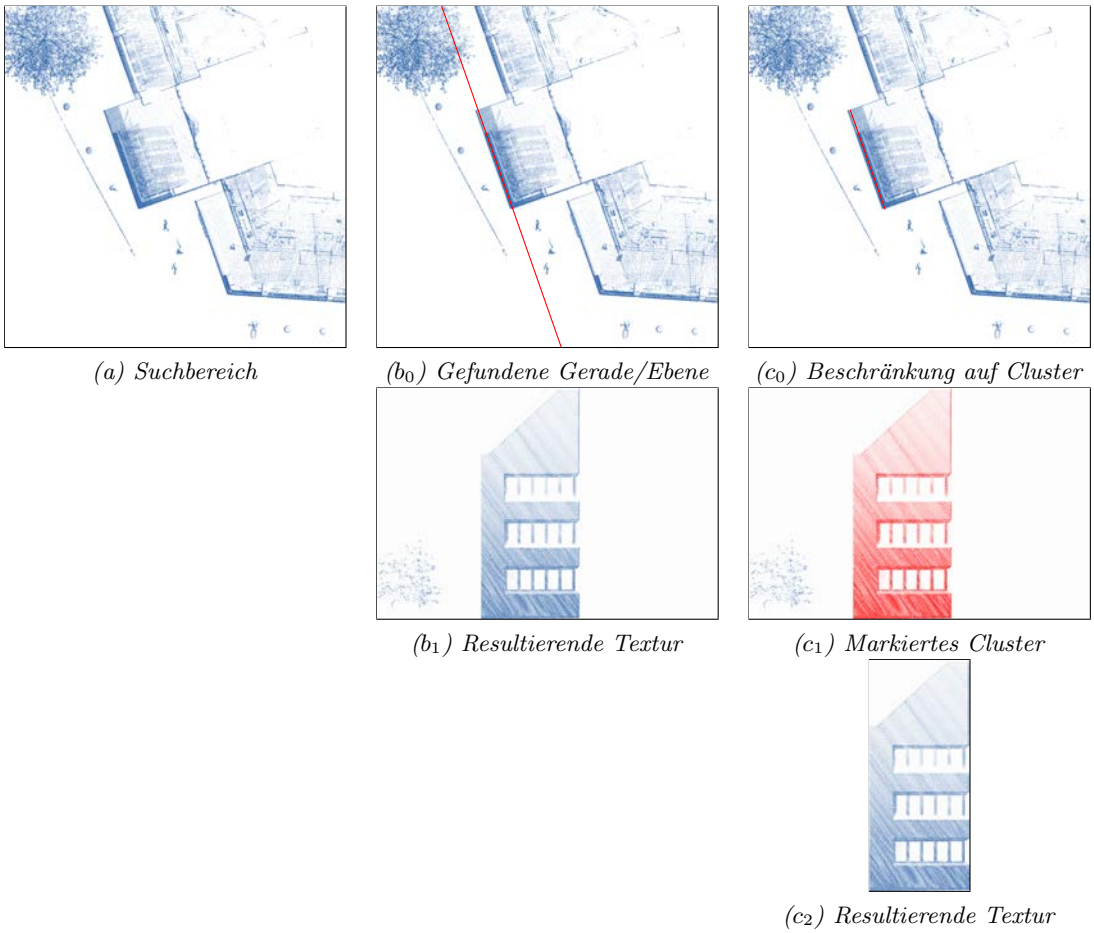


Abbildung 7.3: Punktwolke mit einfacher Fassade (links) und mit komplexer Fassade (rechts).

Um auch Ebenen in komplexeren Fassaden wie aus Abbildung 7.3 (rechts) zuverlässig zu bestimmen, wurde der Prozess der Liniesuche in zwei Schritte unterteilt. Zunächst werden im ersten Schritt Ebenen bzw. Geraden unter bestimmten Randbedingungen gesucht. Erst im zweiten Schritt wird die Suche dann auf beliebige Geraden erweitert. Bei der Betrachtung der komplexen Fassade aus 7.3 wird deutlich, dass die Fassade zwar aus vielen kurzen einzelnen Abschnitten besteht, diese jedoch einer gewissen Regelmäßigkeit unterliegen. Diese Regelmäßigkeit drück sich dadurch aus, dass viele Segmente die gleiche Normale aufweisen. Dieser Sachverhalt wird im ersten Schritt als Randbedingung für das RANSAC Verfahren berücksichtigt. Dafür wird zunächst ein Histogramm der Normalen erstellt. Daraus werden die prominentesten Kandidaten ausgewählt. Nun werden mittels RANSAC Geraden mit genau diesen Normalen gesucht. Ist diese Suche beendet, wird im zweiten Schritt in den verbliebenen Punkten nach beliebig orientieren Geraden gesucht.

Wie das Beispiel in Abbildung 7.4 (b_0) und die daraus resultierende Textur in (b_1) zeigen, genügt das bloße Finden einer Geraden im Suchbereich nicht aus, um unmittelbar die geforderten Eigenschaften (Umfang, Ausdehnung, Planarität und Punktdichte) zu erfüllen. Während Umfang und Ausdehnung der Ebene ausreichend sind, werden die Planaritäts- und Punktdichteeigenschaften nicht erfüllt. Da die gefundene Gerade ebenfalls einen Teil Baumkrone beinhaltet sinkt der Anteil der planaren Punkte unter Umständen unter den festgelegten Schwellwert. Darüber hinaus sind große Teile der Textur leer, da sich die eigentliche Fassade lediglich in der Mitte befindet, was die Punktdichte der Textur unter den festgelegten Schwellwerten fallen lässt. Beide Probleme werden behoben, wenn ein Clustering der Punkte auf der gefundenen Gerade durchgeführt wird und nur die Geradenabschnitte mit entsprechender Punktdichte beibehalten werden. Eine solche Beschränkung für die in 7.4 (b_0) gefundene Gerade ergibt das in 7.4 (c_0) dargestellte Segment, welches in (c_1) noch einmal farblich hervorgehobenen ist. Die finale Textur auf Basis dieses Clusters ist in (c_2) abgebildet und erfüllt nun alle geforderten Eigenschaften.

Ausgehend vom Suchbereich aus Abbildung 7.5 (a) führt obiges Vorgehen beispielsweise zu dem in (b_0) gezeigten Segment und der daraus resultierenden Textur (b_1). Der Großteil der Punkte der Baumkrone sind hier nicht planar. Damit werden nicht alle Voraussetzungen für die Erstellung einer texturierten Ebene erfüllt. Daher genügt ein Clustering auf der Geraden im $\mathbb{R}^2$ nicht aus. Abhilfe schafft hier lediglich ein Clustering in der gesamten (Textur-)Ebene. Von allen gefundenen Clustern



(a) Suchbereich *(b_0) Gefundene Gerade/Ebene* *(c_0) Beschränkung auf Cluster*

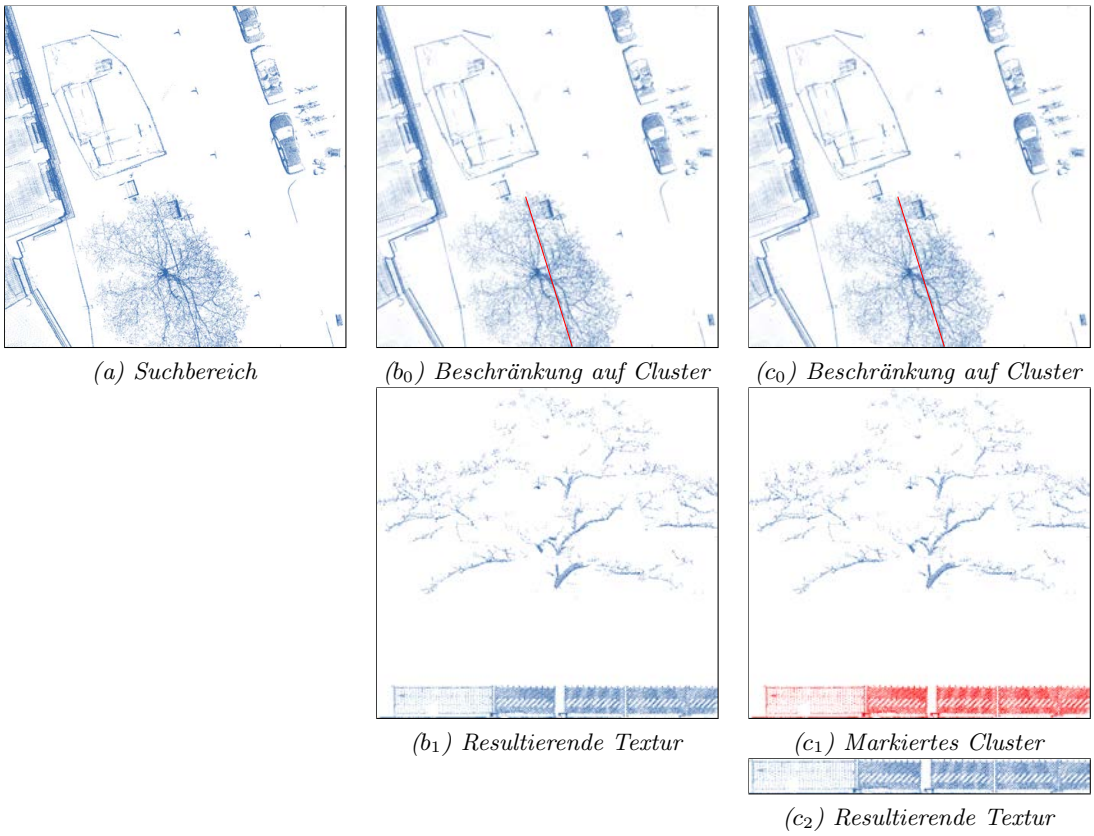
(b_1) Resultierende Textur *(c_1) Markiertes Cluster*

(c_2) Resultierende Textur

Abbildung 7.4: Beschränkung der Textur auf Punktcluster bezüglich der gefundenen Linie

erfüllt lediglich der in (c_1) markierte Bereich die geforderten Bedingungen an Umfang, Ausdehnung, Planarität und Punktdichte. Die resultierende Textur ist in Abbildung 7.5 (c_2) dargestellt.

Neben vertikalen planaren Bereichen lässt sich in urbanen Gebieten der Boden ebenfalls gut durch eine Ebene annähern. Daher wird dieser mittels des in Abschnitt 4.4 beschriebenen Verfahrens extrahiert und die resultierende Ebene auf die genannten Eigenschaften (Umfang, Ausdehnung, Planarität und Punktdichte) hin überprüft. Genügt die Ebene den Bedingungen, so wird eine entsprechende Bodentextur generiert. Die vorliegende Kachelstruktur der Eingangsdaten führt dabei i.d.R. zu direkt aneinandergrenzenden Bodentexturen. Da jede Bodentextur unabhängig von den umliegenden generiert wird können hier kleinere Verwerfungen zwischen Bodentexturen zu angrenzenden Kacheln führen. Dies wird behoben, indem sichergestellt wird, dass alle Bodentexturen exakt der Kachelausdehnung entsprechen (bspw. 25x25). Dies garantiert exakt identische Eckkoordinaten für die x und y Achse. Abschließend werden die Höhen der Eckpunkte als Mittelwert der Höhen der jeweiligen Bodentextur Eckpunkte angeglichen. Grundsätzlich ist diese Problematik auch bei den senkrechten Ebenen, wie Fassaden usw. gegeben. Jedoch fallen solche Unstimmigkeiten dort weniger auf, da zum einen der Blickwinkel in der Regel eher frontal auf die Ebene fällt und zum anderen Fassadenstrukturen ohnehin komplexer ausfallen und somit verbliebene Punkte welche nicht auf die Ebene projiziert wurden etwaige Lücken kaschieren. Nichtsdestotrotz würde eine gemeinsame Ausrichtung nahegelegener Eckpunkte zu einer ansprechenderen Visualisierung führen.



(a) Suchbereich *(b_0) Beschränkung auf Cluster* *(c_0) Beschränkung auf Cluster*

(b_1) Resultierende Textur *(c_1) Markiertes Cluster*

(c_2) Resultierende Textur

Abbildung 7.5: Beschränkung der Textur auf Punktcluster in der gefundenen Ebene

7.1.2 Nachbearbeitung der erstellten Texturen

Beim Rendern der originalen Punktwolke kann die Größe jedes Punktes individuell angepasst werden. Wie in Abbildung 7.6 dargestellt kann durch eine geeignete Wahl der Punktgröße der Eindruck einer nahezu geschlossenen Oberfläche erzeugt werden. Nach der Projektion der Punkte auf eine Textur wird die Punktgröße durch die gewählte Texturauflösung festgelegt und kann zur Renderzeit nicht mehr beeinflusst werden. Soll auch bei durch Texturen repräsentierten Flächen der Eindruck einer geschlossenen Oberfläche erzeugt werden, so müssen hier die transparenten Bereiche zwischen den projizierten Punkten geschlossen werden.

Ein Füllen der transparenten Zwischenräume kann zum einen durch morphologisches Schließen als auch durch Interpolation der vorhandenen Farbwerte erreicht werden. Bei letzterem kann im Wesentlichen auf die gleichen Verfahren wie bei der Interpolation von Höhen, wobei zwischen den Punkten die RGB Werte anstatt der Höhe interpoliert wird, zurückgegriffen werden. Aus den diversen möglichen Verfahren werden an dieser Stelle das morphologische Schließen, als auch zwei Verfahren zur Farbinterpolation vorgestellt und potentielle Verwendungsszenarien diskutiert.

Die morphologische Operation *Schließen* (engl. Closing) bezeichnet das sequentielle Ausführen der morphologischen Basisoperationen Dilatation und Erosion und schließt Löcher im Bild. Bei der Dilatation wird die Ausdehnung jedes Punktes/Pixels ähnlich wie beim Rendern der Punktwolke erhöht (dilatiert). Im Falle der erzeugten Texturbilder werden dabei alle nicht transparenten Pixel auf angrenzende transparente Bereiche um einen gegebenen Wert ausgedehnt. Bei der anschließenden Erosion werden alle verbliebenen transparenten Pixel um denselben Wert auf die nicht transparenten Bereiche ausgedehnt. Wie Abbildung 7.7 (Mitte) zeigt führt ein großer Ausdehnungswert (Kernelgröße) zu deutlich erkennbarem Weichzeichnen. Visuell ansprechendere Ergebnisse, vgl. Abb. 7.7 (rechts), liefert ein iteratives Dilatieren. Um den gleichen Schließungseffekt eines größeren Kernels

Abbildung 7.6: Durchlässige Oberfläche bei Punktgröße von einem Pixel (links) und Eindruck einer geschlossenen Oberfläche durch Erhöhung der Punktgröße (rechts)

zu erreichen, braucht lediglich entsprechend oft iteriert zu werden. Im Falle eines Kernels der Größe zehn sind somit zehn Iterationen mit einem Kernel der Größe eins notwendig.

Abbildung 7.7: Durchlässige Oberfläche bei auf eine Textur projizierten Punkten (links) Zwischenraumschließung mittels morphologischer Operation Schließung mit Kernelgröße von 10 (Mitte) und Kernelgröße 1 mit 10 Iterationen (rechts).

Ein gängiger Ansatz bei der Interpolation von Höhenwerten ist die Erstellung eines *Dreiecksnetzes*, bspw. nach dem Verfahren der Delaunay-Triangulation, und der anschließenden linearen Interpolation innerhalb der einzelnen Dreiecke. Problematisch ist jedoch, dass die gesamte konvexe Hülle der projizierten Punkte gefüllt wird. Damit werden, wie in Abbildung 7.8 (rechts) gezeigt, Regionen gefüllt welche eigentlich transparent bleiben sollten. Abhilfe schafft hier die Einführung eines Schwellwertes, der definiert, welches Dreieck des Netzes gefüllt werden sollten und welches nicht. Die Länge der längsten Kante eines Dreiecks kann bspw. dafür herangezogen werden. Überschreitet diese Kante den vorgegebenen Schwellwert, so wird die Fläche innerhalb des zugehörigen Dreiecks nicht gefüllt. Abbildung 7.10 (links) zeigt das Interpolationsergebnis unter Verwendung eines Schwellwertes von 16 Pixeln bei einer Bilddimension von 767x640 Pixeln. Im visuellen Vergleich mit der Punktgrößenbasierten Interpolation aus Abbildung 7.6 (rechts) fällt auf, dass die Ecken der Fenster nicht mehr rechtwinklig sind. Dem kann durch ein Verringern des Schwellwertes für die maximale Kantenlänge begegnet werden. Dies kann jedoch, wie in Abbildung 7.8 (Mitte) gezeigt, dazu führen, dass nicht die gesamte Oberfläche geschlossen wird.

Ausgehend von weitestgehend rechtwinkligen Strukturen, stellt die lineare Interpolation zwischen *horizontal* bzw. *vertikal benachbarten Pixeln* eine weitere Möglichkeit zum Füllen der Zwischenräume dar. Dabei muss wieder angegeben werden, wie groß die zu interpolierenden Abschnitte sein dürfen. Darüber hinaus sollte die Interpolation zum einen iterativ mit bis zum Schwellwert steigenden Abständen durchgeführt, als auch abwechselnd horizontal und vertikal interpoliert werden. Dies vermeidet weitestgehend das Auftreten von etwaigen Alaisingeffekten. Abbildung 7.10 (Mitte) zeigt

Abbildung 7.8: Durchlässige Oberfläche bei auf eine Textur projizierten Punkten (links) Zwischenrauminterpolation mittels Delaunay Dreiecksnetz mit Schwellwert für maximale Kantenlänge, bei zu gering gewähltem Schwellwert (Mitte) und zu hoch gewähltem Schwellwert (rechts).

das Ergebnis der horizontal und vertikal Interpolation mit einem maximalen Abstand von 16 Pixeln. Das Ergebnis ist näher an dem aus Abbildung 7.6 (rechts), da die rechten Winkel erhalten bleiben.

Die Wahl eines geeigneten Interpolationsverfahrens für die zu erstellenden Texturen hängt von mehreren Faktoren ab. Ist das Einsatzszenario laufzeitkritisch, sollte demnach das schnellste Verfahren gewählt werden. Das Diagramm aus Abbildung 7.9 stellt die Laufzeiten der Verfahren gegenüber. Dabei lässt sich die Interpolationen zwischen horizontal und vertikal benachbarten Pixeln als schnellstes Verfahren identifizieren. Die morphologische Operation *Schließen* benötigt bereits fünf mal solange wohingegen die Laufzeit der Delaunay basierten Interpolation schon fast das 20-fache Beträgt.

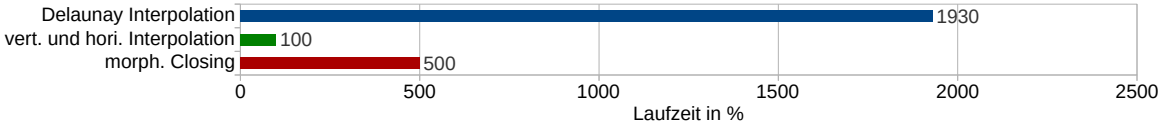


Abbildung 7.9: Diagramm der Laufzeiten der vorgestellten Interpolationsarten in Prozent bezogen auf die beste Laufzeit.

Steht die visuelle Qualität der Ergebnisse im Vordergrund, so lassen sich die Verfahren anhand der Gegenüberstellung in Abbildung 7.10 beurteilen. Geht man von Texturen zu anthropogenen Objekten aus, so scheidet die Delaunay basierte Interpolation aus, da hier i.d.R. keine rechten Winkel (siehe Fenster) erzeugt werden. Die Ergebnisse der Interpolation benachbarter Pixel und des morphologischen Schließens sind visuell nahezu identisch. Der Unterschied liegt hier eher im Farbumfang der resultierenden Texturen. Während bei der Interpolation benachbarter Pixel Zwischenfarben interpoliert (erzeugt) werden, bleibt die Anzahl der Farben beim Schließen gleich, da die vorhandenen Pixelfarben lediglich expandiert werden. Ersteres hat den Vorteil, dass etwaige Übergänge etwas glatter verlaufen. Letzteres kann u.U. zu einem Speichervorteil führen, da der Farbumfang (Anzahl an genutzten Farben) im Gegensatz zu der Interpolationsvariante geringer ist. Dies erlaubt manchen Fällen eine Indizierung der Farben und das Nutzen einer Farbpalette möglich, was der Speichergröße der resultierenden Textur zugute kommt.

Nach dem Füllen der Zwischenräume verbleiben je nach gewählter Interpolationsweite unter Umständen immer noch transparente Bereiche in der Textur. Die Ursachen für die fehlende Farbinformation können vielfältig sein. Bei der zuvor betrachteten Beispielfassade aus Abbildung 7.10 wurde der Laserstrahl nicht vom Fensterglas reflektiert, weshalb es dort keine Scanpunkte gibt. Verdeckungen führen häufig ebenfalls zu fehlenden Scanpunkten und damit zu transparenten Bereichen auf den hinter dem Verdecker liegenden Objekten. Unabhängig von der Ursache ist es vom visuellen Standpunkt her jedoch wünschenswert, diese Bereiche zu füllen.

Grundsätzlich könnte dies durch die Erhöhung der Interpolationsweite erreicht werden. Dies führt jedoch bei allen drei diskutierten Ansätzen zu visuell nicht ansprechenden Resultaten. Daher wird hier an anderer Ansatz genutzt. Zunächst müssen die zu füllenden Bereiche identifiziert werden. Dafür wurde das Verfahren zum Finden von Zusammenhangskomponenten von Rosenfeld und Pfaltz (1966)

Abbildung 7.10: Eindruck einer geschlossenen Oberfläche durch Füllen der Zwischenräume mittels linearer Interpolation innerhalb von Dreiecksflächen (links), horizontaler/vertikaler benachbarter Pixel (Mitte) und der morphologischen Operation Schließen (rechts).

in Verbindung mit der Union-Find Struktur von Tarjan (1979) umgesetzt. Nach der Ermittlung der Löcher als Zusammenhangskomponenten werden die nicht transparenten Randpixel jeder Komponente mittels α-shapes bestimmt. Dabei werden alle Zusammenhangskomponenten ignoriert, welche den Bildrand berühren. Für die Randpixel jedes Loches wird nun eine 2D-Farbinterpolationsfunktion approximiert. Unter Verwendung dieser Funktionen werden abschließend alle transparenten Pixel der Löcher eingefärbt. Da üblicherweise der entstandene Farbverlauf sehr glatt und gleichmäßig ist, wird noch ein minimales Rauschen über die eingefärbten Pixel gelegt um eine natürlichere Optik zu erhalten.

Abbildung 7.11 zeigt den beschriebenen Ansatz am Beispiel der Fassadentextur der vorangegangenen Abbildungen. Ausgehend von der Textur (links) werden die transparenten Zusammenhangskomponenten (Mitte) ermittelt. Abbildung 7.11 (rechts) zeigt das Ergebnis.

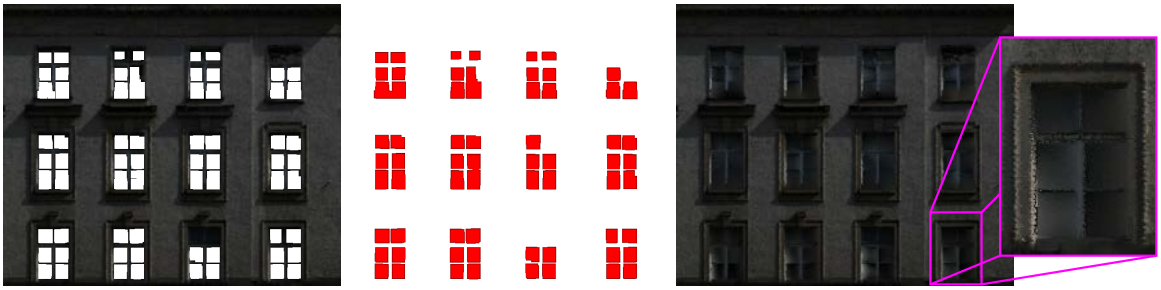


Abbildung 7.11: Fassadentextur mit transparenten Fensterbereichen (links) und die daraus ermittelten transparenten Zusammenhangskomponenten (Mitte) und das Ergebnis der Farbinterpolation (rechts).

Die häufigste Ursache für Löcher in einer Bodentextur sind Verdeckungen. Abbildung 7.12 (links) zeigt ein typisches Beispiel. Die Löcher auf der Fahrbahn wurden durch aufgestellte Warnbaken verursacht. Darüber hinaus sind parkende Autos ebenfalls häufig Ursache für durch Verdeckung entstandene Löcher. Wie schon beim vorangegangen Beispiel sind die ermittelten Zusammenhangskomponenten (Mitte) und das Ergebnis (rechts) abgebildet. Wie der markierte Bereich verdeutlicht, eignet sich das gezeigte Verfahren insbesondere für Bereiche mit geringem Farbumfang, wie beispielsweise der Fahrbahn. Bereiche mit größerem Farbumfang und ungleichmäßigerer Verteilung der Farben über die Randpixel führen bei dieser Art der Interpolation zu deutlich sichtbaren Abgrenzungen bzw. Übergängen. Wie gehabt werden Bereiche, welche den Rand berühren nicht gefüllt.

7.1.3 Effiziente Verwaltung von Texturen

Jede Textur eines einzelnen Objektes in ein eigenes Bild zu speichern ist sehr ineffizient, da entsprechend viele Bildreferenzen verwaltet, gespeichert und zur Visualisierung wieder geladen werden

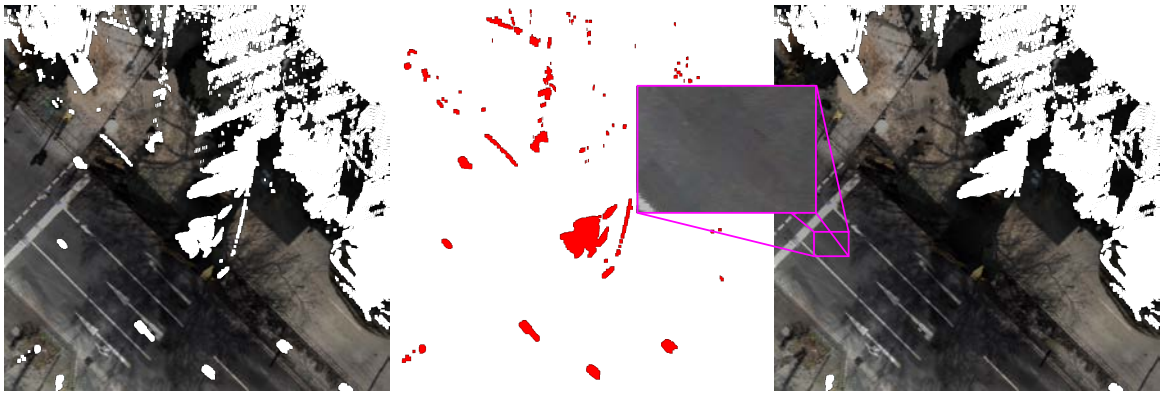


Abbildung 7.12: Löchrige Textur einer Bodenebene (links) und die daraus ermittelten transparenten Zusammenhangskomponenten (Mitte) und das Ergebnis der Farbinterpolation (rechts).

müssen (vgl. Perdeck (2011)). Daher ist es in der Regel sinnvoll, mehrere Bilder/Texturen zusammengefasst in einem *Texturatlas* zu speichern. Dafür wird ein großes Texturbild erzeugt, wobei die eigentlichen Texturen in diesem verteilt werden und die Position der Textur (die Texturkoordinaten) vermerkt wird.

In diesem Kontext sind zwei Fragestellungen relevant. Zum einen, welches sind die Dimensionen des Texturatlases, so dass alle Bilder dort überlappungsfrei platziert werden können und möglichst kein Platz verschwendet wird? Zum anderen, wie müssen die Bilder im Atlas angeordnet werden, so dass alle in einem Atlas mit gegebener Dimension Platz finden?

Verallgemeinert ist die beschriebene Problemstellung ein *Behälterproblem* (vgl. Korte und Vygen (2008)) und damit NP-Schwer. Wie Korf (2003) zeigt, kann unter Berücksichtigung der existierenden Randbedingungen dennoch eine effiziente Lösung gefunden werden.

Das im Rahmen dieser Arbeit implementierte Verfahren zur Bestimmung der minimalen Dimension des Texturatlas' hält sich dabei im Wesentlichen an den in Korf (2003) vorgeschlagenen Ansatz. Dabei begrenzt sich der zweidimensionale Suchbereich wie folgt. Die minimale Breite des Atlas' entspricht der des breitesten unterzubringenden Bildes. Gleiches gilt für die Höhe. Darüber hinaus entspricht die maximale Breite der Summe der Breiten aller unterzubringenden Bilder, was auch für die Höhe gilt. Abschließend muss die Fläche des Atlas' größer gleich der Summe der aller Bilder sein. Für den so eingeschränkten Suchbereich muss nun überprüft werden, ob eine überlappungsfreie Anordnung der Bilder für die jeweilige Atlasdimension existiert. Diese Überprüfung kann übersprungen werden, sollte die Fläche der gewählten Atlasdimension die Minimalfläche unterschreiten oder schlechter/größer sein, als das der bis dato besten Atlasdimension.

Der Grundgedanke für die Bestimmung einer überlappungsfreien Anordnung wurde ebenfalls von Korf (2003) übernommen. Der vorgeschlagene Ansatz beruht im Wesentlichen darauf, dass die der Größe nach sortierten Bilder eins nach dem anderen dem Atlas hinzugefügt werden. Als Position wird der am weitesten oben und links gelegene freie Bereich gewählt. Im Rahmen dieser Arbeit wurde dafür die im folgenden beschriebene baumbasierte Implementierung erstellt, welche den beschriebenen Ansatz effizient umsetzt. Abbildung 7.13 gibt dabei einen Überblick über die Abbildung des Ansatzes zur Bestimmung einer überlappungsfreien Anordnung auf die Traversierung einer Binärbaumstruktur.

Beim implementierten Baum handelt es sich um einen Binärbaum. Die Knoten des Baumes besitzen dabei eine gewisse Ausdehnung (der zur Verfügung stehende Platz), sowie Referenzen auf zwei Kindknoten. Ausgehend von einem leeren Wurzelknoten, dessen Ausdehnung der des Texturatlas' entspricht, werden die Texturbilder wie folgt hinzugefügt. Handelt es sich um ein Blatt, wird diesem Knoten das Bild zugeordnet, sofern dieser noch keine Zuordnung besitzt und die Ausdehnung des Knotens groß genug ist. Konnte das Bild dem Knoten zugeordnet werden, so werden die zwei Kindknoten initialisiert, welche die verbliebenen Teilflächen repräsentieren. Die Traversierung des Baumes entspricht dabei einer Tiefensuche. Wurde der komplette Baum traversiert, ohne

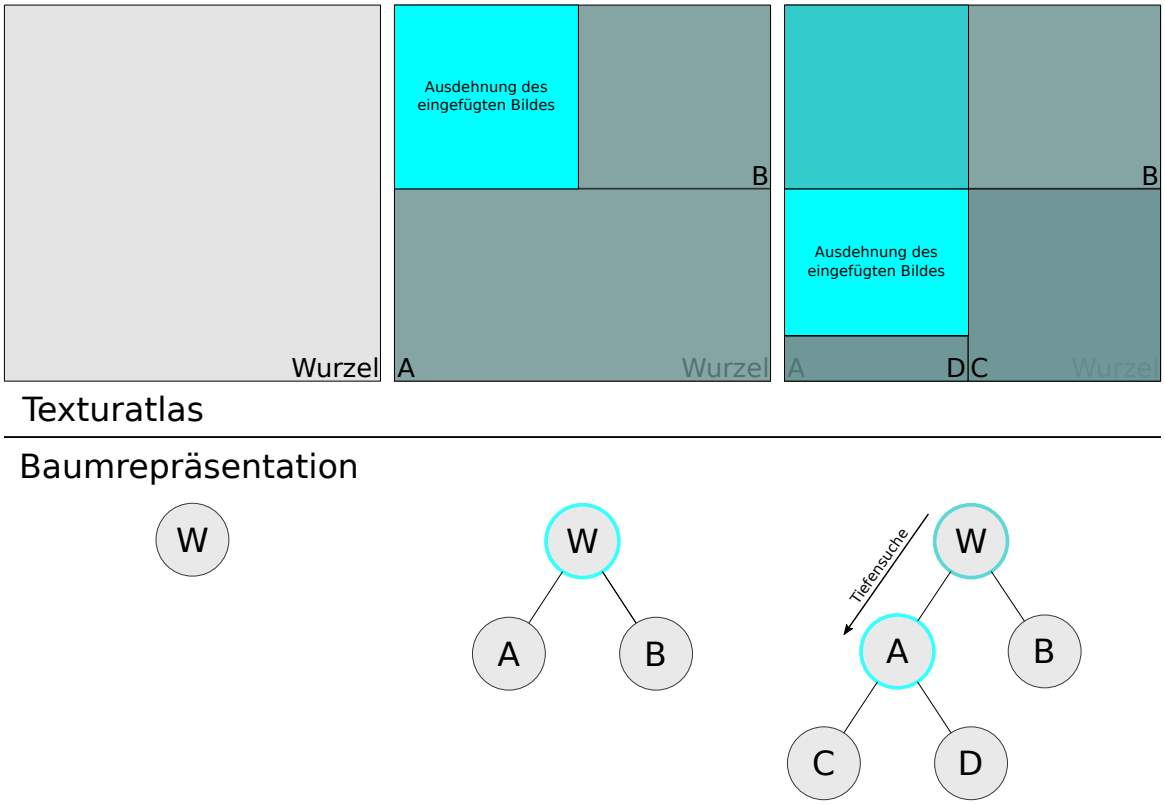


Abbildung 7.13: Baumbasiertes Finden einer überlappungsfreien Anordnung von Bildern in einem Texturatlas.

dass das einzufügende Bild einem Blatt zugeordnet werden konnte, so kann auf diesem Wege keine überlappungsfreie Anordnung für die gegebenen Dimensionen und Bildmenge gefunden werden. Somit muss eine neue Dimension gewählt und die unterzubringenden Bilder erneut hinzugefügt werden.

Abbildung 7.14 (links) zeigt das Ergebnis der Anwendung des Verfahrens auf die generierten Texturen einer 25x25 Meter Kachel. Da viele der Einzeltexturen transparente Bereiche aufweisen, fällt eine Beurteilung der erzeugten Anordnung relativ schwer. Zu diesem Zweck zeigt Abbildung 7.14 (rechts) eine Farbcodierung der einzelnen Texturen. Der Anteil an *verschwendetem* Platz beträgt im gezeigten Beispiel 5%, welches einen guten Wert darstellt. Betrachtet man darüber hinaus den persistenten Speicherbedarf (als komprimierte Datei auf einem Datenträger) des Bildes, so sinkt dieser Anteil durch die verwendete Kompression nochmals, wobei der verbleibende Anteil durch die Einsparung der Header-Informationen der jeweiligen Einzelbilder nahezu komplett egalisiert wird. Nichtsdestotrotz ist es wichtig, den verschwendeten Anteil möglichst gering zu halten, da bei einer Visualisierung der Atlas unkomprimiert im Grafikspeicher vorgehalten werden muss.

7.1.4 Erhöhung der Speichereffizienz

Soll das Ersetzen der Punkte durch die Projektion auf eine Textur zur Erhöhung der Speichereffizienz genutzt werden, so muss zunächst bestimmt werden, in welchen Situationen sich der benötigte Speicherplatz auch tatsächlich verringert. Ersetzt eine Textur nur wenige Punkte, so kann sich der benötigte Speicherbedarf unter Umständen sogar erhöhen. Betrachtet man den Speicherbedarf eines Texturpixels von vier Byte (je einer pro RGBA-Farbkanal) und den Speicherbedarf eines Punktes von 16 Byte (je vier Byte pro Koordinate XYZ und noch einmal vier Byte für die Farbe RGBA), so ergibt sich bei eine Reduktion auf die Farbe ein Verhältnis von 1:4. Dies bedeutet, dass eine Textur mindestens zu einem Viertel aus nicht transparenten Pixeln bestehen muss, um einen geringeren Speicherbedarf als die ersetzten Punkte aufzuweisen.

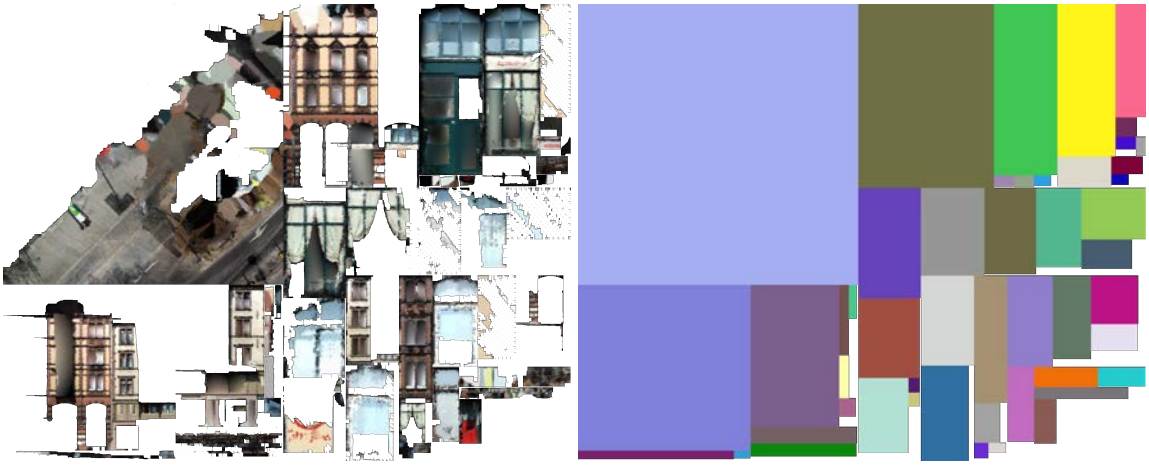


Abbildung 7.14: Beispiel für einen erstellten Texturatlas (links), Farbcodierung separater Texturen (rechts).

Dieses Texturfüllverhältnis $fill_{ratio} = \frac{1}{4}$ gilt es bei der Bestimmung der Texturauflösung zu berücksichtigen. Seien w und h die Realweltdimensionen der gefundenen Fläche und n die Anzahl der zur Fläche gehörenden Punkte, dann ergibt sich ein Texturauflösungsverhältnis res_{ratio} nach Gleichung 7.1.

$$res_{ratio} = \sqrt{fill_{ratio} \frac{w * h}{n}} \tag{7.1}$$

Beispiel:

$$\begin{array}{lcl} w & = & 1m \\ h & = & 2m \\ n & = & 200Pt \\ fill_{ratio} & = & \frac{1}{4}\frac{Pt}{Pixel^2} \end{array}$$

$$res_{ratio} = \sqrt{\frac{1}{4}\frac{Pt}{Pixel^2}\frac{1m*2m}{200Pt}} = \sqrt{0.0025\frac{m^2}{Pixel^2}} = 0.05\frac{m}{Pixel}$$

Das Texturauflösungsverhältnis lässt sich interpretieren als die Länge (in Breite und Höhe) welche von einem Pixel im Bild repräsentiert wird. Im gezeigten Beispiel sind dies 5cm. Demnach sollte die Textur aus dem gezeigten Beispiel eine maximale Höhe $H_{max} = \frac{h}{0.05} = 40$ und maximale Breite $W_{max} = \frac{w}{0.05} = 20$ aufweisen. Wird eine höhere Auflösung gewählt, so entsteht ein schlechteres Texturfüllverhältnis als 1:4, was wiederum zu einer Erhöhung des Speicherbedarfs führt. Somit lässt sich für jede gefundene Ebene das passende Texturauflösungsverhältnis bestimmen. Je nach vorliegenden Ebenenparametern kann das ermittelte Auflösungsverhältnis starken Schwankungen unterliegen. Die Spanne erstreckt sich dabei von sehr grob aufgelöst (mehrere Dezimeter pro Pixel) bis hin zu sehr fein aufgelöst (wenige Millimeter pro Pixel). Ersteres führt zu optisch wenig ansprechenden Texturen, wohingegen letzteres zu sehr großen Texturen führen kann, welche nicht mehr von der Grafikhardware unterstützt werden. Daher wird die erlaubte Spanne begrenzt. Als gröbstes Auflösungsverhältnis wurden zehn Zentimeter pro Pixel gewählt um stark weichgezeichnete Texturen zu vermeiden. Die eingangs aufgezeigte Punktdichte Bedingung für die Identifikation der Ebenen ist dabei von diesem Wert abgeleitet. Das feinste Auflösungsverhältnis wird dynamisch angepasst, so dass die resultierende Textur eine Größe von 1024x1024 nicht überschreitet.

Tabelle 7.1 zeigt die Statistik für die Anwendung des beschriebenen Verfahrens für ein Beispielprojekt, bestehend aus 1456 Kacheln, welche 316 Millionen Punkte umfassen. Wie die Statistik zeigt werden fast die Hälfte aller Punkte auf Bodentexturen projiziert, wohingegen lediglich ein Zehntel der Punkte vertikalen Ebenen zugeordnet wurden. Von sämtlichen durch RANSAC detektierten Flächen erfüllten mehr als die Hälfte nicht das eingangs gestellte Planaritätskriterium ein weiteres Drittel war zu klein. Insgesamt erfüllten lediglich 7% der gefunden 77.390 Flächen die gestellten Kriterien.

Durch die Festlegung des Texturfüllverhältnis auf $fill_{ratio} = \frac{1}{4}$ ist das Speicherverhältnis von ersetzten Punktdaten (*Anzahl Bytes ersetzter Punktdaten*) und erzeugten Bilddaten (*Anzahl Bytes*

Anzahl Punkte:	316.025.366	316M	100%
Anzahl ersetzter Bodenpunkte:	148.426.834	148M	47%
Anzahl ersetzter Flächenpunkte:	32.061.017	32M	10%
Anzahl Kacheln:	1456		100%
Anzahl erstellter Bodentexturen:	793		54%
Anzahl gefunden vertikaler Flächen:	77.390		100%
Anzahl zu kleiner Flächen (Umfang od. Ausdehnung):	27.129		35%
Anzahl zu dünner Flächen (Punktdichte):	462		$< 1\%$
Anzahl nicht planarer Flächen (Planarität):	44.499		57%
Anzahl erstellter Flächentexturen:	5.300		7%
Anzahl Bytes Punktdaten:	5.056.405.856	4,7GB	100%
Anzahl Bytes ersetzter Punktdaten:	2.887.805.616	2,7GB	57%
Anzahl Bytes Bilddaten (Roh):	2.906.109.964	2,7GB	101%
Anzahl Bytes Atlas Bilddaten (Roh)	4.196.009.504	3,9GB	145%
Anzahl Bytes Atlas Bilddaten (Deflate):	1.248.948.547	1,2GB	43%
Anzahl Bytes Atlas Bilddaten (optimiert):	1.000.005.659	0.9GB	35%

Tabelle 7.1: Statistik der Erstellung Hybrider Modelle.

Bilddaten (Roh)) nahezu Identisch. Durch das Zusammenfassen der Einzelbilder zu Texturatlanten erhöht sich der Umfang der Bilddaten zunächst auf das Eineinhalbfache. Da jedoch die erstellten Texturen im PNG Format (G. Randers-Pehrson (1999)) gespeichert werden, welches das verlustfreie Kompressionsverfahren Deflate (Deutsch (1996)) nutzt, verringert sich der Umfang der Atlasdaten auf weniger als die Hälfte (*Anzahl Bytes Atlas Bilddaten (Deflate)*) der ursprünglichen Punktdaten. Das von PNG genutzte Deflate Verfahren besitzt diverse Parameter, welche die resultierende Kompressionsrate beeinflussen. Zu Gunsten einer hohen Kompressionsgeschwindigkeit werden jedoch nicht immer die optimalen Parameter gesucht, bzw. verwendet. Programme wie AdvPNG (Mazzoleni (2014)), PNGOUT (Silverman (2015)) oder Googles Zopfli (Vandevenne (2013)) optimieren die Kompressionsrate auf Kosten der Geschwindigkeit. Wird eine hohe Kompressionsrate benötigt, wie bspw. bei der in Kapitel 8.1 vorgestellten webbasierten Visualisierung, so kann der Speicherbedarf wie im Beispiel gezeigt (*Anzahl Bytes Atlas Bilddaten (optimiert)*) nochmals um einige Prozent auf etwa ein Drittel des Speicherbedarfs der ersetzten Punkte gesenkt werden.

7.1.5 Level of Detail

Wie bereits im Grundlagen Abschnitt 2.2.5 erläutert, werden Level of Detail (LOD) Techniken zur Erhöhung der Rendergeschwindigkeit angewendet. Dies ermöglicht auch in weniger performanten Umgebungen, bspw. innerhalb von Web-Browser Anwendungen (Software), älteren GPUs (Grafik-Hardware) oder langsamer Datenübertragungsraten (Netzwerk-Hardware), eine flüssige Darstellung des Modells. LOD-Techniken umfassen dabei in der Regel drei Komponenten. Zum einen wird eine *Datenstruktur* definiert, welche die unterschiedlichen Detailstufen eines Modells abbildet. Eine solche Datenstruktur wird üblicherweise mittels eines *Vereinfachungsverfahrens* im Vorfeld aufgebaut und kann im Anschluss zum Rendern des Modells genutzt werden. Zum Zeitpunkt des Renderings des Modells müssen auf Basis eines *Verfahrens zu Bestimmung der genutzten Detailstufe* die zu nutzenden LOD-Modelldaten aus der Datenstruktur ermittelt werden. Im Folgenden wird zunächst auf die ersten beiden Komponenten, LOD-Datenstruktur und Vereinfachungsverfahren, eingegangen. Das genutzte Verfahren zur Bestimmung der Detailstufe ist stark vom verwendeten Renderframework und der Anwendungsumgebung abhängig und wird daher erst in den Abschnitten 8.1.1 und 8.1.3 beschrieben.

Da es sich im Falle der hybriden Modelle sowohl um Punktwolken- als auch um Bilddaten handelt, muss für beide Repräsentationsformen eine LOD Strategie entworfen werden. Ausgehend von den drei aufgezeigten LOD-Techniken, diskret, kontinuierlich und blickwinkelabhängig wurde für die texturierten Objektteile (Bilddaten) eine diskretes LOD mittels des *Mip Mapping* Verfahrens nach Williams (1983) eingesetzt. Im Falle der Punktwolkendaten kommt eine Eigenentwicklung zum Ein-

satz, welche eine blickwinkelabhängige LOD-Technik umsetzt. Eine Kachel mit allen enthaltenden Punkten und Texturen bildet dabei das grundlegende Objekt, für welches unterschiedliche Detailstufen erzeugt werden sollen.

Für das Mip Mapping der Bilddaten wird für das original Texturbild eine Bildpyramide, als LOD-Datenstruktur, entsprechend einer vorgegebenen Anzahl an diskreten LOD-Stufen generiert. Das Vereinfachungsverfahren erzeugt dabei für jede Stufe ein eigenständiges Texturbild, wobei die Kantenlänge zur vorherigen Stufe jeweils halbiert wird. Somit wächst der zusätzlich benötigte Speicherbedarf um weniger als ein Drittel gegenüber dem Originaltexturbild. Die erstellte MipMap Bildpyramide für den Texturatlas einer Beispiel-Kachel ist in Abbildung 7.16 (links) dargestellt. Abbildung 7.15 zeigt das gerenderte Texturmodell in vier LOD Stufen. Während die Texturen von LOD0 und LOD1, aufgrund der niedrigen Texturauflösung, noch deutlich weichgezeichnet wirken, erwecken LOD3 und LOD4 einen fast fotorealistischen Eindruck.

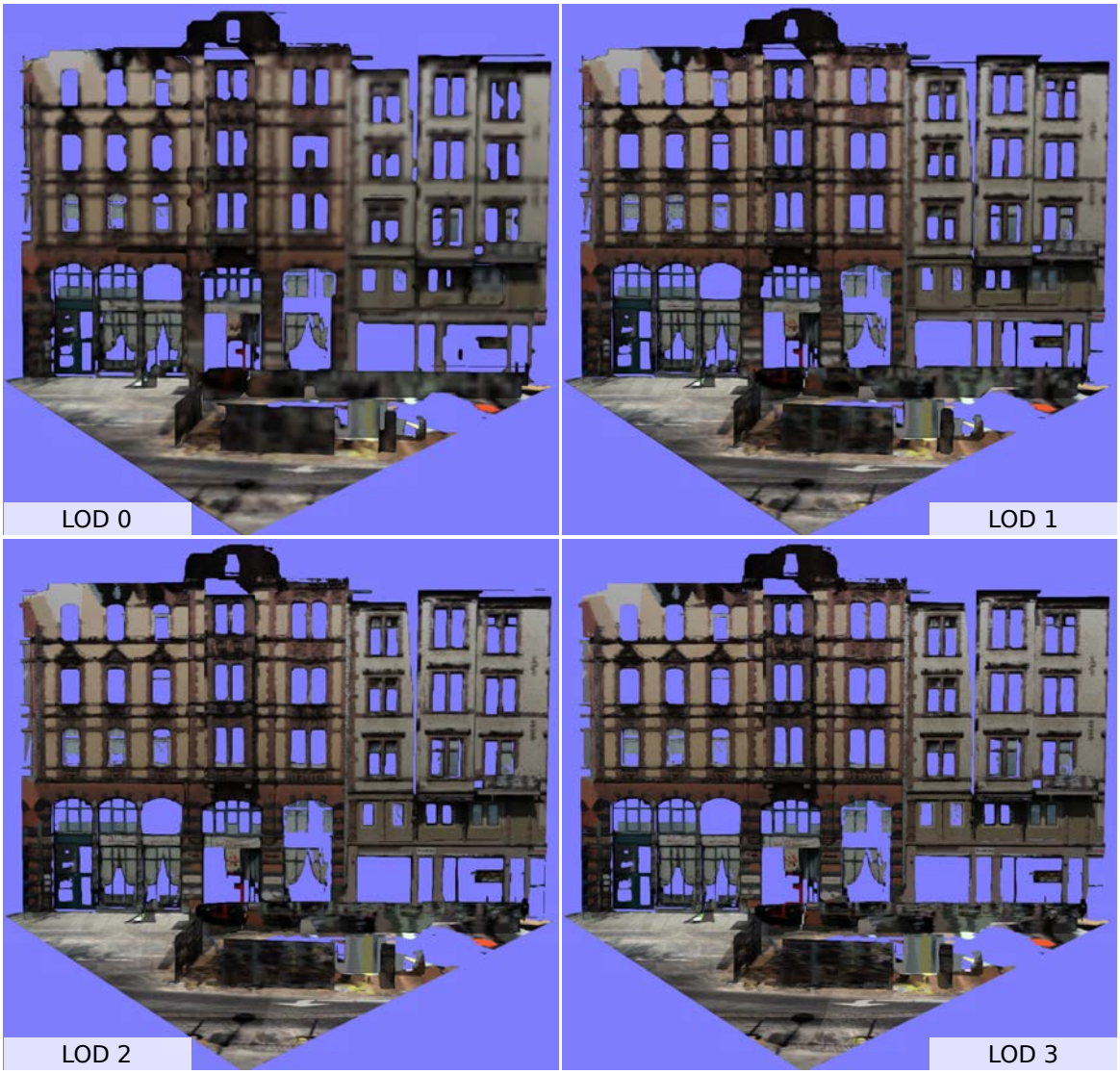


Abbildung 7.15: Gerenderte texturierte Modelle in den LOD Stufen LOD0, LOD1, LOD2 und LOD3.

Wie eingangs erläutert wurde für die Punktwolkendaten eine eigene blickwinkelabhängige LOD-Technik realisiert. Als zugrundeliegende Datenstruktur wird hier ebenfalls eine Pyramidenstruktur genutzt. Dies vereinfacht die spätere Kombination der LOD-Punktdaten und LOD-Bilddaten. Im Gegensatz zur diskreten LOD-Technik umfasst in diesem Fall eine LOD-Stufe nicht das komplette Modell, sondern lediglich einen inkrementellen Teil. Somit werden die bereits geladenen Punktdaten

aus niedrigen LOD-Stufen nicht durch neue detaillierte ersetzt, sondern kontinuierlich mit jeder LOD-Stufe erweitert. Dabei umfasst die Punktmenge einer LOD Stufe alle Punkte der unteren Stufen, wobei lediglich der inkrementelle Teil nachgeladen werden muss. Wurden also bereits alle Punkte für LOD0 geladen, genügt es für die Darstellung von LOD1 die Menge an Punkten, die nicht bereits in LOD0 enthalten ist, zu laden.

Um eine Punktwolkenkachel blickwinkelabhängig rendern zu können, muss die Pyramidenstruktur entsprechend erweitert werden. Dafür wird die Kachel in Teilbereiche untergliedert. Viele Teilbereiche und viele Pyramidenstufen ermöglichen eine feine Granularität (bezüglich blickwinkelabhängiger und inkrementeller Darstellung), erzeugen jedoch einen hohen Verwaltungsaufwand und Berechnungsaufwand zur Laufzeit (engl. overhead). Daher gilt es hier ein gutes Maß zwischen Granularität und erzeugtem Overhead zu finden. Um die Anzahl and Pyramidenstufen (entspricht der Anzahl der LOD-Stufen) möglichst gering zu halten, kann der Detailgrad mit jeder Stufe exponentiell gesteigert werden. Analog dazu kann die Anzahl an blickwinkelrelevanten Teilbereichen ebenfalls exponentiell gesteigert werden. Ausgehend von der kleinsten LOD-Stufe (LOD0) mit genau einen Teilbereich (gesamte Kachel) ergibt sich nach dem gezeigten Schema für die nächste LOD-Stufe (LOD1) eine Unterteilung in vier Teilbereiche (mit halbierter Kantenlänge), wobei jeder der vier Teilbereiche die gleiche Anzahl an Punkten enthält wie der gesamte Bereich der kleinen LOD-Stufe, in diesem Fall LOD0. Somit wächst die Punktanzahl und damit auch der Detailgrad mit jeder LOD Stufe exponentiell an. Das daraus abgeleitete Vereinfachungsverfahren wählt dafür zufällig Punkte aus, die in die jeweiligen Teilbereiche fallen. Um kompatibel zur diskreten LOD-Struktur der Bilddaten zu bleiben wird hier ebenfalls eine feste Anzahl an LOD-Stufen vorgegeben. Um LOD-Stufen, bzw. deren Teilbereiche mit sehr wenigen Punkten zu vermeiden, werden zwei Schwellwerte definiert. Zunächst wird eine Mindestpunktanzahl pro Bereich gefordert. Dies reduziert die Anzahl an LOD-Stufen für eine Kachel bzw. Teilbereiche einer Kachel mit nur wenigen Punkten. Abschließend werden Teilbereiche, welche weniger als 20% dieses Punktumfangs besitzen dem übergeordneten Bereich der nächst kleineren LOD-Stufe zugeordnet.

Beispiel: Angenommen eine Punktwolkenkachel beinhaltet 10.000 Punkte und soll nach dem obigen Schema in die beschriebene LOD Struktur überführt werden. Ferner sollen, der Einfachheit halber, lediglich zwei LOD Stufen (LOD0 und LOD1) generiert werden. Als Mindestpunktanzahl seien 5.000 Punkte pro Teilbereich gegeben. Zunächst wird die Anzahl an zu erstellenden Teilbereichen bestimmt. Für LOD0 wird ein Bereich, für LOD1 werden vier Teilbereiche erzeugt (vgl. Abbildung 7.16 magenta und grün), somit ergibt sich fünf Teilbereiche, auf die die 10.000 verteilt werden müssen. Dies ergibt 2.000 Punkte pro Bereich, was unter der festgelegten Mindestpunktanzahl von 5.000 liegt und daher auf diesen Wert angehoben wird. Für die Erstellung der LOD0 werden nun zufällig 5.000 gewählt und als LOD0 Punktmenge zusammengefasst. Anschließend wird die Zuordnung der verbliebenen Punkte auf die vier LOD1 Teilbereiche bestimmt. Sei folgende Verteilung angenommen: auf die Teilbereiche $LOD1_1$ fallen 0 Punkte, $LOD1_2$ 500, $LOD1_3$ 2.000 und $LOD1_4$ 2.500 Punkte. Bei dieser Verteilung werden lediglich die Teilbereiche $LOD1_3$ und $LOD1_4$ mit der entsprechenden Punktanzahl erstellt. Ein zufälliges Ziehen findet hier nicht statt, da es sich um die letzte LOD-Stufe handelt, der sämtliche verbliebene Punkte zugeordnet werden müssen. Die Teilbereich $LOD1_1$ ist leer und wird somit nicht generiert. Die Teilbereich $LOD1_2$ ist nicht leer, jedoch liegt die Anzahl der Punkte unten der genannten Schwelle von 20% der Mindestpunktanzahl, was sich im diskutierten Beispiel auf 1.000 beläuft. Daher werden diese Punkte dem zuvor generierten LOD0 Teilbereich zugeordnet, welche nun 5.500 Punkte umfasst.

Abbildung 7.16 gibt einen Überblick über das gezeigte LOD Stufen Modell. Demnach setzt sich ein hybrides Modell auf einer gewissen LOD Stufe aus jeweils zwei Teilen zusammen. Zum einen aus dem entsprechenden Texturatlas des MipMappings sowie aus den verbliebenen Punkten der jeweiligen LOD Stufe, wobei alle Punkte der kleineren LOD Stufen hinzugenommen werden.

Abbildung 7.17 zeigt das gerenderte original Punktwolken Modell (links) im Vergleich zum erstellten hybriden Modell in LOD3. Der visuelle Eindruck ist nahezu identisch, wobei der Speicheraufwand auf ein Drittel reduziert wurde, da 78% der Punkte 51 texturierten Flächen zugeordnet und entsprechend ersetzt wurden. Der reduzierte Speicheraufwand resultiert wie beschrieben zum einen daher, dass die Geometrie Information (xyz-Kooridnaten) der ersetzten Punkte entfällt, darüber hinaus entfallen

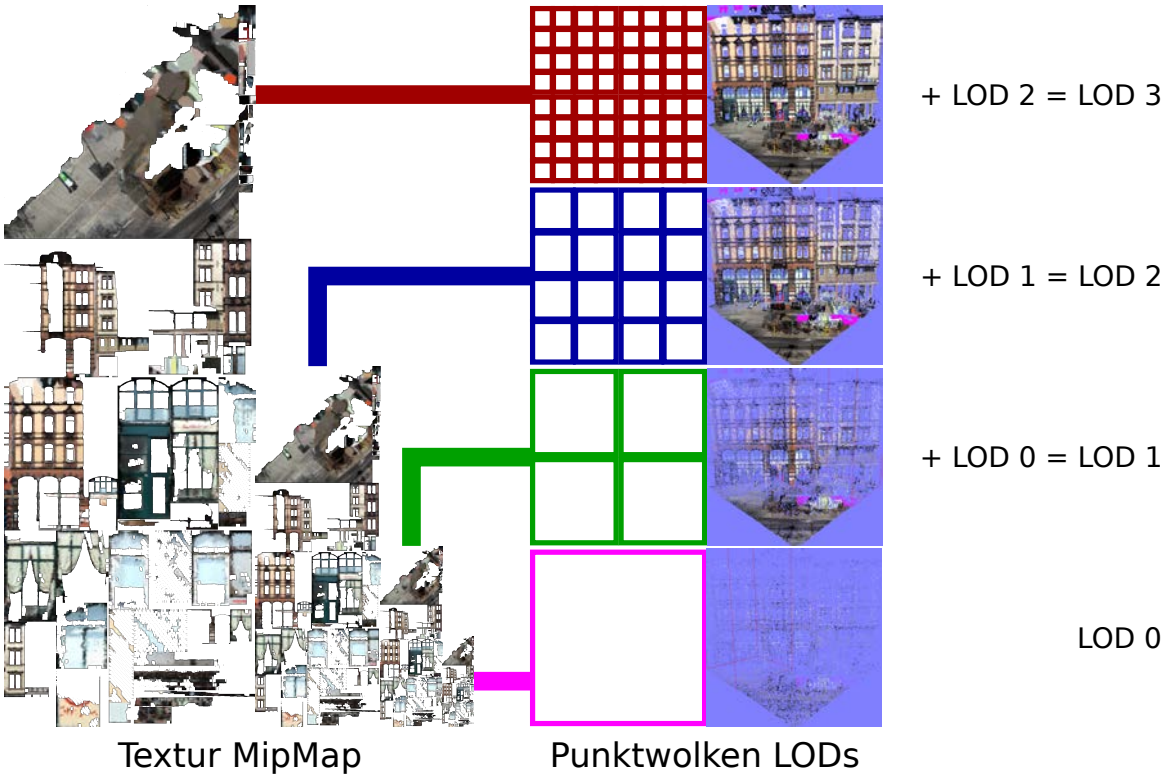


Abbildung 7.16: LOD Stufen

die Punktnormalen der ersetzten Punkte, da diese nun durch die Ebenennormale repräsentiert wird und abschließend die Bilddaten via Deflate komprimiert vorliegen. Letzteres verringert den Umfang an persistent gespeicherten, sowie zu übertragenen Daten, sollten diese nicht lokal vorliegen. Werden die Daten zur Visualisierung geladen, liegen sie entsprechend dekomprimiert im Arbeits- bzw. Grafikspeicher vor.

Abbildung 7.17: Vergleich von Punktwolkenmodell (links) und generiertem hybrid Modell (rechts).

7.2 2D Modelle

Allgemeine Renderprogramme für Punktwolken können mehrere Millionen Punkte mit einer Bildwiederholrate von mehr als 15 Bildern pro Sekunde darstellen. Sollen jedoch größere Punktwolken dargestellt werden oder soll schwächere Hardware, wie bspw. mobile Geräte, genutzt werden, so sinkt die Bildwiederholrate stark oder eine Darstellung ist überhaupt nicht mehr möglich. Eine Form diesem Problem zu begegnen ist, den Datenumfang drastisch zu reduzieren. Dies kann beispielsweise über eine Reduktion der Dimensionalität geschehen, was einer Generalisierung der Daten entspräche. Klassischerweise wird hierzu einfach die Höheninformation verworfen und man erhält eine „kartenähnliche“Abbildung der Mobile Mapping Daten. Über eine Einfärbung der jeweiligen Messpunkte kann die entstandene Karte dann mit entsprechender Semantik angereichert werden, bspw. eine fotorealistische Einfärbung oder eine Einfärbung nach der Höhe oder Intensität, wie in Abbildung 7.18 zu sehen.

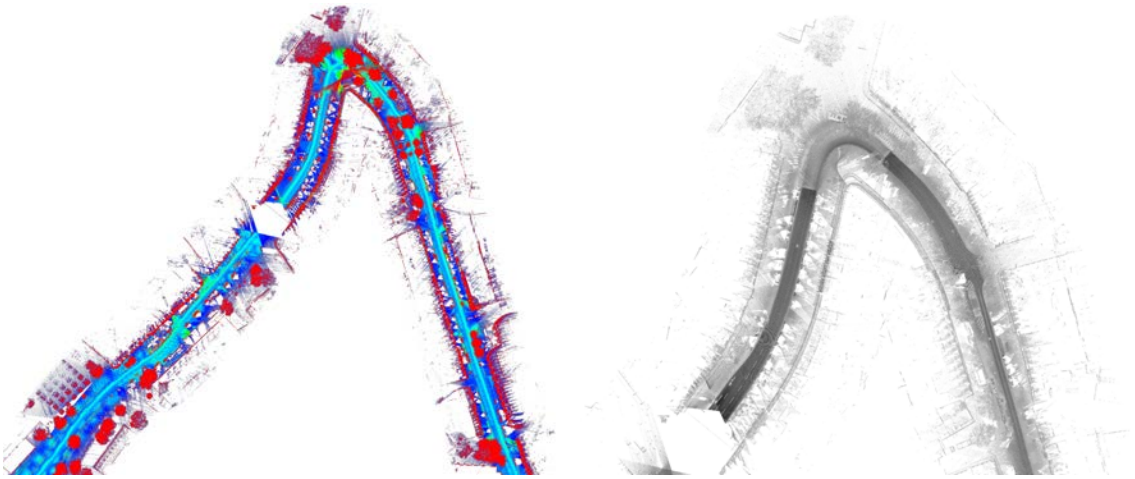


Abbildung 7.18: 2D Kartenmodelle von Mobile Mapping Daten, erzeugt durch Reduktion um eine Dimension, Einfärbung nach Höhe (links) und nach Intensität (rechts)

Mit der Reduktion auf solche 2D Modelle sinkt der Speicherbedarf drastisch und passende Darstellungstechniken ermöglichen darüber hinaus eine interaktive Visualisierung, selbst webbasierte Darstellungen sind leicht zu realisieren. Da jedoch die Höheninformation verworfen wurde, fehlen solchen Modellen sämtliche Fassadenstrukturen.

Um 2D Modelle mit Fassadenstrukturen zu erstellen, dürfen demnach die Höhenwerte nicht verworfen werden. Die Reduktion der Dimensionalität muss folglich in Richtung der Normalen der Fassadenebene erfolgen. Darüber hinaus können Objekte des Straßenraums, welche sich vor der Fassade befinden, genutzt werden, um einen Tiefeneindruck zu simulieren. Diese Visualisierungstechnik wird als Bewegungsparallaxe (engl. *Parallax Scrolling*) bezeichnet und ist in Abschnitt 2.2.4.3 beschrieben.

Die Erstellung von 2D Fassadenmodellen basiert im Wesentlichen auf dem gleichen Ansatz wie die eingangs erwähnten „kartenähnlichen“Abbildungen, nur, dass hier nicht die Höheninformation verworfen wird, sondern die Tiefeninformation entlang der Ebenennormale der Fassadenebene. Hierfür muss zunächst entschieden werden, welcher Fassadenteil der erfassten Gebäude modelliert werden soll. Grundsätzlich sind natürlich die Fassadenteile, welche parallel zur Trajektorie des Mobile Mapping Fahrzeugs orientiert sind, am dichtesten erfasst und eigenen sich daher am besten zur Erzeugung eines Fassadenmodells.

Die Basis für die Generierung Trajektorie-paralleler Fassadenmodelle bilden sowohl die Trajektorie des Mobile Mapping Fahrzeugs, als auch die erfasste, eingefärbte Punktwolke. Im Folgenden werden mehrere Ansätze zur Erzeugung solcher Modelle entwickelt und hinsichtlich der Effizienz der Erstellung und der visuellen Qualität des resultierenden Modells bewertet. Grundsätzlich lässt sich die Modellgenerierung in zwei Schritte unterteilen. Zunächst müssen im ersten Schritt geeignete

Trajektorienabschnitte bestimmt werden. Für jeden Abschnitt werden dann im zweiten Schritt die relevanten Ebenen identifiziert. Die dafür untersuchten Ansätze sind *manuelle Definition*, *Cluster Analyse* sowie *semantische Identifikation*. Abschließend werden für jeden Trajektorieabschnitt die teiltransparenten Texturen für alle Ebenen durch Projektion der zur jeweiligen Ebene gehörenden Punkte auf die Ebene erstellt.

Abbildung 7.19 veranschaulicht die beschriebenen Schritte. Geht man von einem lokalen Koordinatensystem mit x-Achse in Fahrtrichtung, y-Achse senkrecht zur Fahrtrichtung (Tiefe) und z-Achse als Höhe aus, dann kann Schritt eins als Analyse in x-Richtung und Schritt zwei als Analyse in y-Richtung betrachtet werden. Dabei entspricht die Segmentlänge der Trajektorienabschnitte der Bildbreite. Setzt man ein festes Seitenverhältnis voraus, so ergibt sich die Bildhöhe implizit aus den zugehörigen Punkten. Die Menge der Ebenen bzw. Texturen pro Abschnitt ergibt sich dabei aus der Analyse der y-Richtung.

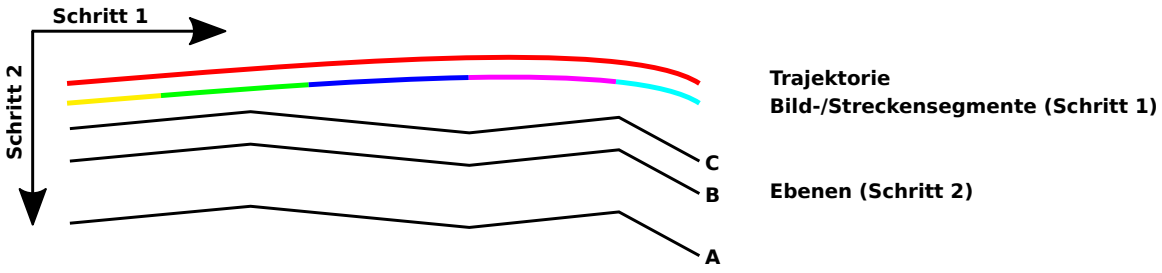


Abbildung 7.19: Visuelle Darstellung der Schritte der Modellgenerierung.

7.2.1 Trackjektorienabschnitte

Das Ziel des ersten Schrittes stellt die Unterteiltung der Trajektorie in einzelne Segmente dar. Zu jedem dieser Segmente werden dann im zweiten Schritt parallele Ebenen bestimmt. Eine *gute* Segmentierung liegt vor, wenn die einzelnen Segmente zum einen nur gering von der originalen Trajektorie abweichen und zum anderen weder zu kurz noch zu lang sind, da dies den späteren Dimensionen der generierten Bilder entspricht. Eine Trajektorie kann geometrisch gesehen als Kurve bzw. Linie aufgefasst werden. Eine Segmentierung kann demnach mit bekannten Liniengeneralisierungstechniken erfolgen. Im Folgenden werden verschiedenste Verfahren vorgestellt und abschließend anhand ihrer Eignung für die Modellgenerierung sowie ihrer Laufzeiteffizienz bewertet.

7.2.1.1 Äquidistanzvereinfachung

Eine Form der Liniengeneralisierung stellt die Unterteilung in gleich lange (äquidistante) Liniensegmente dar. Im Gegensatz zu Generalisierungstechniken wie Subsampling bzw. Subsampling mit Mindestabstand werden hier exakt gleich lange Liniensegmente generiert.

Abbildung 7.20 zeigt das Resultat der Äquidistanzvereinfachung (rot) anhand eines beispielhaften Trajektorienabschnitts (grün). Aus Modellsicht sind die resultierenden gleichgroßen Abschnitte von Vorteil, da hier die erstellten Texturen ebenfalls die gleiche Größe aufweisen und sich somit leichter Optimierungsansätze finden lassen (bspw. bei der Erstellung des Texturatlas'). Nachteilig ist hier jedoch, dass, wie in Abbildung 7.20 markiert (Blitz), Situationen auftreten können, bei denen die generierten Trajektoriesegmente die vorhandenen Fassadenflächen (gestrichelt) geschnitten werden. Dies führt dazu, dass eine Falschzuordnung der Strassenseite für Teile der Fassade erfolgt, was in einem fehlerhaften Modell resultiert. Selbst wenn das Trajektoriesegment die Fassadenfläche nicht schneidet, so erschwert dies das Finden von Ebenen parallel zur Trajektorie, da bei stark gekrümmten Fassaden nur ein geringer Teil wirklich parallel zum entsprechenden Segment orientiert sind.

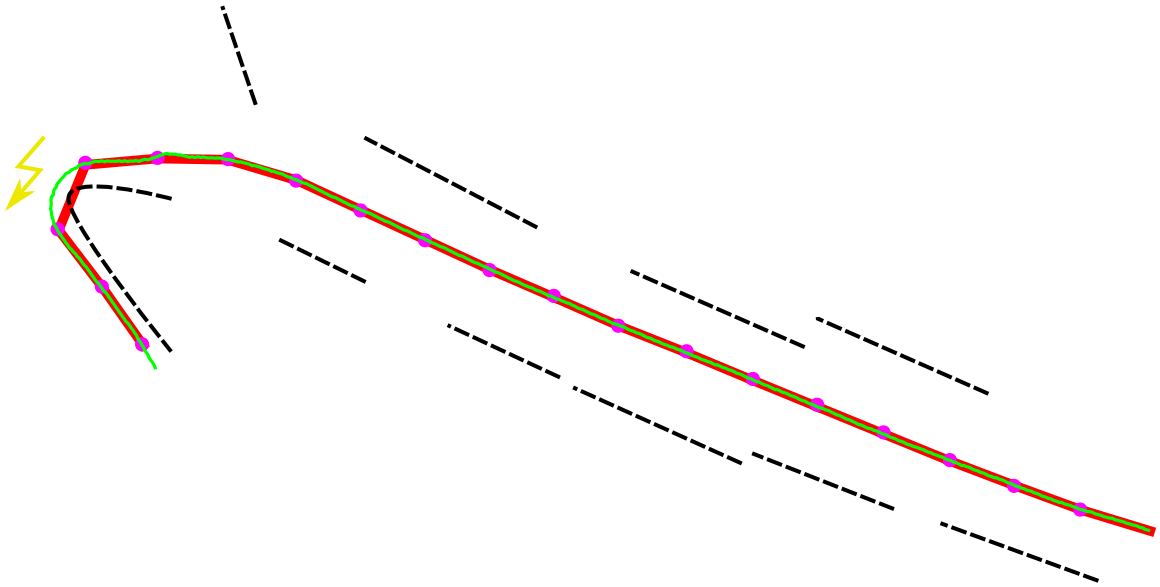


Abbildung 7.20: Ergebnis der Äquidistanzvereinfachung (rot) der erfassten Trajektorie (grün) und angedeuteten Fassadenflächen (gestrichelt) mit markierter Problemstelle (Blitz).

7.2.1.2 Douglas-Peucker

Eine weitere Möglichkeit zur Erstellung der Trajektorieabschnitte stellt die Liniengeneralisierung nach Douglas und Peucker (1973) dar. Ausgehend von Anfangs- und Endpunkt der Trajektorie wird der von diesen Punkten gebildeten Geraden am weitesten entfernte Punkt ausserhalb eines vorgegebenen Grenzwertes hinzugefügt. Der Vorgang wiederholt sich dabei rekursiv für die neuen Abschnitte bis kein Punkt mehr ausserhalb des Grenzwertes liegt.

Im Gegensatz zum zuvor vorgestellten Verfahren der Äquidistanzvereinfachung haben die Trajektoriesegmente hier einen maximalen Abstand vom vorgegebenen Grenzwert zur originalen Trajektorie. Mit der Wahl eines geeigneten Grenzwertes kann somit das Schneiden der Segmente mit den Fassadenflächen verhindert werden.

Analog zu Abbildung 7.20 zeigt Abbildung 7.21 zeigt das Ergebnis der Generalisierung nach Douglas-Peucker. Die Fassaden werden nun nicht mehr von den Trajektoriesegmenten geschnitten. Jedoch treten hier zwei andere Probleme auf. Je nach Wahl des Grenzwertes und Beschaffenheit der Trajektorie werden sehr lange oder aber sehr kurze Trajektoriesegmente gebildet. Lange Segmente resultieren in entsprechend großen Texturbildern, welche unter Umständen nicht von der verwendeten Hardware angezeigt werden können. Im Gegensatz dazu besteht die Gefahr bei sehr kurzen Segmenten, dass eine verlässliche Ermittlung der Ebenen nicht möglich ist, da nur wenige Punkte zum Trajektoriesegement zugeordnet werden.

Wie Abbildung 7.22 zeigt, lassen sich die langen Segmente mittels eines Grenzwertes in entsprechend mehreren kürzere Segemente unterteilen. Vereint man analog dazu die kurzen Segmente, so besteht wiederum die Gefahr, dass die Fassaden von den resultierenden Segmenten wieder geschnitten werden.

7.2.1.3 Opheim

Eine Liniengeneralisierung bei dem sowohl eine minimale, als auch eine maximale Segmentlänge festgelegt werden kann ist das Verfahren nach Opheim (1981) und Opheim (1982). Es stellt im Wesentlichen eine Erweiterung des Verfahrens nach Reumann und Witkam (1974) dar. Ausgehend vom Startpunkt der Trajektorie wird dabei iterativ der am weitesten entfernte Punkt bestimmt, der ausserhalb des unteren Grenzwertes und innerhalb des oberen Grenzwertes, sowie innerhalb eines Puffers um die Linie, welche vom aktuellen Punkt und seinem Nachfolger gebildet wird, liegt.

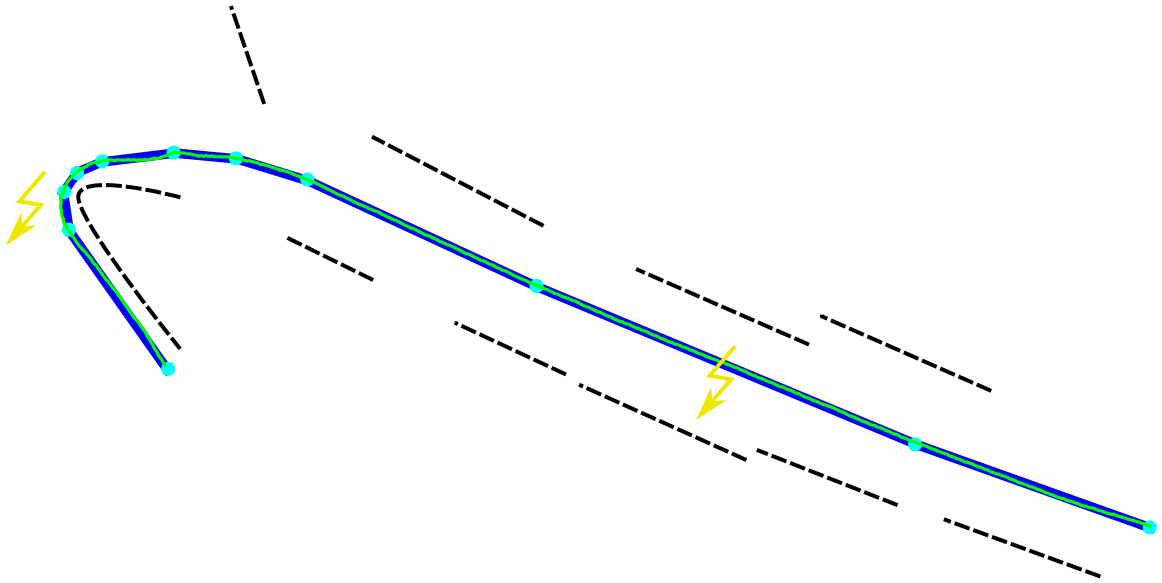


Abbildung 7.21: Ergebnis der Generalisierung der erfassten Trajektorie (grün) nach Douglas-Peucker (blau) und angedeuteten Fassadenflächen (gestrichelt), mit markierten Problemstellen (Blitz).

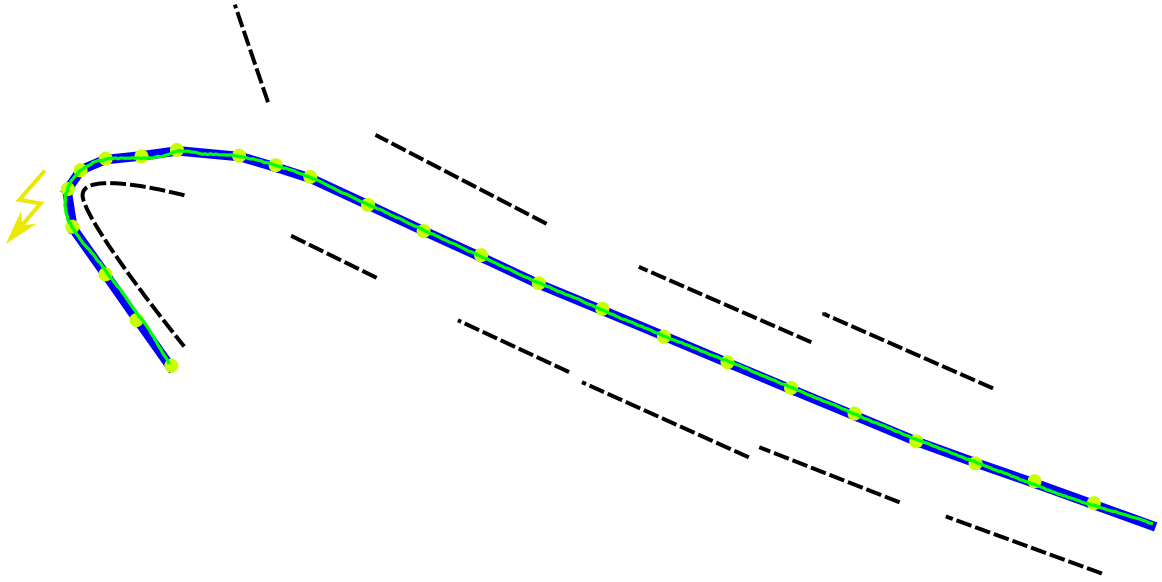


Abbildung 7.22: Ergebnis der Generalisierung der erfassten Trajektorie (grün) nach Douglas-Peucker (blau) mit maximaler Segmentlänge, angedeuteten Fassadenflächen (gestrichelt) und markierter Problemstelle (Blitz).

Abbildung 7.23 zeigt das Ergebnis des Verfahrens. Aufgrund des oberen Grenzwertes entstehen bei geraden Abschnitten Segmente ähnlich der Äquidistanzvereinfachung. Durch die Nutzung des unteren Grenzwertes sind selbst in Kurvenbereichen keine zu kurzen Segmente entstanden. Darüber hinaus ist die Gefahr, dass Segmente Fassaden schneiden, bei geeigneten Grenzwerten ebenfalls gering.

Einen Vergleich bekannter Verfahren zur Liniengeneralisierung gibt Shi und Cheung (2006). Allerdings stehen dort Distanz- und Ähnlichkeitsmetriken im Vordergrund. An dieser Stelle sollen die vorgestellten Verfahren anhand ihrer Laufzeit, sowie ihrer Eignung für das Modellgenerierungsverfahren analysiert werden.

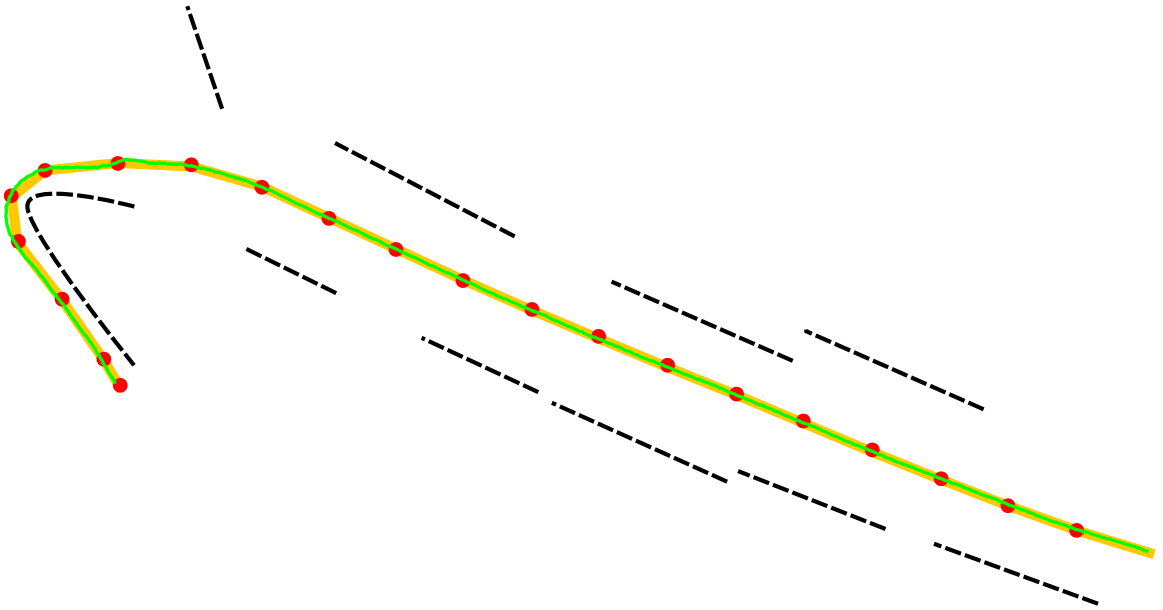


Abbildung 7.23: Ergebnis der Generalisierung der erfassten Trajektorie (grün) nach Opheim (orange) und angedeuteten Fassadenflächen (gestrichelt).

Die potentiellen Probleme wie zu kurze oder lange Segmente, sowie Schnitte mit Fassaden wurden bereits bei der Vorstellung der Verfahren diskutiert. Ein grundsätzliches Ausschlusskriterium sind zu lange Segmente, da dies eine Darstellung des generierten Modells unter Umständen unmöglich macht. Je nach verwendetem Renderingframework werden Bilder mit mehr als 1024 Pixeln Breite nicht unterstützt. Geht man von einer mittleren Auflösung von 2cm pro Pixel aus, so darf beispielsweise die Länge eines Trajektoriensegments $1024pix * 0.02\frac{m}{pix} = 20,48m$ nicht überschreiten. Daher scheidet die Douglas-Peucker in der Referenzvariante als ungeeignet aus. Werden im Nachgang zu lange Segmente geteilt, so ist das Douglas-Peucker Verfahren durchaus geeignet. Zu kurze Segmente und Schnitte mit Fassaden führen unter Umständen zu visuell weniger ansprechenden Modellen. Das einzige der vorgestellten Verfahren welches diese Probleme minimiert ist, wie bereits diskutiert, das Verfahren nach Opheim.

Unter dem Aspekt der effizienten Erstellung der Modelle spielt die Laufzeit der Verfahren eine ebenfalls elementare Rolle. Das Diagramm aus Abbildung 7.24 gibt Aufschluss über die Laufzeiten der vorgestellten Verfahren. Die Implementierung des Opheim Verfahrens erzielt dabei die beste Laufzeit. Die geringe Laufzeit liegt im Wesentlichen in der iterativen Natur des Verfahrens begründet, wobei nur genau eine Distanz zu jedem Punkt der Trajektorie bestimmt wird. Die Äquidistanzvereinfachung ist ebenfalls iterativ, dabei müssen jedoch Punkte auf sämtlichen Segmenten interpoliert werden, um Segmente gleicher Länge zu erhalten, dies resultiert in einer etwa um die Hälfte längeren Laufzeit. Im Falle des Douglas-Peucker Ansatzes handelt es sich um ein rekursives Verfahren. Dabei werden für jeden Trajektoriepunkt mehrfach Punkt-Linien Abstände bestimmt, was die Laufzeit auf fast das dreifache der Opheim Laufzeit anwachsen lässt.

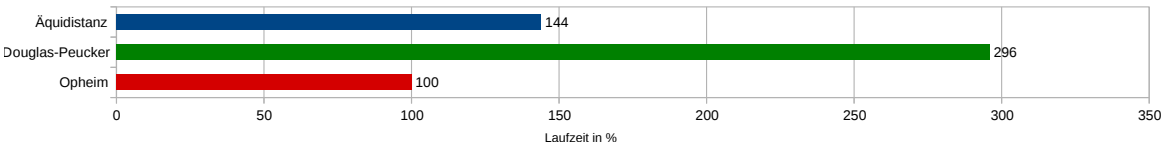


Abbildung 7.24: Diagramm der Laufzeiten der vorgestellten Verfahren zur Liniengeneralisierung.

Aufgrund geeigneter Segmentgrößen und der geringen Laufzeit wird die Liniengeneralisierung nach Opheim für das weitere Vorgehen gewählt.

7.2.2 Ermittlung relevanter Ebenen

Die Entscheidung über die Anzahl und Lage der zu generierenden Ebenentexturen ist maßgebend für die visuelle Qualität des erzeugten Modells. Werden die Punkte auf nur wenige Ebenen projiziert, bspw. ein oder zwei, führt dies zu einem Verlust der gewünschten Tiefenillusion und somit des 3D Eindrucks, was die Visualisierung weniger immersiv wirken lässt. Wird eine große Anzahl an Ebenen gewählt bzw. erzeugt, hat dies zur Folge, dass verhältnismäßig wenig Punkte auf die einzelnen Ebenen projiziert werden. Dies kann zu „löchrigen“ Texturen führen, welche die Qualität der Visualisierung mindern. Aus diesen Gründen ist es essentiell, die *relevanten* Ebenen für eine gegebene Szene zu ermitteln. Darüber hinaus ist unter visuellen Aspekten wichtig, dass die Fassadenebene die Hintergrundebene repräsentiert, da dies im Wesentlichen der üblichen optischen Wahrnehmung einer urbanen Umgebung (Fahrbahn > Straßenmöblierungen > Hausfassade) entspricht, denn man kann grundsätzlich nicht hinter eine Fassade blicken.

Eine Ebene definiert sich im folgenden über vier Parameter: zum einen das zugehörige Trajektoriesegment S_{Track} und zum anderen drei Entfernungen, bezogen auf S_{Track}. Dabei werden alle Punkte mit einer Distanz größer als einer *Mindestentfernung* d_{min} und kleiner als eine *Maximalentfernung* d_{max} auf eine Ebene mit der Entfernung d_{ref} projiziert.

7.2.2.1 Manuelle Definition

Der einfachste Weg relevante Ebenen zu identifizieren ist die manuelle Definition durch den Nutzer. Dabei obliegt dem Nutzer die Entscheidung über Anzahl der zu erstellenden Ebenen, sowie deren jeweiligem Abstand von der Trajektorie. Dafür gibt der Nutzer die gewünschte Anzahl (bspw. 4m, 8m und 12m) an Distanzen d_{ref} vor. Daraus werden nach obiger Definition die Ebenen gebildet, wobei als Mindestentfernung d_{min} die Maximalentfernung d_{max} der vorherigen Ebene genutzt wird. Die Mindestentfernung der ersten/vordersten Ebene wird dabei auf Null gesetzt. Als d_{max} wird jeweils die Mitte zwischen aufeinander folgenden d_{ref} Distanzen gewählt. Die Maximalentfernung der letzen/hintersten Ebene wird auf einen vorgegebenen Maximalwert gesetzt.

Abbildungen 7.25 und 7.26 zeigen Teile des generierten 2D Ebenenmodells bei manueller Definition dreier Ebenen in 4 Metern, 8 Metern und 12 Metern Entfernung zur Trajektorie.

Abbildung 7.25: 2D Ebenen Modell bei manueller Definition der Ebenenentfernungen.

Während bei Abbildung 7.25 alle Fassadenebenen in der hintersten Ebene liegen und damit ein stimmiger Gesamteindruck entsteht, wechselt Die Fassadenebene bei Abbildung 7.26 aus der Hintergrundebene in die mittlere Ebene. Vom visuellen Standpunkt her ist es jedoch wünschenswert die Fassadenebene immer als Hintergrundebene zu besetzen. Diesem Sachverhalt kann mit einer fixen Definition der Ebenen keine, bzw. nur schwer, Rechnung getragen werden.

Abbildung 7.26: 2D Ebenen Modell bei manueller Definition der Ebenenentfernungen mit Wechsel der Fassadenebene vom Hintergrund in die mittlere Ebene.

7.2.2.2 Histogrammbasierte Clusteranalyse

Wie im vorangegangen Abschnitt gezeigt, stellt die manuelle Definition der Anzahl und Lage der Ebene zwar eine einfache Möglichkeit dar, jedoch ist das Ergebnis nicht optimal. Im Folgenden wird ein Ansatz diskutiert, welcher auf einer Clusteranalyse beruht und damit sowohl die Anzahl der Ebenen, als auch deren genaue Lage automatisch ermittelt.

Dazu wird zunächst ein Histogramm der (2D) Distanzen der Punkte zum jeweiligen Trajektoriesegment erstellt. Abbildung 7.27 zeigt ein solches Distanzhistogramm. Für eine bessere Trennung der Objektcluster werden vor der Erstellung des Histogramms nach dem Verfahren aus Abschnitt 4.4.1 die Bodenpunkte aus der Punktwolke entfernt.

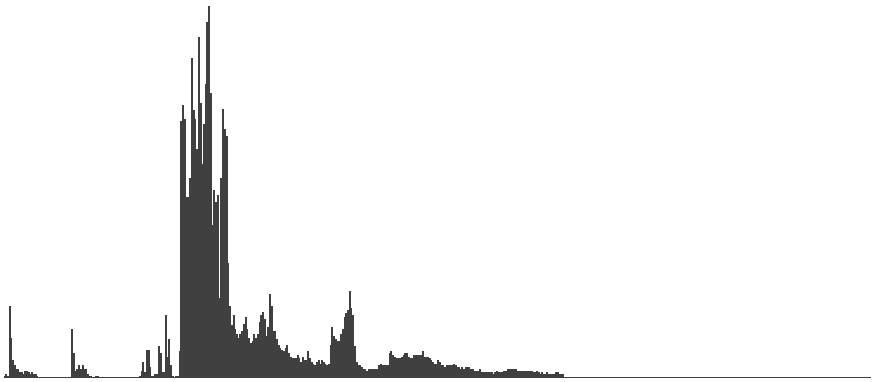


Abbildung 7.27: Histogramm der Distanzen der Punkte bezüglich der Trajektorie.

Grundsätzlich ist davon auszugehen, dass die Fassadenebenen die höchste Anzahl an Punkten aufweist. Das Histogramm aus Abbildung 7.27 stützt diese Annahme. Das Maximum des Histogramm entspricht dabei exakt der Entfernung der Fassadenebene zur Trajektorie.

Auf Basis des erstellten Histogramms der Distanzen werden nun iterativ alle relevanten Ebenen mit dem dazugehörigen Distanzparameter $d_{min,ref,max}$ für das jeweilige Trajektoriesegment ermittelt. Die beschriebene Vorgehensweise ist dabei in Abbildung 7.28 veranschaulicht. Als Entfernung der Projektionsebene d_{ref} wird das aktuelle Maximum (Peak) des Histogramms herangezogen (1). Anschließend müssen die Minimal- und Maximalentfernungen $d_{min,max}$ ermittelt werden. Diese werden zunächst über eine zuvor festgelegte maximale Ebenenbreite initialisiert (2). Für die tatsächlichen Entfernungen werden im Schritt (3) die äußersten lokalen Minima ermittelt. Unter der Prämisse, dass die Fassadenebene die meisten Punkte einschließt, ist die erste auf diesem Wege ermittelte Ebene die Fassadenebene. Da keine Ebene hinter dieser existieren darf, wird d_{max} für diese Ebene auf den

letzten Wert des Histogramms ausgedehnt (4). Abschließend werden alle Werte zwischen d_{min} und d_{max} aus dem Histogramm entfernt (5). Diese Schritte werden wiederholt bis das Histogramm leer ist, wobei Schritt (4) entsprechend entfällt.

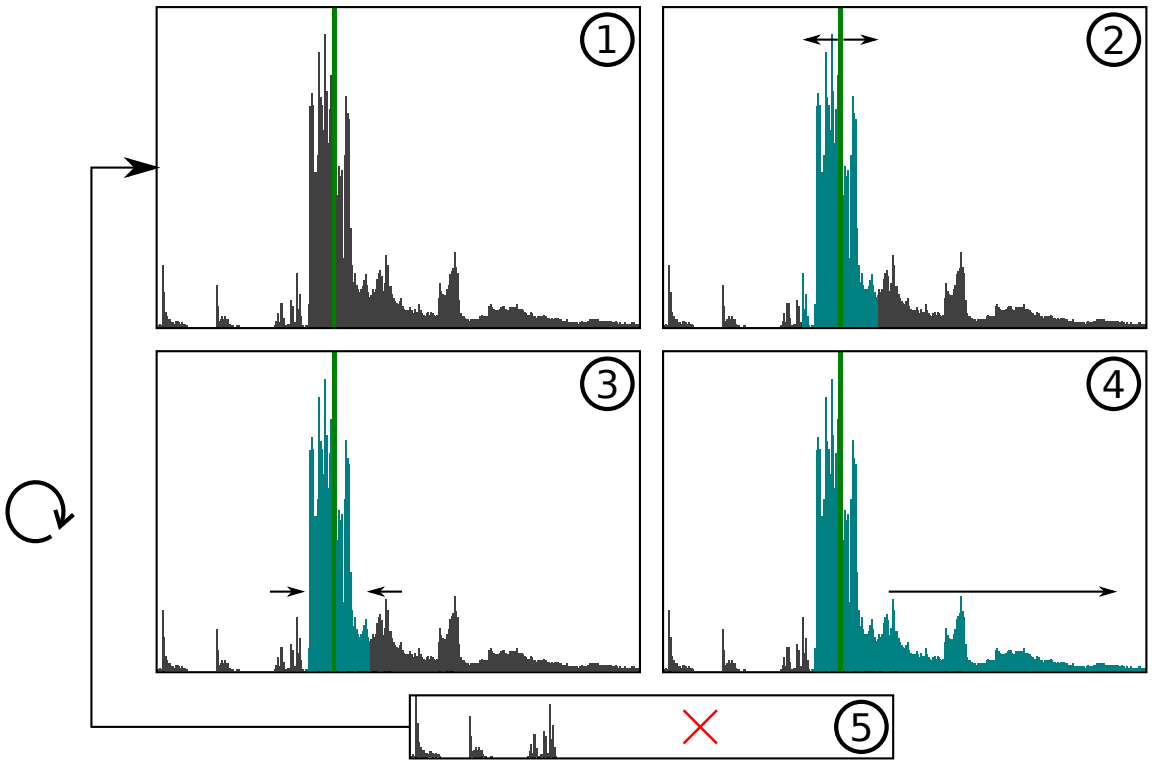


Abbildung 7.28: Ablauf der Ebenenermittlung auf Basis des Distanzhistogramms. (1) Maximum finden, (2) Ebenenbreite mit maximaler Ausdehnung initialisieren, (3) auf äußerste lokale Minima reduzieren, (4 nur bei Fassadenebene) Ausdehnung der maximal Entfernung auf letzten Histogrammwert, (5) Entfernung des gefundenen Bereichs aus dem Histogramm. Abschließend Wiederholung bis Histogramm leer.

Um die Erstellung kleiner bzw. schmaler Ebenen zu verhindern wird zusätzlich zur maximalen Ebenenbreite noch eine minimale Ebenebreite definiert. Die lokalen Minima müssen sich folglich außerhalb dieser Entfernung befinden. Darüber hinaus werden die Ebenengrenzen, abweichend von der definierten maximalen Breite angepasst, wenn dadurch in folgenden Iterationen zu kleine Ebenen entstehen würden.

Abbildung 7.29 zeigt ein konkretes Beispiel für das beschriebene Verfahren, mit den ermittelten Ebenen und den daraus erzeugten Ebenenprojektionen. Das Resultat zeigt drei Ebenen, wobei zunächst die Fassadenebene (1), dann die vorderste Ebene (2) und abschließend die mittlere Ebene (3) erstellt wurden. Wie zu erkennen ist, werden nicht notwendigerweise objektbezogene Ebenen erzeugt. Das Maximum der vordersten Ebene wurde durch das Fahrzeug erzeugt, wohingegen das Maximum der mittleren Ebene durch die Litfaßsäule, sowie durch die Baumstämme erzeugt wurde. Die Baumkrone erstreckt sich hingegen über beide Ebenen.

Die vorgestellte histogrammbasierte Clusteranalyse ermittelt, wie gezeigt, automatisch die Anzahl, Lage, sowie Ausdehnung der relevanten Ebenen einer Szene. Die maximale und minimale Ebenenbreite stellen dabei die einzigen festzulegenden Parameter dar. Im Gegensatz zur zuvor diskutierten manuellen Definition der Ebenen werden hier immer Fassaden als Hintergrundebenen erzeugt. Dies beruht auf der Annahme, dass die Fassadenpunkte das globale Maximum des Distanzhistogramms repräsentieren. Obgleich diese Annahme in den meisten Fällen zutrifft, so gilt sie jedoch nicht immer. Beispielsweise dann, wenn die Fassade nicht parallel zur Trajektorie verläuft. In diesem Fall wird kein klarer Peak, sondern eher ein Plateau erzeugt. Da die Punktdichte mit der Entfernung abnimmt, kann davon ausgegangen werden, dass das globale Maximum des Distanzhistogramms sich

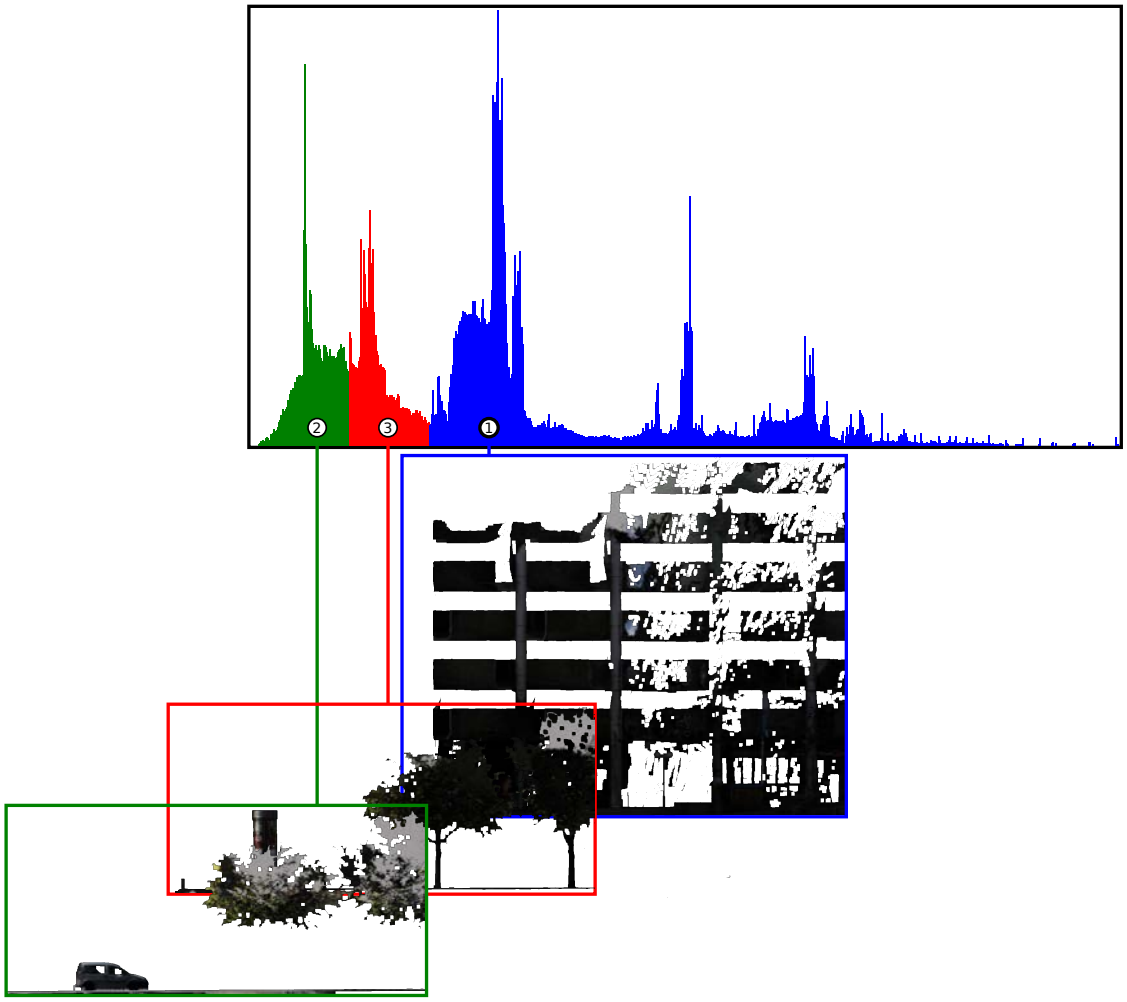


Abbildung 7.29: Beispiel für ermittelte Ebenen via Histogramm Analyse (Reihenfolge: 1,2,3 basierend auf globalen Maxima) und den daraus generierten Ebenentexturen.

vor oder auf der Fassade befindet. In diesem Fall werden dennoch alle Fassadenpunkte auf die erste erstelle Ebene (Hintergrundebene) projiziert, was der ursprünglichen Intention entspricht.

7.2.2.3 Semantische Identifikation

Neben der vorgestellten Clusteranalyse kann zur Erstellung der Bildebenen auch ein semantischer Ansatz herangezogen werden. Dazu muss die Szene zunächst in separate Objekte segmentiert werden. Dies kann bspw. durch das in Abschnitt 4.4 beschriebene Verfahren erfolgen. Mittels der durchgeführten Segmentierung wurden die Punkte implizit mit einer Objektsemantik angereichert, wonach alle Punkte eines Segments zu ein und demselben Objekt gehören. Darauf basierend lassen sich alle Objekte auf einzelne Ebenen projizieren. Je nach Anzahl der Objekte kann hier die Ebenenanzahl entsprechend hoch ausfallen. Um dies zu verhindern, werden die Objekte zunächst in Entfernungsgruppen zusammen gefasst. Dies kann mittels eines beliebigen Clustering Verfahrens, wie K-Means (MacQueen (1967) bzw. Alsabti u. a. (1997)) oder Jenks-Natural-Breaks (Jenks und Caspall (1971)), erfolgen.

Die Qualität des erzeugten Modells hängt dabei stark von der Güte der durchgeführten Segmentierung ab. Dabei übertragen sich die üblichen Segmentierungsprobleme wie Über- bzw. Untersegmentierungen direkt auf die generierten Objekttexturen, wobei sich Über- als auch Untersegmentierungen

in Richtung der Trajektorie weniger stark auf das Ergebnis auswirken, da Objekte in ähnlicher Distanz zur Trajektorie durch das Clustering sehr wahrscheinlich ohnehin derselben Ebene zugeordnet worden wären.

7.2.2.4 Vergleich

Vergleicht man die drei vorgestellten Ansätze unter Effizienzgesichtspunkten, so ist die manuelle Definition der Ebenen vorzuziehen, da hier keinerlei Aufwand in das automatisierte Finden relevanter Ebenen investiert werden muss. Dies relativiert sich jedoch, sobald keine globalen Distanzen definiert werden können. Werden vom Nutzer manuell Distanzen für individuelle Trajektoriesegmente definiert, so erhöht sich der zeitliche Aufwand (des Nutzers) enorm. Beschränkt man sich bei der Laufzeitbetrachtung auf die reine Verarbeitungszeit des Algorithmus, so bleibt die manuelle Definition der Ebenen die laufzeiteffizienteste Variante. Unter selbigem Aspekt ist die Clusteranalyse der Distanzhistogramme der Semantischen Identifikation vorzuziehen, da die Erstellung und Verarbeitung der Histogramme wesentlich performanter erfolgt als die Segmentierung und das anschließende Clustering der Objektsegmente. Unter visuellen Aspekten erzeugt die manuelle Definition der Ebenen jedoch wie beschrieben mangelhafte Modelle. Die Ergebnisse der anderen beiden Ansätze sind dahingegen weitestgehend vergleichbar. Daher werden für die im folgenden Kapitel vorgestellte Visualisierung Modelle mittels der Histogrammmethode generiert.

7.2.3 Ergebnis

Das Ziel des beschriebenen Verfahrens war die Erzeugung eines stark vereinfachten Modells, welches geringe Speicheranforderungen aufweist und effizient visualisiert werden kann. Die Speicherreduktion hängt stark von der gewählten Auflösung der erzeugten Bilder ab. Als guter Kompromiss zwischen Speichereffizienz und Detailgrad wurde ein Auflösungsverhältnis von $0,02\frac{Meter}{Pixel}$ gewählt. Damit entspricht ein Bildpixel 2 Zentimetern in der Realität. Der Messabschnitt zur Trajektorie aus den Abbildungen 7.20 bis 7.23 umfasst ca. 42 Mio. Scanpunkte, mit einem Datenumfang von etwa 1,2 GB. Das erzeugte Fassadenmodell umfasst für den genannten Abschnitt 305 Bilder für die linke und 401 Bilder für die rechte Straßenseite. Der Datenumfang der Abbildungen beträgt zusammen etwa 130 MB. Dies entspricht ein Reduktion auf unter 11% des ursprünglichen Datenvolumens. Eine stärkere Kompression wird durch Anpassung des Auflösungsverhältnisses erreicht. Erhöht man das Verhältnis von 2 auf bspw. 4 Zentimenter pro Pixel, so beträgt der Datenumfang der Abbildungen lediglich 42 MB, bedingt durch die Halbierung der Breite, sowie Höhe des Bildes.

Die Laufzeit der Modellgenerierung hängt von vielen Faktoren, wie die Performanz des verarbeitenden Systems, die vorliegende Punktdichte, das gewählte Auflösungsverhältnis und die Geschwindigkeit des Speichermediums ab. Die folgenden Laufzeitangaben haben daher einen eher qualitativen Charakter. Die Erstellung aller Bilder eines Trajektoriesegments betrug auf einem aktuellen Computersystem und unter Verwendung der Cachingstrategie aus Abschnitt 3.2.4 im Durchschnitt 10 Sekunden. Bei Segmentlängen von etwa 20 Metern kommt man auf eine Verarbeitungszeit von etwa einer halben Sekunde pro gefahrenem Meter, oder 8-9 Minuten pro Kilometer. Eine Erstellung des Modells in Echtzeit, bspw. direkt während der Erfassungsfahrt, ist somit nur bedingt möglich. Da ein Großteil der Verarbeitungszeit für das Laden der Scandaten in den Hauptspeicher benötigt wird, ist das Einsparpotential hier am größten. Würde beispielsweise lediglich die Hälfte der Punkte, z.B. jeder zweite, verarbeitet, so ließe sich der Prozess der Modellgenerierung signifikant beschleunigen und unter Umständen echtzeitfähig machen.

Die Analyse der Rendereffizienz der generierten Fassadenmodelle ist in Abschnitt 8.2.3 beschrieben.

8 Visualisierung von Mobile Mapping Daten

Die oberste Schicht des vorgestellten Frameworks bilden die Visualisierungsmodule. Sie umfassen Komponenten und Techniken zur Darstellung der generierten Modelle und bilden somit das Ende der Verarbeitungskette. Abbildung 8.1 ermöglicht die Einordnung in das Gesamtframework.

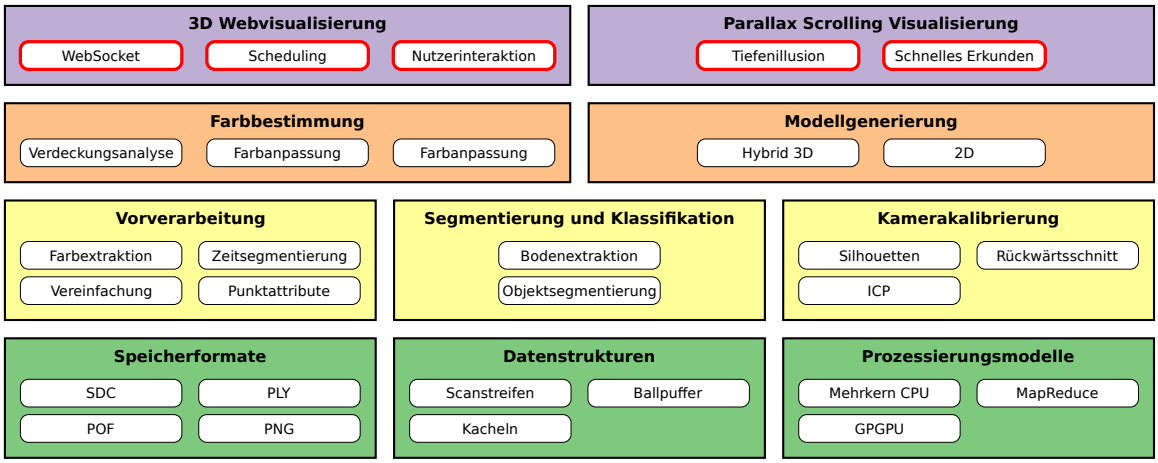


Abbildung 8.1: Einordnung in Framework.

Auf Basis der in Kapitel 7 beschriebenen Modellvarianten wurden zwei zugehörige Visualisierungskonzepte entwickelt, welche mittels entsprechender Prototypen umgesetzt wurden. Abschnitt 8.1 beschreibt eine Web-basierte Visualisierungsvariante zur Darstellung der hybriden 3D Modelle, wohingegen Abschnitt 8.2 sich auf die Darstellung der 2D Ebenenmodelle fokussiert. Dabei werden ausgehend von zuvor gewählten Anwendungsfällen die jeweiligen Entwurfsentscheidungen dargestellt und die Vor- und Nachteile der jeweiligen Zielplattformen, sowie deren Auswirkungen auf die Implementierung beleuchtet.

8.1 3D Visualisierung

Das Ziel ist die Darstellung der hybriden 3D Modelle, welche mittels der in Abschnitt 7.1 beschriebenen Verfahren aus den Mobile Mapping Daten generiert wurden. Hierfür muss zunächst eine Zielplattform definiert werden. Dabei kann, neben weiteren Varianten, zwischen unterstützten Geräteklassen, Betriebssystemen und Implementierungssprache bzw. -framework unterscheiden werden. Um Argumente für oder gegen eine Zielplattform formulieren zu können, sollten zunächst Anwendergruppe und entsprechende Anwendungsfälle definiert werden.

Für die vorliegende Arbeit wurde ein Erkundungsszenario als Anwendungsfall gewählt. Dabei soll dem Nutzer die Möglichkeit geboten werden, die Umgebung, ähnlich Google Streetview (Vincent (2007)) interaktiv zu erkunden. Die angestrebte Anwendergruppe soll dabei der der existierenden Systeme entsprechen, also nicht fachkundigen Nutzern das Erkunden der Umgebung ermöglichen.

Aus dem definierten Anwendungsfall und der angestrebten Nutzergruppe lassen sich nun Anforderungen an die Zielplattform formulieren. Zum einen sollte der Zugang, insbesondere für nicht fachkundige Nutzer so leicht wie möglich gestaltet sein. Zum anderen muss die verwendete Zielplattform eine möglichst große Verbreitung besitzen um ein entsprechend großes Publikum anzusprechen. Eine für die genannten Anforderungen ideale Zielplattform wäre daher geräte- und betriebsystemübergreifend einsetzbar. Ohne an dieser Stelle weiter auf einzelne Geräteklassen oder Betriebsystemen eingehen

zu wollen, erfüllt lediglich eine Plattform die genannten Anforderungen. Diese wäre ein Webbrowser in Kombination mit einer HTML5-basierten Implementierung, so wie es auch beim Vorbild (Google Streetview) umgesetzt wurde. Webbrowser, welche den HTML5 Standard unterstützen, sind für nahezu alle Geräteklassen und Betriebssysteme verfügbar. Darüber hinaus bieten sie leichten Zugang, da weder Programme oder Daten händisch heruntergeladen oder gar installiert werden müssen. Geht man von der Zielplattform Webbrowser/HTML5 aus gilt es noch ein Implementierungsframework zu finden. Hier wurde auf das *Google Web Toolkit* (kurz: *GWT*) zurückgegriffen. GWT ermöglicht die Entwicklung einer Client/Server-basierten Web-Applikation mittels einer einheitlichen Programmiersprache, welche in diesem Fall Java ist. Zur Veröffentlichung der Applikation (engl. deploying) werden dabei alle für den Client-Teil relevanten Klassen vom GWT Compiler von Java nach JavaScript transformiert. Die Serverseitige Implementierung bleibt in der Java Domäne und wird als JavaEE Servlet Container veröffentlicht. Abbildung 8.2 fasst die definierten Anwendergruppen, Anwendungsfälle, sowie die daraus entstandenen Design Entscheidungen und das genutzte Framework zur Umsetzung zusammen.

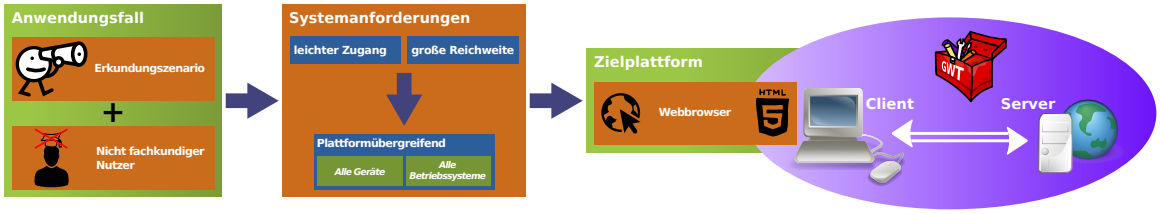


Abbildung 8.2: Übersicht über den Anwendungsfall, die Anwendergruppe und den daraus abgeleiteten Systemanforderungen, sowie das zur Umsetzung genutzte Framework GWT.

8.1.1 Visualisierung via Web-App

Wie eingangs beschrieben, besteht das Ziel darin, eine Web-basierte Anwendung (kurz: Web-App) zur Visualisierung der hybriden 3D Modelle bereitzustellen. Zur Entwicklung dieser Web-App wird im Rahmen dieser Arbeit das GWT Framework genutzt. GWT bringt bereits eine Schnittstelle zur Kommunikation zwischen Clientkomponente, welche als JavaScript Anwendung im Browser des Nutzer läuft und der Serverkomponente, welche als JEE Servlet Container auf einem Web-Server läuft. Darüber hinaus stellt GWT diverse Klassen zur Erstellung der Graphischen Weboberfläche bereit.

Clientseitig sollen die 3D Modelle bestehend aus Punktwolken und Texturen dargestellt werden. Dafür wird die *Web Graphics Library* (kurz: WebGL, Jackson und Gilbert (2015)) genutzt. Sie basiert auf der OpenGL|ES Spezifikation und bietet eine hardwarebeschleunigte 3D Grafik Programmierschnittstelle für HTML5 / JavaScript Umgebungen. Bei dem zu rendernden Modell handelt es sich um das in Abschnitt 7.1.5 beschriebene LOD Modell. Um das Modell darstellen zu können, muss dem Client zunächst die globale Struktur des Modells bekannt gemacht werden. Sie beinhaltet die Verteilung der Punktwolkenkacheln, die dazugehörigen Texturen, sowie die jeweiligen LOD Pyramidenstrukturen der Kacheln. Anschließend muss auf Basis des aktuellen Standortes (der virtuellen Kamera), sowie der aktuellen Blickrichtung der darzustellende Detailgrad bestimmt und die zugehörigen Daten vom Server an den Client übertragen werden. Nach erfolgreicher Übertragung der Daten, werden diese in den Grafikspeicher geladen und können abschließend gerendert werden. Zur Visualisierung der Daten müssen, OpenGL|ES-typisch, für jeden Datentyp (Punkte, texturierte Flächen) ein Shader-Paar aus Vertex- und Fragment-Shader bereitgestellt werden. In den Shadern werden grundlegende Transformationen und Beleuchtungsmodelle auf die jeweilgen Daten angewendet und schließlich das darzustellende Einzelbild einer Szene generiert. Eine Änderung der Position oder der Blickrichtung der Kamera löst ein erneutes rendern der Szene mittels der genannten Shader aus.

Einen Eindruck über die erstelle Web-Applikation vermittelt Abbildung 8.3. Die Nutzeroberfläche besteht dabei aus einer Übersichtskarte (oben rechts), welche die Trajektorie des Erfassungsfahr-

zeugs sowie die aktuelle Position anzeigt, einem Optionsbereich (unten rechts) welcher grundlegende Einstellungen zur Navigation und Statistiken bietet, sowie der gerenderten Szene (links).

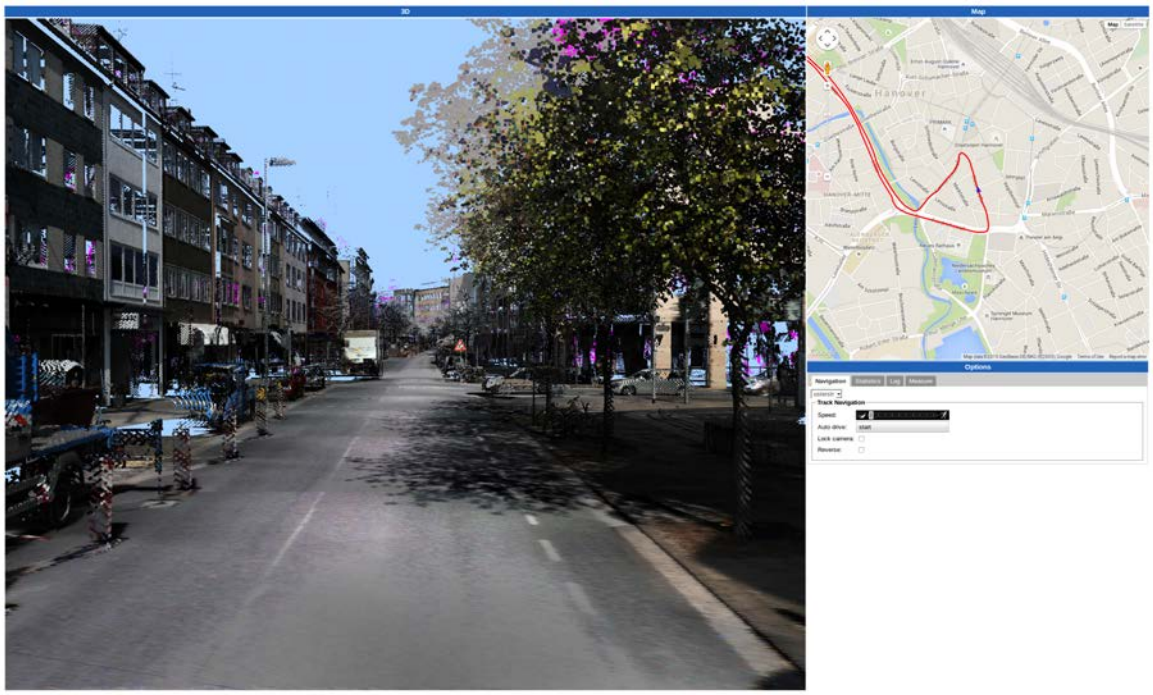


Abbildung 8.3: Nutzeroberfläche der Web-Applikation, bestehend aus Übersichtskarte (oben rechts), Optionsbereich (unten rechts) sowie Renderbereich (links).

8.1.1.1 Blickwinkelabhängige Bestimmung der Detailstufe

Das in Abschnitt 7.1.5 entwickelte blickwinkelabhängige LOD-Konzept umfasst die drei Komponenten *LOD-Datenstruktur*, *Vereinfachungsverfahren* und *die Bestimmung der genutzten Detailstufe zur Laufzeit*. Während die LOD-Datenstruktur und das genutzte Vereinfachungsverfahren Teil der Modellerstellung sind, setzt die Web-Applikation, die Bestimmung der genutzten Detailstufe, als dritte Komponente, um.

Hierfür wird die LOD Pyramidenstruktur in einer Tiefensuche traversiert. Dabei wird auf Basis der Kameraposition und -orientierung entschieden, ob ein Bereich gerendert werden muss und wenn ja in welcher LOD Stufe. Zunächst wird für jeden LOD Teilbereich überprüft, ob er sich im Sichtkegel (View-Frustrum) der Kamera befindet. Ist dies nicht der Fall, findet keine tiefere Traversierung des Baumes statt und es wird mit der nächsten Kachel fortgefahren. Befindet sich eine Kachel im Sichtkegel, so wird die darzustellende LOD Stufe (Keine, LOD0, LOD1, LOD2 oder LOD3) anhand der Entfernung zur Kamera bestimmt. Ist eine zu rendernde LOD Stufe nicht lokal vorhanden, müssen die entsprechenden Daten vom Server abgerufen werden (vgl. Abschnitt 8.1.2). Dabei trägt eine feingranulare Priorisierung der zu ladenden Daten, beschrieben in Abschnitt 8.1.3, zu einer stets umfassenden und performanten Darstellung der Szene bei.

Der visuelle Eindruck ergibt sich im Wesentlichen aus dem Verhältnis des dargestellten Detailgrads (LOD Stufe bzw. Anzahl Punkte / Texturauflösung) zur Kameraentfernung. Dieses Verhältnis ist bei einer hybriden Darstellung von texturierten Flächen und einzelnen Punkten nicht einheitlich. Die Bilder werden unabhängig von der vorliegenden Auflösung, immer auf die Größe der zu texturierenden Fläche skaliert. Dies geschieht bei der Darstellung einzelner Punkte nicht. Um einen einheitlichen Gesamteindruck zu schaffen, wird hierfür die verwendete Punktgröße jeder Kachel inklusive ihrer Teilbereiche an die vorliegende Punktmenge angepasst. Liegen nur wenige Punkte vor, so wird die Punktgröße entsprechend erhöht. Erhöht sich die Punktanzahl (bspw. weil zusätzliche LOD Stufen

geladen wurden) so verringert sich die Punktgröße entsprechend und die Szene wirkt detaillierter, da nun kleinere, aber dafür mehr Punkte dargestellt werden. Dies entspricht exakt der Ersetzung einer gering aufgelösten Textur durch eine höher aufgelöste. Optisch ist der Effekt mit dem *progressive coding* des JPEG Bildstandards (CCITT (1992)) zu vergleichen, bei dem das eigentliche Bild bereits dargestellt werden kann bevor die Bilddatei komplett empfangen bzw. verarbeitet wurde. Mit zunehmendem Datenumfang steigt dabei der Detailgrad des Bildes, bis die Bilddatei vollständig geladen werden konnte. Abbildung 8.4 zeigt die Darstellung einer Punktwolke mittels adaptiver Punktgröße (obere Reihe). Zum Vergleich ist die gleiche Szene mit einer konstanten Punktgröße (untere Reihe) abgebildet. Im Falle der geringen Punktanzahl der niedrigen LOD Stufen ist bei konstanter Punktgröße das Modell kaum zu erkennen. Bei der Darstellung mittels adaptiver Punktgröße ist die Oberfläche des Modells unabhängig von der Anzahl der Punkte nahezu immer geschlossen und liefert somit einen ansprechenderen und in Verbindung mit texturierten Flächen konsistenteren visuellen Eindruck.

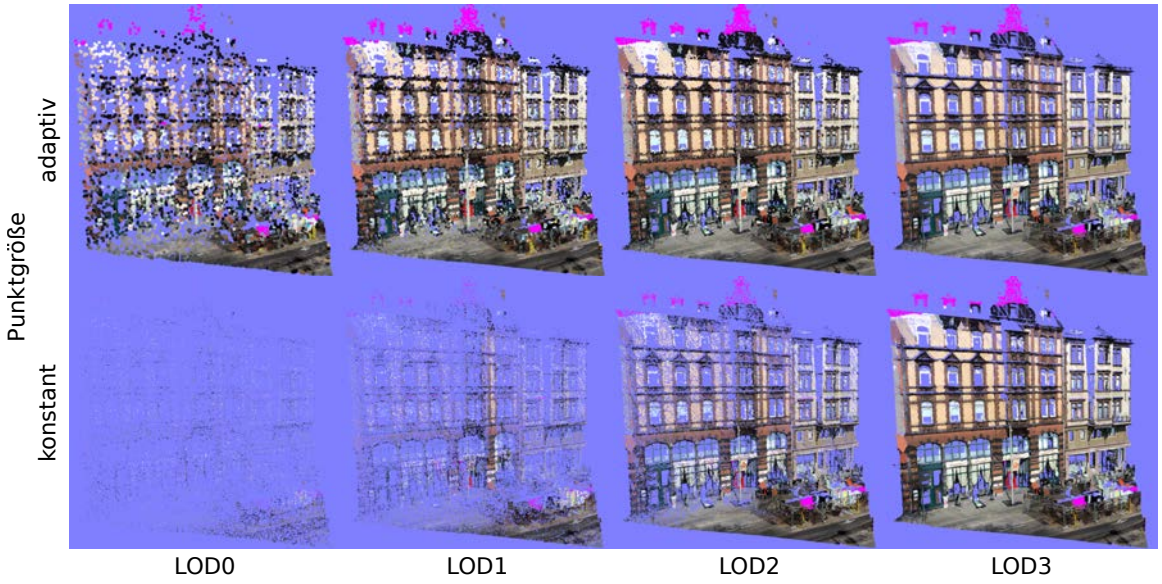


Abbildung 8.4: Darstellung eines Modells in unterschiedlichen LOD Stufen mit konstanter (untere Reihe) und adaptiver (obere Reihe) Punktgröße.

8.1.2 Performante Client-Server Kommunikation und Serialisierung

Zwischen den Client- und Serverinstanzen müssen unterschiedlichste Daten ausgetauscht werden, bspw. die Koordinaten der Punkte und Texturbilder, darüber hinaus aber auch globale Modelldaten wie die Trajektorie des Erfassungsfahrzeug oder die Verteilung und Struktur der jeweiligen LOD Pyramiden. Hinzu kommen noch die zugehörigen Anfragenachrichten vom Client an den Server, um die entsprechenden Daten zu erhalten.

Initial müssen die globalen Modelldaten übertragen werden. Um eine möglichst kurze Initialisierungsphase der Web-App zu gewährleisten, muss diese Information so kompakt wie möglich an den Client übertragen werden. Die interne Darstellung der LOD Pyramiden, muss dabei nicht notwendigerweise kompakt sein. Es genügt, eine möglichst kompakte Serialisierung der Pyramide zu erzeugen, welche an den Client übertragen wird. Ausgehend von der Modellierung der LOD Pyramide als Quadtree lässt eine bytebasierte Repräsentation des Baumes erzeugen, welche lediglich ein halbes Byte (vier Bit) pro Knoten benötigt. Bei der Serialisierung wird der Quadtree mittels Tiefensuche traversiert und pro Knoten die Existenz etwaiger Kinderknoten über vier Bit innerhalb des Byte codiert. Bei vier LOD Stufen ergibt sich ein Baum mit maximal $1 + 4 + 16 + 64 = 85$ Knoten pro Kachel, wobei ein Knoten einem LOD Teilbereich entspricht. Damit ergibt sich ein maximale Datenumfang von 43 Byte pro Kachel. Die genutzten Beispielprojekte besitzen eine Anzahl von bis zu 2000 Kacheln. Somit

ergibt sich ein Umfang bei der genannten Serialisierungsform von bis zu 84KB für ein komplettes Projekt. Da in der Regel nicht alle LOD Stufen immer voll besetzt sind (was eine Übertragung der Struktur ja erst sinnvoll macht) ist der typische Gesamtumfang nochmals deutlich kleiner.

Für kleinere Datenmengen, wie die globalen Modelldaten ist die von GWT bereitgestellte RPC (kurz für: Remote-Procedure-Call) Schnittstelle völlig ausreichend. Sollen allerdings Massendaten wie die Punktwolken und Texturbilder übertragen werden, stößt diese Form der Übertragung schnell an ihre Grenze, da ein relativ aufwendiger Serialisierungs- bzw. Deserialisierungsprozess stattfindet. Darüber hinaus ist es nicht möglich, Anfragen gebündelt zu verschicken und einzelne Antwortdaten zu bekommen, sobald sie vom Server geladen wurden. Es besteht also keine Möglichkeit einer *Push* Kommunikation, sondern nur *Pull*. Das heißt, dass jede Anfrage für jedes Datum separat gesendet und empfangen werden muss, was bei den einzelnen kleinen LOD Bereichen zu einem großen Overhead führt und daher für diesen Einsatzzweck nicht geeignet ist. Darüber hinaus gibt es ebenfalls keine Möglichkeit, eine RPC Anforderung zu ändern oder gar abzubrechen.

Um eine entsprechend performante Push-basierte Kommunikation zwischen Client und Server umzusetzen, wurde für die Web-App neben der GWT RPC-Kommunikation ein weiterer Kommunikationskanal implementiert. Dieser nutzt das von Fette und Melnikov (2011) in RFC6455 vorgeschlagene *WebSocket*-Protokoll und ermöglicht einen direkteren Push-basierten Datenaustausch zwischen Client und Server.

Das umgesetzte Kommunkationsprotokoll gestaltet sich wie folgt. Ausgehend von der globalen Modellinformation kann bestimmt werden, welche Kacheln in welchem Detailgrad existieren. Darüber hinaus ist bekannt, welche Daten bereits vorhanden sind und welche angefragt wurden. Wird nach der Änderung der Kameraposition festgestellt, dass Teile des Modells mit einem höheren Detailgrad dargestellt werden sollen, so werden sämtliche anzufragende Teilbereiche in einer Nachricht gebündelt an den Server über einen WebSocket geschickt (vgl. Abschnitt 8.1.3). Dieser lädt die Daten und überträgt (pusht) jeden LOD Teilbereich, sobald er von der Platte geladen wurde, sofort an den Client. Abbildung 8.5 stellt die Kommunikationsvarianten GWT-RPC (links) und WebSocket (rechts) in einem Sequenzdiagramm qualitativ gegenüber. Obgleich der serverseitige Ladevorgang einen WebGL konformen Speicherblock erzeugt, verzögern die GWT-internen Serialisierungs- und Deserialisierungsvorgänge die Übertragung bzw. die finale Darstellung der Daten.

Im Falle der Kommunikation via WebSockets entfallen die Serialisierungsschritte, da die serverseitige Serialisierung der Daten bereits darin besteht, einen WebGL konformen Speicherpuffer zu füllen und direkt zu übertragen. Nach dem clientseitigen Empfang des Puffers kann dieser ohne weitere Prozessierung in den Grafikkartenspeicher übertragen werden. Hierfür muss die Größe des übertragenen Puffers ein Vielfaches der Größe eines Punktes entsprechen (engl.: Message Header Aligning). Die Daten eines Punktes strukturieren sich dabei wie folgt: x als *short*, y als *short*, z als *float*, sowie Farbe als RGBA mit vier Byte. Dies ergibt in der Summe ($2+2+4+4 = 12$ Byte). Da die x, y Koordinaten nur relativ zum Kachelursprung vorliegen müssen, können hier jeweils zwei Byte bei einer Kodierung mittels Short anstelle von Float eingespart werden, ohne sichtbare Genauigkeitsverluste. Der somit auf 67% (12 anstelle von 16 Byte pro Punkt) reduzierte Datenumfang ermöglicht eine schnellere Übertragung der Daten und spart Clientseitig wertvolle Ressourcen auf der Grafikkarte.

Mit dem gezeigten Serialisierungsschema stellt die Bandbreite des Übertagungskanals (neben der Grafik-Hardware) die einzige maßgebende Komponente für die Geschwindigkeit der Darstellung dar. Dies stellt einen wesentlichen Faktor für die Skalierbarkeit des vorgestellten Systems dar. Neben der Vermeidung des Serialisierungsoverheads, werden darüber hinaus alle Datenanfragen gebündelt an den Server übertragen, so dass weiterer Overhead eingespart wird.

Neben dem Zeitvorteil der gezeigten WebSocket-basierten Kommunikation bietet diese einen weiteren entscheidenden Vorteil. Im Gegensatz zu den RPC Anfragen können diese bis zur Bearbeitung aktualisiert oder sogar verworfen werden. Dies ist unter anderem dann nützlich, wenn schnelle oder großräumige Positionsveränderungen der Kamera stattfinden und sich die Prioritäten für die jeweiligen Anfragen entsprechend ändern oder angefragte Bereiche unter Umständen sogar obsolet werden. Im Falle der GWT-RPC Anfragen kann dies nicht erfolgen, was zum Blockieren wertvoller Ressourcen führt und eine zügige Übertragung und Visualisierung der aktuellen Umgebung verhindern kann.

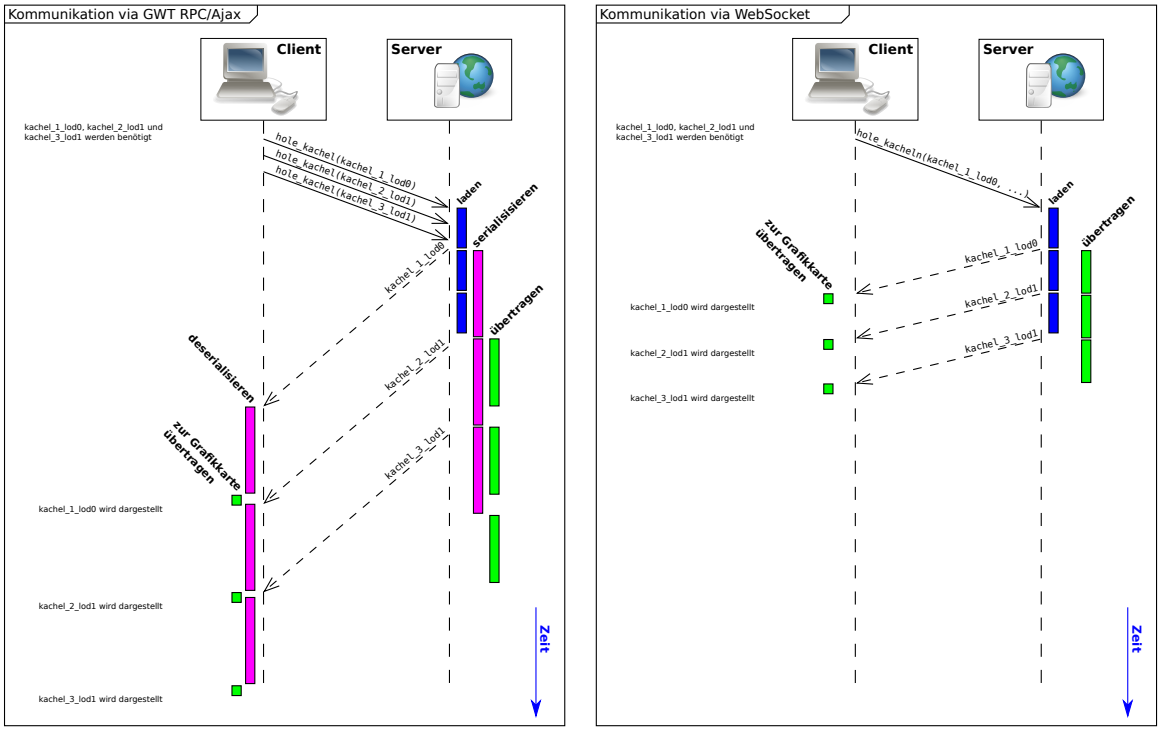


Abbildung 8.5: Sequenzdiagramme der Kommunikation via GWT-RPC (links) und via WebSockets (rechts).

8.1.3 Scheduling der LOD-Daten

Gilt es die für eine bestimmte Kameraposition und -orientierung darzustellenden Daten anzufragen, so kann dies auf verschiedenste Weise geschehen. Die Entscheidung, ob ein Bereich angefragt werden muss, ergibt sich bei der Traversierung der LOD Pyramidenstruktur während des Renderings. Versucht der Renderprozess einen Bereich darzustellen, dessen Daten lokal nicht vorhanden sind, so müssen diese vom Server angefragt werden. Dafür bieten sich im Wesentlichen zwei Optionen an. Entweder werden einzelne Bereiche direkt angefragt (genau dann, wenn ihr Fehlen während der Traversierung festgestellt wird) oder es werden alle Anfragen gebündelt am Ende des Renderschrittes verschickt. Wie bereits in Abschnitt 8.1.2 beschrieben, geht das Senden separater Anfragen für jeden Bereich mit einem großen Overhead einher. Daher werden alle Anfragen für fehlende Bereiche während der Renderphase gesammelt und am Ende gebündelt an den Server übermittelt.

Neben dem Schema der Anfrageübermittlung ist es darüber hinaus relevant, in welcher Reihenfolge die Anfragen vom Server abgearbeitet werden. Unter visuellen Aspekten ist es vorteilhaft, Daten unmittelbar nach dem Setzen bzw. Ändern der Kameraposition oder -orientierung präsentiert zu bekommen. Dafür müssen angefragte Bereiche in Blickrichtung der Kamera, sowie geringer Entfernung zur Kamera priorisiert werden. Darüber hinaus ist ein möglichst großer, grober, jedoch unmittelbarer Überblick über die Umgebung ebenfalls vorteilhaft. Abschließend liegt eine hohe Wahrscheinlichkeit vor, dass der Nutzer am aktuellen Kamerastandpunkt die Orientierung der Kamera ändert, bspw. um sich umzusehen. Um nicht erst Daten anzufragen wenn dies geschieht werden bereits vorher weitere Daten der näheren Umgebung angefragt. Diese vier genannten Priorisierungen (Blickrichtung, Entfernung, Level-of-Detail und Sichtkegel) werden zur Berechnung eines *Scores* verwendet, welcher an jeden angefragten Bereich angehängt wird. Serverseitig werden die Anfragen bezüglich des Scores absteigend sortiert, sodass Bereiche mit dem höchsten Score als erstes geladen und an den Client zur Visualisierung geschickt werden. Abbildung 8.6 veranschaulicht die zu kombinierenden Priorisierungen.

Die Entfernungspriorität (0 - max. Renderdistanz) sinkt dabei linear mit der Distanz des Bereichmittelpunktes zur Kameraposition. Somit werden Bereiche nahe der Kameraposition als erstes geladen

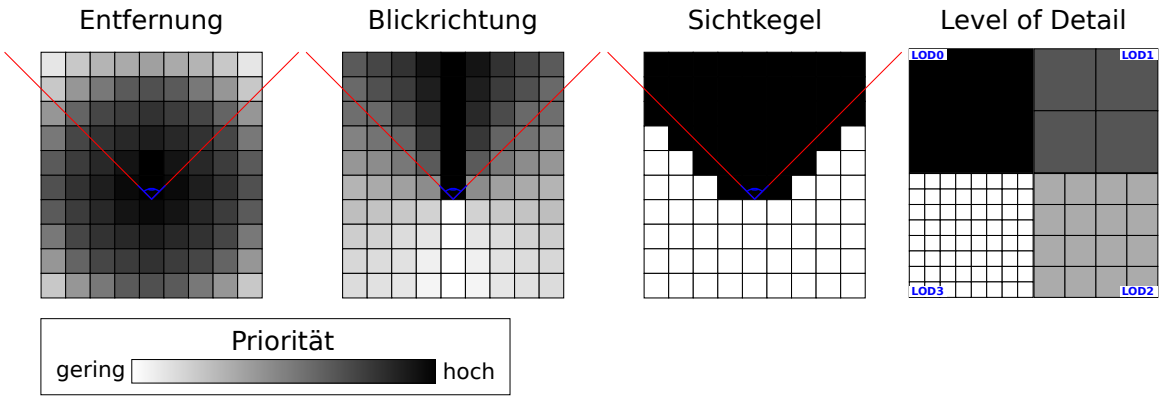


Abbildung 8.6: Übersicht über Priorisierung nach Entfernung, Blickrichtung, Sichtkegel und Level-of-Detail (v.l.n.r.).

und angezeigt. Die Blickrichtungspriorität (0° bis 180°) sinkt mit steigendem Winkel des Bereichmittelpunktes bezogen auf die Blickrichtung. Dies bewirkt, dass Bereiche in der Mitte des Blickfeldes bevorzugt werden. Bei der Sichtkegelpriorität (0 oder 1) handelt es sich um ein binäres Verhältnis. Liegt einer der Eckpunkte eines Bereichs im Sichtkegel, so ergibt sich die maximale Priorität. Liegen alle Eckpunkte außerhalb, so ergibt sich die minimale Priorität. Damit wird verhindert, dass Daten außerhalb des Sichtkegels geladen werden bevor der komplette Sichtkegelbereich geladen wurde. Im Falle der Priorität des Level of Details (0 - 3) sinkt diese linear mit steigendem LOD. Dies bevorzugt Bereiche mit geringem LOD und wirkt der Entfernungs- und Blickrichtungspriorisierung entgegen, um schnell einen groben Überblick über die Umgebung zu gewährleisten.

Der finale Priorisierungsscore eines Bereichs ergibt sich dabei additiv aus den Einzelpriorisierungen, wobei jede Einzelpriorisierung anteilig gewichtet wird. Um über die Sichtkegelpriorisierung unabhängig von den übrigen zu gewährleisten, dass alle Bereiche außerhalb des Sichtkegels erst nach denen innerhalb des selbigen geladen werden, erhält diese eine Gewichtung von 50%. Die Gewichte der übrigen Priorisierungen wurden dabei wie folgt verteilt: Entfernung (20%), Blickrichtung (10%) und Level of Detail (20%). Abbildung 8.7 zeigt den Score aller Teilbereiche für alle vier LOD Stufen. Deutlich zu erkennen ist die Sichtkegelpriorisierung, welche allen Bereiche außerhalb des Sichtkegels geringe Prioritäten (0-128) und innerhalb hohe Prioritäten (128-255) zuweist. Der Effekt der Level of Detail Priorisierung zeigt sich beim direkten Vergleich der Scores zweier LOD Stufen. Hier erhalten Bereiche mit geringem LOD eine wesentlich höhere Priorität als Daten mit höherem Detailgrad. Beispielsweise liegt die niedrigste Priorität aller LOD0 Bereiche bei 220, was der höchsten Priorität der Bereiche der LOD2 Stufe entspricht. Damit werden zunächst sämtliche LOD0 (und einige LOD1) Daten für einen groben Überblick der Umgebung geladen und erst dann der Detailgrad weiter erhöht.

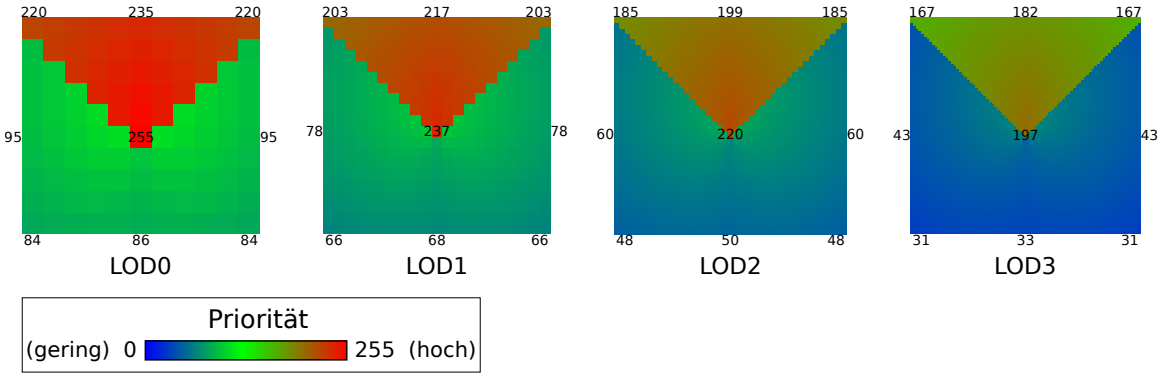


Abbildung 8.7: Ergebnis der gewichteten Überlagerung der Einzelpriorisierungen unterteilt nach LOD-Stufen.

Abbildung 8.8 zeigt die Statistik zur dargestellten Szene in der Web-Applikation. Dabei wurden 1160 LOD Bereiche übertragen, welche etwa 4,3 Mio. Punkte umfassen. Gerendert wurden davon lediglich 1,4 Mio. Punkte aus 152 Bereichen.

Abbildung 8.8: Statistik zur dargestellten Szene.

8.1.4 GUI Responsiveness

Ein wichtiges Ziel einer interaktiven Visualisierung ist eine konstante, möglichst hohe Bildwiederholrate (engl. framerate), sowie eine unmittelbare Reaktionsfähigkeit bei Nutzeraktionen (engl. responsiveness). Findet eine zeitintensive clientseitige Verarbeitung der Daten statt, bpsw. eine Deserialisierung, so kann weder eine konstante Bildwiederholrate, noch eine unmittelbare Reaktionsfähigkeit auf Nutzereingaben gewährleistet werden. Die Gründe dafür liegen im Wesentlichen an der Form der Ereignisverarbeitung (engl. Event-Handling) von JavaScript. JavaScript ist von Natur aus auf eine Einzel-Thread (Thread - engl. für Ausführungsstrang) Verarbeitung ausgelegt. Sämtliche Ereignisse (Rendern einer Szene, Nutzereingaben, Empfang und Verarbeitung von Daten, usw.) werden dabei in eine Ereigniswarteschlange (engl. Event-Loop) eingereiht. Diese Warteschlange wird dabei kontinuierlich abgearbeitet . Benötigt die Verarbeitung eines Ereignisses viel Zeit, so verschiebt sich die Verarbeitung der folgenden Ereignisse entsprechend und eine flüssige Darstellung und Interaktion ist nicht mehr gegeben.

Das in Abschnitt 8.1.2 gezeigte Serialisierungsschema (Serverseitige Erstellung und Übertragung eines WebGL konformen Puffers) verhindert eine aufwendige Deserialisierung und Konvertierung der Daten für die Darstellung. Jedoch findet dies lediglich bei Punktdaten Anwendung. Grundsätzlich ließe sich dieser Ansatz auch auf die zu übertragenden Bilddaten ausweiten, indem die Bilddaten ebenfalls Serverseitig in einen WebGL konformen Puffer geschrieben und anschließend übertragen werden. Dies erfordert jedoch die vollständige Dekomprimierung der Bilddaten auf Serverseite. Wie bereits in Abschnitt 7.1.4 zur Speichereffizienz der hybriden Modelle gezeigt, liegt das Verhältnis des Speicherumfangs zwischen den komprimierten und dekomprimierten Bilddaten bei etwa eins zu vier. Somit müsste hier die vierfache Menge an Daten übertragen werden, was bei der Menge an Bilddaten, die zur Visualisierung genutzt werden, unnötig wertvolle Übertragungskapazitäten verschwendet. Um den Komprimierungsvorteil auch für die Übertragung nutzen zu können, müssen die Bilder auch komprimiert an den Client übertragen und dort dekomprimiert werden. Dies führt jedoch zum eingangs beschriebenen Verlust der flüssigen Darstellung und Interaktion, beim Laden der Daten der aktuellen Szene.

Um dennoch eine flüssige Darstellung zu gewährleisten, muss die Dekomprimierung (im Falle von PNG als INFLATE bezeichnet) der Bilddaten außerhalb des Event-Loops geschehen. Moderne Browser unterstützen dafür die von (Hickson, 2012) vorgeschlagene WebWorker API. Dabei können, ähnlich der Mehrkern CPU Variante aus Abschnitt 3.1.2.1, mehrere nebenläufige *Worker* bzw. Ausführungsstränge erstellt werden. Diese werden parallel zu JavaScripts Event-Loop ausgeführt und blockieren diesem damit nicht. Abbildung 8.9 stellt die Verarbeitung via Event-Loop und die im Rahmen dieser Arbeit entwickelte WebWorkern Variante gegenüber. Für eine bestmögliche Ausnutzung der Ressourcen sollte die zur Verfügung stehende Anzahl an Prozessoren ermittelt werden. Stehen mehr als zwei Prozessoren zur Verfügung, wird nicht nur eine flüssigere Darstellung erreicht, es können darüber hinaus auch mehrere Bilder gleichzeitig dekomprimiert werden, was die Zeit zwischen Empfang und Darstellung der Bilder noch einmal erheblich verkürzt. Die Anzahl der Prozessoren lässt sich mittels der JavaScript API über das Feld `navigator.hardwareConcurrency` ermitteln. Anschließend wird ein Pool an WebWorkern entsprechend der ermittelten Anzahl, abzüglich des Event-Loops, erzeugt. Darüber hinaus wird eine Warteschlange eingerichtet, welche von den WebWorkern abgearbeitet wird. Sobald zu dekodierende Bilder eintreffen, besteht die einzige Aufgabe des Event-Loops darin, den empfangenen Datenblock in die Warteschlange einzureihen, wobei eine Aufgaben-ID generiert wird, welche die spätere Zuordnung des dekodierten Bildes ermöglicht.

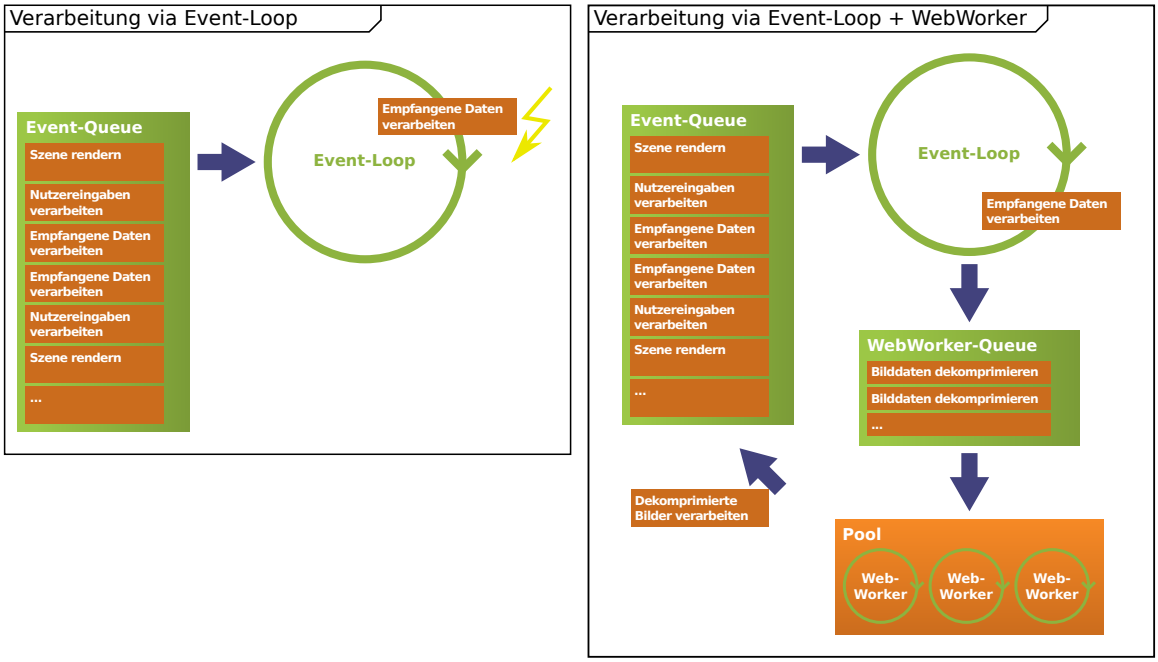


Abbildung 8.9: Abarbeitung der Event Warteschlange via Event-Loop (links) wobei Zeitaufwendige Dekomprimierung von Bildern zu Verzögerungen in der Abarbeitung führen und Abarbeitung mit WebWorker Pool zur parallelen Dekomprimierung der Bilddaten (rechts).

Nach erfolgter Dekomprimierung der Bilddaten können diese in den Speicher der Grafikkarte geladen und visualisiert werden. Die Zuordnung zum zugehörigen Polygon geschieht, wie bereits beschrieben, über die Vergabe einer ID bei der Einreihung der zu dekomprimierenden Bilddaten in die Warteschlange.

8.1.5 Navigation und Nutzerinteraktion

Die Navigations- und Interaktionsmöglichkeiten eines Nutzers lassen sich in zwei wesentliche Kategorien unterteilen. Zum einen, das für die Erkundung der Umgebung notwendige Ändern der Position und Orientierung der virtuellen Kamera und zum anderen weiterführende Interaktionen. Ersteres sind Basisfunktionalitäten und müssen so intuitiv und einfach wie möglich gestaltet werden, um nicht

fachkundige Nutzer nicht zu überfordern. Als weiterführende Interaktion wurde eine Möglichkeit umgesetzt, in der dargestellten Szene Distanzen (bspw. die Straßenbreite oder die Höhe von Bäumen und Gebäuden, um nur einige zu nennen) zu bestimmen bzw. zu messen. Dies stellt eine besondere Herausforderung dar, da die Daten zum einen in hybrider Form (Punkte und texturierte Polygone) und zum anderen nur im Speicher der Grafikkarte vorliegen.

Beim Erkunden von virtuellen Welten stehen im Wesentlichen sechs Freiheitsgrade zur Verfügung (Mine (1996)): Position in X, Y und Z, sowie Orientierung als Roll, Nick und Gier Winkel. Diese lassen sich selbst für erfahrene Nutzer nur schwer mit einer gewöhnlichen Computermaus als Eingabegerät manipulieren (Hachet u. a. (2008)). Daher ist es notwendig die Navigation durch die virtuelle Umgebung zu vereinfachen und auf die Möglichkeiten einer Computermaus zu reduzieren. Der einfachste Weg dies zu erreichen, ist die Anzahl der Freiheitsgrade zu reduzieren.

Während die Änderung der Orientierung bzw. Blickrichtung am intuitivsten über das Bewegen der Maus erfolgt, gibt es für die Änderung der Position kaum eingängige Möglichkeiten. Um dennoch die Maus als einziges Eingabegerät zu unterstützen, wurden die drei Freiheitsgrade der Position auf einen Freiheitsgrad reduziert. Um sich dabei nach wie vor sinnvoll durch die Umgebung bewegen zu können, wurde die Kameraposition auf beliebigen Positionen der Trajektorie des Erfassungsfahrzeug beschränkt. Dies ermöglicht die Änderung der Position durch die Wahl eines Punktes auf der Trajektorie. Dafür wurde zwei Möglichkeiten in der Web-App umgesetzt. Zum einen kann über das Mausrad zur nächsten bzw. vorherigen Position auf der Trajektorie gesprungen werden (die Sprungweite lässt sich vom Nutzer dabei individuell anpassen), zum anderen kann auf einer Übersichtskarte eine beliebige Position angeklickt werden und die Kameraposition wird auf die nächstgelegene Position auf der Trajektorie verschoben (vgl. Abbildung 8.3). Beides stellt sich in der Praxis als durchaus einfach und intuitiv heraus. Da die Punktdichte in direkter Umgebung des Erfassungsfahrzeugs ohnehin am höchsten ist, ist der Verlust der komplett freien Bewegung in der virtuellen Umgebung vernachlässigbar. Neben dem manuellen Springen mittels Mausrad besteht darüber hinaus noch die Möglichkeit einer *Fahrt* durch die Punktwolke, wobei automatisch zum nächsten Punkt auf der Trajektorie gesprungen und die Szene neu gerendert wird.

Bei der Wahl des virtuellen Kameramodells, welches die Freiheitsgrade der Orientierung (Roll, Nick und Gier Winkel) beeinflussen lässt, wurde ein *Orbit Kamera* Modell gewählt. Das Modell setzt sich dabei aus einer Kameraposition und einem *Point of Interest* (POI) zusammen. Zur Änderung der Orientierung wird die Kameraposition auf einer virtuellen Kugel um den POI herum bewegt. Das mentale Modell dahinter wird von Ware und Osborne (1990) als *scene in hand* bezeichnet. Zur weiteren Vereinfachung wurde die Möglichkeit des Rollens um die Blickrichtung entfernt. Damit existieren für die Orientierung nur noch zwei Freiheitsgrade, welche intuitiv und direkt über die Bewegung der Maus in X und Y Richtung manipuliert werden können. Um der stark eingeschränkten Bewegungsfreiheit entgegenzuwirken wurde darüber hinaus noch die Möglichkeit hinzugefügt den Abstand der Kamera vom POI zu ändern. Als Nutzerinteraktion wurde hier abermals das Mausrad gewählt, nur in Kombination mit einer der Maustasten.

Abbildung 8.10 gibt einen Überblick über das beschriebene Kameramodell sowie die dem Nutzer zur Verfügung stehenden Freiheitsgrade. Alle vier Freiheitsgrade werden mit Mausinteraktionen abgedeckt. Die Reduktion auf vier anstatt sechs Freiheitsgrade und die Maus als einziges Eingabegeräte machen die Navigation gerade für nicht fachkundige Nutzer direkt und intuitiv. Insbesondere ist ein *Verirren* in der Punktwolke nicht möglich, da die möglichen Positionen auf die Erfassungstrajektorie beschränkt wurden.

Über die bloße Navigation hinaus wurde eine erweitere Nutzerinteraktion prototypisch umgesetzt. Dabei handelt es sich, wie eingangs beschrieben, um die Möglichkeit Abstände innerhalb der dargestellten Szene zu bestimmen. Dabei wählt der Nutzer zwei Punkte durch anklicken aus, woraufhin der Abstand dieser Punkte angezeigt wird. Dies hat mehrere Herausforderungen. Zunächst muss über die Position des Mausklicks in der Abbildung der Szene die echte räumliche Position ermittelt werden. Darüber hinaus liegen die Punktkoordinaten aus Speichergründen lediglich als Puffer im Grafikkartenspeicher. Abschließend muss dem Fakt Rechnung getragen werden, dass es sich bei den dargestellten Modellen um hybride Modelle bestehend aus Punkten und texturierten Flächen handelt.

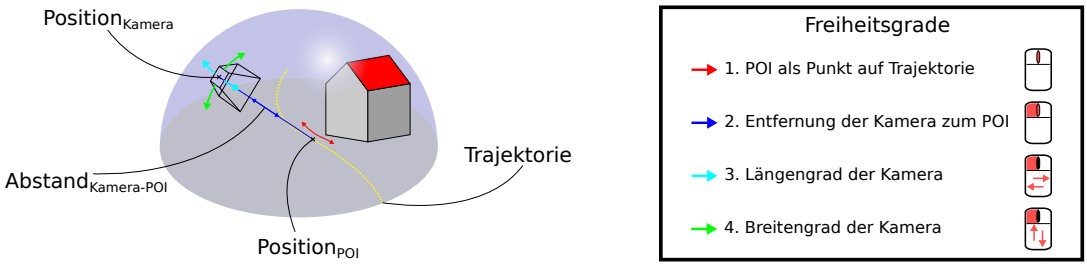


Abbildung 8.10: Übersicht über das gewählte Kameramodell sowie die sich daraus ergebenen Freiheitsgrade.

Die Auswahl von gerenderten Objekten wird als *Picking* bezeichnet. Die häufigste Umsetzung besteht darin die IDs der Objekte mittels der gleichen Transformationsmatrix in einen Framebuffer (Bild mit IDs analog zur dargestellten Szene) zu schreiben. Bei der Auswahl (Klick) eines beliebigen Pixels des gerenderten Bildes kann die IDs aus dem genannten Framebuffer an gleicher Stelle ausgelesen werden. Über die ausgelesene ID werden dann die Daten des jeweiligen Punktes bzw. Objektes ermittelt. Diese Standardvorgehensweise ist in der umgesetzten Web-App jedoch nicht möglich. Zum einen besteht die Szene aus mehreren, von einander unabhängigen Bereichen bzw. Kacheln. Daher würden neben einer Punkt-ID zusätzlich noch Bereichs-IDs benötigt werden. Darüber hinaus handelt es sich um ein hybrides Modell, welche neben Punkten noch texturierte Flächen umfasst. Bei dem genannten Ansatz würde eine ID pro Fläche ermittelt, nicht jedoch die genaue Lage auf der Fläche. Neben den genannten Problemen ist es nach der OpenGL|ES Spezifikation (die Basis für WebGl) nicht möglich die, in Form von Vertex-Buffer-Objekten (kurz: VBO), in den Grafikkartenspeicher geschriebenen Daten wieder auszulesen. Dass heißt, selbst wenn der exakte Punkt ermittelt werden konnte, so besteht keine Möglichkeit die zugehörigen Koordinaten zu erhalten, da diese lediglich im Speicher der Grafikkarte vorliegen.

Aus den genannten Gründen wurde ein eigener *Picking*-Ansatz entwickelt und umgesetzt. Da keine VBOs ausgelesen werden können, müssen die (x, y, z) Koordinaten der ausgewählten Punkte demnach, mittels der beschriebenen Shader, direkt in den auslesbaren Framebuffer geschrieben werden. Dies hat mehrere Vorteile. Zum einen müssen die Koordinaten nicht über etwaige Punkt- bzw. Objekt-IDs ermittelt werden, zum anderen werden die Koordinaten im Falle der Flächen interpoliert und man erhält die exakten Koordinaten auf der Fläche. Die maximale Wortbreite des Framebuffers beträgt jedoch lediglich 32 Bit. Damit stünden den drei Koordinatenwerten lediglich 10 Bit pro Wert zur Verfügung, was zu einem starken Genauigkeitsverlust führt. Um dies zu verhindern, werden drei Renderdurchgänge durchgeführt. Dabei wird jeweils ein Koordinatenwert pro Durchgang in den Framebuffer geschrieben und direkt ausgelesen. Damit stehen für jeden Wert die vollen 32 Bit zur Verfügung und es tritt kein Genauigkeitsverlust auf. Auswirkungen auf die Bildwiederholrate oder gar Qualität der Darstellung hat dieses Vorgehen nicht. Da es sich um drei gewöhnliche Renderdurchläufe handelt und die Kamera während des Pickings üblicherweise nicht bewegt wird, ist kein Abfall der Bildwiederholrate zu erwarten. Der zeitliche Aufwand des Pickings ist ebenfalls vernachlässigbar, da ein Renderdurchlauf, in Abhängigkeit des genutzten Systems, üblicherweise weniger als 100 Millisekunden beträgt.

Die Nutzerinteraktion zur Auswahl der Punkte besteht im Klicken bei gedrückter STRG-Taste. Auf diese Weise kann ein ganzer Pfad aus Punkten gewählt werden, welcher, wie Abbildung 8.11 zeigt, als Feedback für den Nutzer in die Szene gerendert wird. Die Koordinaten der selektierten Punkte, sowie die Längen der einzelnen Pfadsegmente werden im rechten Bereich aufgelistet. Auf diese Weise können beliebige Objekte bzw. Strecken vermessen werden.

8.2 2D Visualisierung

Wie schon bei der zuvor beschriebenen 3D Visualisierung soll bei der Darstellung der 2D Fassadenmodelle von einem Erkundungsszenario ausgegangen werden. Jedoch ist hier ein *mobiles* Einsatzfeld

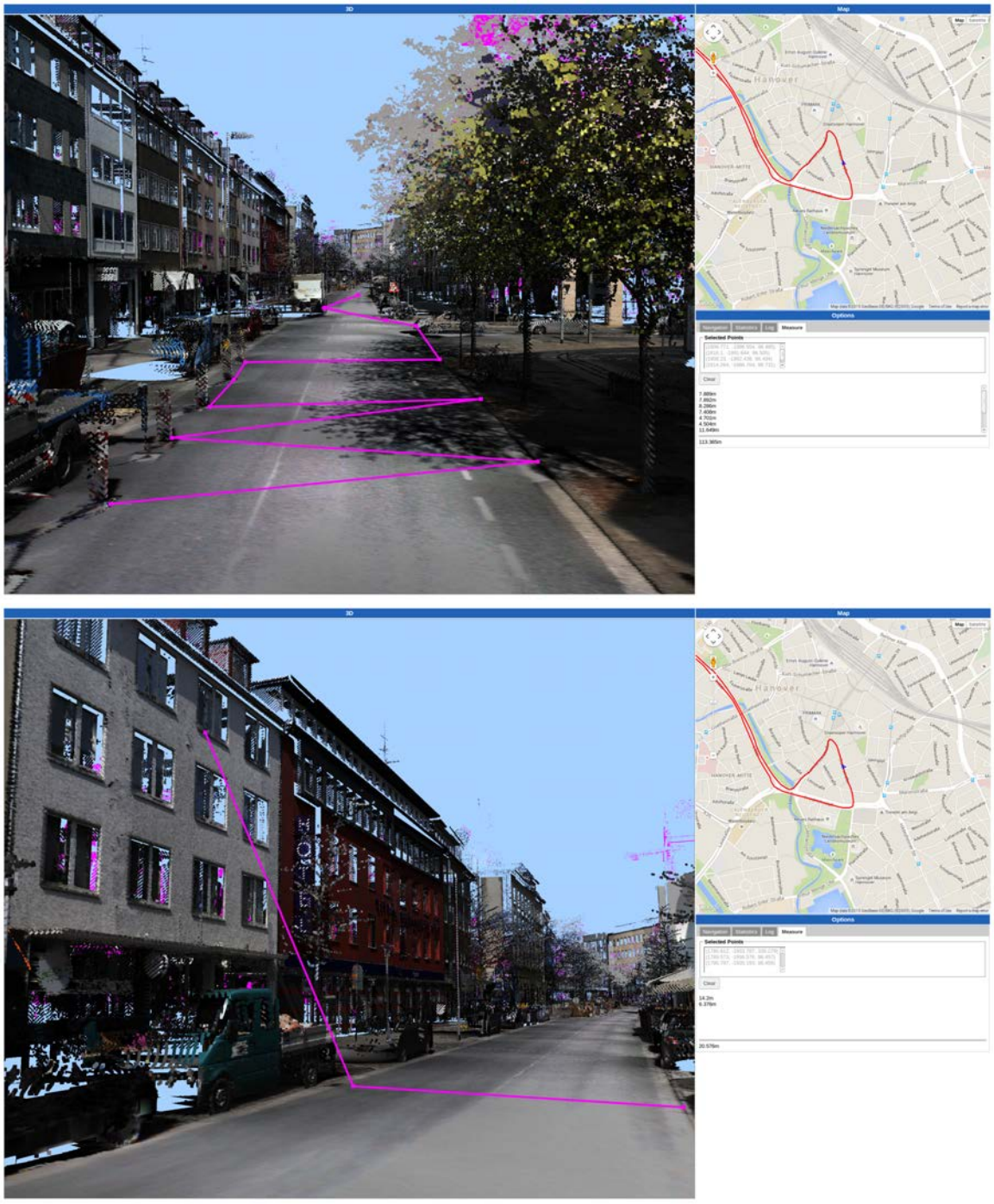


Abbildung 8.11: Beispiel für die umgesetzte Nutzerinteraktion Messen *in der Web-App.*

das Ziel, um beispielsweise einen direkten Vergleich bereits erfasster Daten während einer erneuten Messfahrt zu ermöglichen. Wie auch Abbildung 8.12 aufzeigt, leitet sich aus der mobilen Nutzung die Verwendung eines Mobilen Endgerätes ab. Darüber hinaus ist eine mobile Datenverbindung nicht notwendigerweise gegeben, daher wird von einer *offline* Nutzung ausgegangen. Mit der Nutzung eines mobilen Endgerätes geht für gewöhnlich auch eine Touch-basierte Bedienung einher, welches hier ebenfalls als Systemanforderung gezählt wird. Da bereits viele Module in der Programmiersprache Java umgesetzt wurden, wird als Zielplattform *Android* gewählt, welche ebenfalls auf Java basiert. Dies ermöglicht eine hohe Wiederverwendbarkeit der bereits umgesetzten Funktionen und Module.

Abbildung 8.12: Übersicht über den Anwendungsfall und den daraus abgeleiteten Systemanforderungen, sowie der Zielplattform für die Visualisierung der 2D Fassadenmodelle.

Die Grundlage der 2D Visualisierung bildet die in Abschnitt 2.2.4.3 beschriebene Parallax Scrolling Technik. Dargestellt werden dabei die Ebenen- bzw. Fassadenmodelle, als Ergebnis der Verfahren aus Abschnitt 7.2. Im folgenden wird auf die Umsetzung einer auf Parallax-Scrolling basierenden Android App, einschließlich der Datenhaltung, sowie Touch-basierten Nutzerinteraktionen eingegangen. Abschließend wird in Abschnitt 8.2.2 ein Beleuchtungsmodell vorgestellt, welches die Darstellung sichtbar aufwertet und noch realistischer erscheinen lässt. Einen Eindruck über die entwickelte App verleiht Abbildung 8.13. Dargestellt sind drei Screenshots desselben Straßenzuges, wobei jeweils unterschiedliche Kamerastandpunkte gewählt wurden. Die Überlappung der übereinander geordneten Darstellung entspricht dabei dem Versatz der jeweiligen Kameraposition.

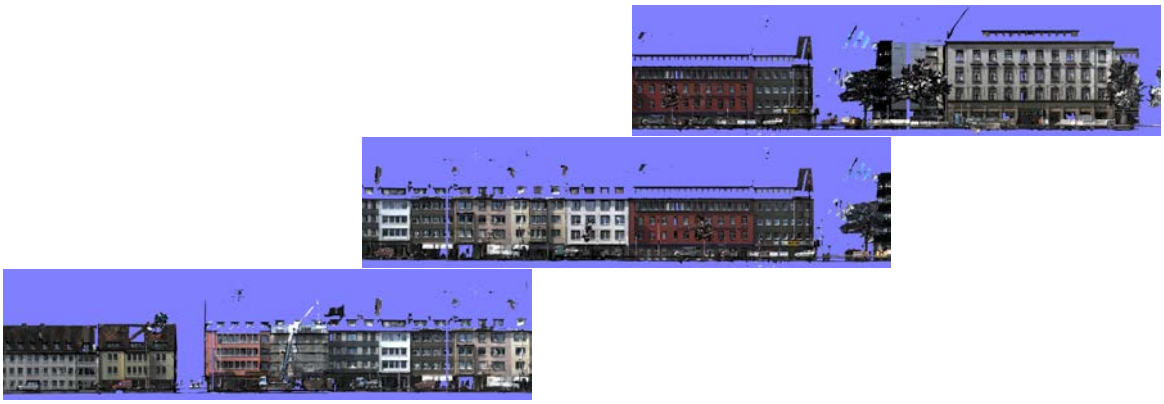


Abbildung 8.13: Screenshots der Android-App.

8.2.1 Parallax Scrolling Visualisierung via Android-App

Eine Visualisierung mittels Parallax Scrolling stellt eine eher ungewöhnliche Darstellungsform dar. Obgleich bei vielen mobilen Spielen eingesetzt, existiert kein frei verfügbares Framework für die gewählte Zielplattform. Daher muss die Technik von Grund auf für die Zielplattform umgesetzt werden. Da es sich dabei um die Darstellung von hintereinander gelegten teiltransparenten Bildern handelt, bieten sich zwei Basistechnologien für die Umsetzung an: die Nutzung der Standard 2D Renderpipeline oder die hardwarebeschleunigte 2D/3D Renderpipeline via OpenGL|ES. Je nach Größe des darzustellenden Modells bzw. der bei der Modellerzeugung gewählten Parameter kann der Umfang der zugehörigen Bilder mehrere hundert Megabyte erreichen. Bei der Nutzung der gewöhnlichen 2D Renderpipeline kann dies zu Speicherproblemen führen, was ein aufwendigeres Datenmanagement notwendig macht. Dies verhindert allerdings das beabsichtigte *schnelle* Browsing durch die Daten. Daten, welche in den Grafikspeicher via OpenGL|ES geladen werden, unterliegen großzügigeren Grenzen und machen ein zusätzliches Datenmanagement überflüssig. Darüber hinaus ermöglicht die hardwarebeschleunigte Renderpipeline deutlich höhere Bildwiederholraten und erlaubt somit die

Nutzung detaillierter Modelle. Daher wird für die Umsetzung der Parallax Scrolling Visualisierung die OpenGL|ES basierte Renderpipeline genutzt.

Dazu müssen zunächst die zu ladenden Modelldaten durch den Nutzer ausgewählt werden. Da es sich um ein Nutzungsszenario ohne aktiver Datenverbindung handelt, müssen sämtliche Modelldaten zunächst auf das Endgerät übertragen werden. Neben der Auswahl des eigentlichen Modells muss sich der Nutzer für eine Straßenseite entscheiden, da aufgrund der Darstellungsform jeweils nur eine Straßenseite visualisiert werden kann. Damit reduziert sich der Datenumfang nochmals auf etwa die Hälfte und ermöglicht damit die Darstellung umfangreicherer Modelle. Um den Speicherplatzbedarf weiter zu reduzieren, werden die vorliegenden Bilder zunächst komprimiert. Um nicht nur einen Vorteil bei der persistenten Speicherung analog zu bspw. Deflate zu erhalten (vgl. Abschnitt 7.1.4), sondern auch wertvollen Speicher auf der Grafikeinheit einzusparen, wird das von OpenGL|ES unterstützte Kompressionsformat ETC1 (nativ ab Version 3.0, Lipchak (2014) und via Erweiterungen bereits ab Version 2.0, Munshi und Leech (2010)) angewendet. ETC1 steht für *Ericsson Texture Compression* und wurde von Ström (2008) auf Basis von *iPACKMAN* (Ström und Akenine-Möller (2005)) entwickelt. Der Vorteil liegt dabei darin, dass die Bilddaten nicht nur komprimiert persistent als Datei vorliegen, sondern ebenfalls komprimiert in den Grafikspeicher übertragen werden. Dabei wurde das Kompressionsverfahren dahingehend entwickelt, dass keine merklichen Geschwindigkeitseinbussen beim Zugriff auf die Daten entstehen. Sämtliche ETC1 komprimierten Bilder, zusammen mit den dazugehörigen Metadaten (wie Lage/Position, Ebenenzuordnung, etc.) werden abschließend in einer Modellarchivdatei (nochmals deflatekomprimiert) gespeichert.

Für den Effekt der Bewegungsparallaxe wird eine perspektivische Projektion benötigt, welche die Größe der Ebenen entsprechend ihrer Entfernung skaliert. Je weiter die Ebene von der Kamera entfernt ist, desto kleiner wird sie dargestellt. Erst dadurch lassen sich die Vordergrundebenen mit einer höheren Geschwindigkeit verschieben als die dahinter liegenden, da sie durch die perspektivische Projektion entsprechend größer sind. Bei der Verwendung einer parallelen bzw. orthogonalen Projektion würde es aufgrund der identischen Größen zu einer inkorrekten Verlagerung der vorderen Ebenen kommen, sollten diese mit einer höheren Geschwindigkeit verschoben werden. Die perspektivische Projektion wird dabei lediglich durch die Skalierung der jeweiligen Bilder simuliert, da es, im Gegensatz zur Technik zur Ebenendarstellung von Eggert und Sester (2013), kein echtes Kameramodell gibt, sondern lediglich die Lage der Bilder auf der x-Achse angepasst wird. Abbildung 8.14 veranschaulicht die beschriebene perspektivische, sowie die parallele Projektion mit den jeweiligen Zuordnungen von Vorder- und Hintergrundebenen.

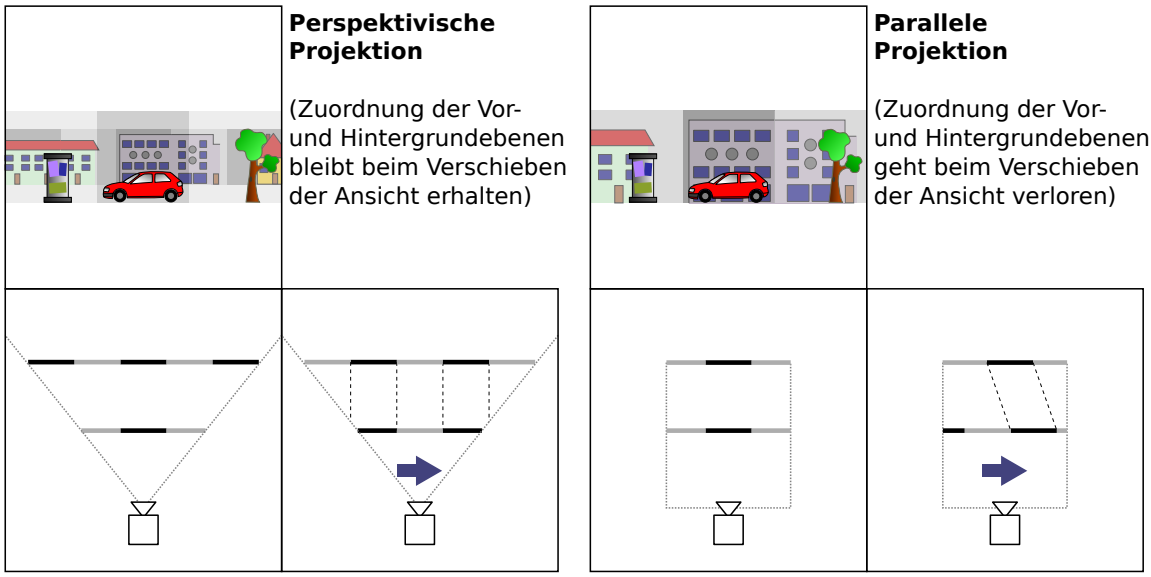


Abbildung 8.14: Gegenüberstellung von perspektivischer (links) und paralleler Projektion (rechts) mit Darstellung der Zuordnung von Vorder- und Hintergrundebenen als Draufsicht auf die Szene.

Die gezeigte Kombination aus simulierter perspektivischer Projektion und den damit einhergehenden unterschiedlichen Verschiebungsgeschwindigkeiten der einzelnen Ebenen ergibt die beabsichtigte Tiefenillusion. Dieser Effekt kann durch ein geeignetes Beleuchtungsmodell (vgl. Abschnitt 8.2.2) weiter verstärkt werden.

Die Nutzerinteraktion ist bei der gewählten Zielplattform entsprechend Touch-basiert. Die einzige Interaktionsmöglichkeit liegt in der horizontalen Verschiebung der Szene bzw. Kamera. Dies wurde durch simple Wischgesten umgesetzt. Um schnell durch die Szene zu navigieren, wurde die *fling* Geste umgesetzt (Google (2015)), welche die Verschiebung des Ausschnitts selbst nach dem Beenden der Geste fortsetzt.

Eggert und Schulze (2014) beleuchten weitere Aspekte der prototypischen Umsetzung der gezeigten Visualisierungstechnik.

8.2.2 Beleuchtungsmodell

Die Tiefenillusion wird vorrangig durch die unterschiedlichen Verschiebegeschwindigkeiten der einzelnen Ebenen erzeugt. Dieser Effekt lässt sich durch ein geeignetes Beleuchtungsmodell weiter verstärken. Ein Beleuchtungsmodell passt dabei die Helligkeit einzelner Pixel an. Die Berechnung der Anpassung basiert üblicherweise auf einer fiktiven Lichtquelle, sowie der Orientierung der Oberfläche bezüglich dieser Lichtquelle. Darüber hinaus können auch mehrere Lichtquellen und Materialeigenschaften in die Berechnung einfließen, worauf an dieser Stelle jedoch verzichtet wird.

Das genutzte Beleuchtungsmodell definiert eine Lichtquelle, welche sich rechts oben hinter der Kamera befindet. Da es sich bei der Darstellung um reine Bilder ohne jeglicher Geometrieinformation handelt, ist eine Berechnung der Helligkeitsanpassung nicht ohne weiteres möglich. Es wird die Orientierung der dargestellten Oberfläche benötigt. Grundsätzlich entspricht die Orientierung der Oberfläche der Ausrichtung der zugehörigen Bildebene. Jedoch wurden zur Bilderzeugung alle zugehörigen Scanpunkte auf diese Ebene projiziert. Dabei entspricht die Orientierung der Scanpunkte (also deren Punktnormale) nicht notwendigerweise der Orientierung der Ebene. Diese Abweichung kann in einer *NormalMap* festgehalten werden. Zur Speicherung kann ebenfalls ein Bild genutzt werden, wobei die Koordinatenachsen auf die Farbkanäle abgebildet werden (x=Rot, y=Grün und z=Blau). Die Ebenennormale zeigt dabei grundsätzlich nach $(0, 0, 1)$ und wird auf die Farbe $(128, 128, 255)$ ■ abgebildet. Normalen die nach links abweichen verringern den Rotanteil zu ■, wohingegen Abweichungen nach rechts selbigen zu ■ erhöhen. Analog dazu erhöhen Abweichungen nach oben den Grünanteil zu ■, wobei Abweichungen nach unten ihn wiederum zu ■ verringern.

Abbildung 8.15 zeigt ein Fassadenbild (links) mit der dazugehörigen NormalMap (rechts). Die jeweiligen Farbverschiebungen lassen die Ausrichtung der Oberfläche bereits gut erahnen. Der visuelle Eindruck bzw. Unterschied lässt sich über bloße Screenshots leider schlecht verdeutlichen, daher zeigt Abbildung 8.16 den Einfluss der NormalMap auf eine ansonsten weiße Fläche. Dabei ist die erzeugte Schattierung des Fensterrahmens deutlich sichtbar. Die Bildsequenz macht ebenfalls die Änderung der Schattierung, was den eigentlichen Tiefeneffekt erzeugt, deutlich. Ausgehend von der fiktiven Lichtquelle (oben rechts) ist im ersten Bild der Sequenz der linke Fensterrahmen komplett ausgeleuchtet und daher weiß, wohingegen der rechte Fensterrahmen einen deutlich sichtbaren Schatten wirft. Dieser Effekt verschiebt sich bei der Änderung des Betrachtungswinkels, bis beim letzten Bild der Sequenz sich der Fensterrahmen unmittelbar gegenüber der Lichtquelle befindet und somit beide Seiten gleichmäßig ausgeleuchtet werden und einen identischen Schatten werfen.

8.2.3 Ergebnis und Ausblick

Eine Analyse der Speichereffizienz der dargestellten Fassadenmodelle findet sich im Kapitel zur Modellgenerierung in Abschnitt 7.2.3.

Die prototypische Umsetzung der vorgestellten Visualisierungstechnik benötigt lediglich ein bis zwei Millisekunden für die Erstellung des gerenderten Bildes auf einem aktuellen Tabletgerät. Dies erlaubt Bildwiederholraten von mehr als 500 Bildern pro Sekunde. Somit ist die vorgestellte Visualisierung

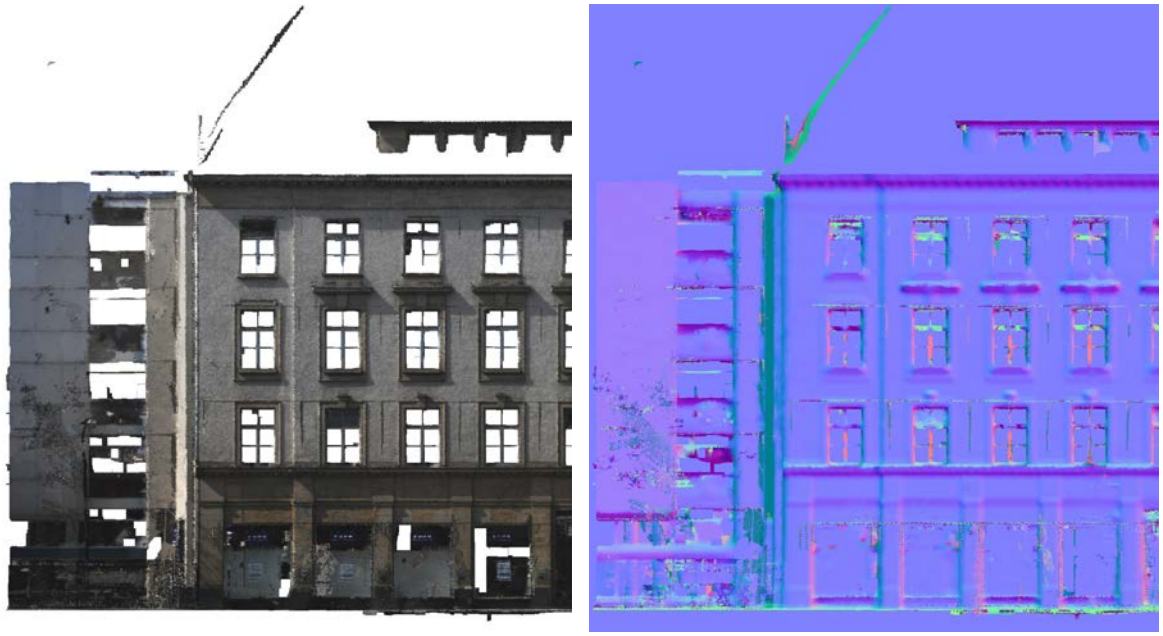


Abbildung 8.15: Textur einer Fassadenebene (links) und die dazugehörige NormalMap (rechts).

Abbildung 8.16: Bildsequenz zur Veranschaulichung des Beleuchtungsmodells basierend auf der NormalMap. Zur Verdeutlichung wurde der resultierende Helligkeitsunterschied auf einen weißen Teilbereich abgebildet.

selbst für sehr schwache Geräte geeignet. Zusammen mit der unterstützten Interaktionsmöglichkeit können Nutzer in wenigen Sekunden einen Überblick über lange Straßenzüge oder umfangreiche Datensätze erhalten. Das zu Beginn skizzierte Ekundungsszenario in einem mobilen Umfeld ist somit problemlos möglich. Eine geeignete Modellgenerierung vorausgesetzt, wären auch Echtzeitszenarien vorstellbar, wobei die, während einer Messfahrt, erfassten Daten unmittelbar auf die gezeigte Weise dargestellt werden. Hierfür müssten jedoch weitere Module des vorgestellten Frameworks für eine Echtzeit- oder Streamingverarbeitung angepasst werden.

Die erzielte Rendergeschwindigkeit zeugt von der hohen Effizienz des vorgestellten Visualisierungsansatzes. Umgekehrt stehen somit Prozessierungsressourcen zur Verfügung, um die Darstellung, bspw.

die Stärke der Tiefenillusion, weiter zu verbessern. Neben der Nutzung noch umfangreicherer und detaillierter Modelle können die besagten Prozessierungsressourcen auch für die Nutzung eines komplexeren Beleuchtungsmodells eingesetzt werden.

Das in Abschnitt 8.2.2 vorgestellte Beleuchtungsmodell für eine Parallax Scrolling Darstellung basierte auf der Berücksichtigung der Punktnormalen der ursprünglichen Scanpunkte für die Erzeugung eines realistischen Schattenwurfs. Analog zu den Punktnormalen könnten weitere Geometrieinformationen, welche durch die starke Vereinfachung der Daten verloren gegangen sind, in das Modell integriert werden. Diese können dann entweder ebenfalls im Rahmen des Beleuchtungsmodells berücksichtigt oder für weiterführende Effekte genutzt werden. Ein Beispiel für weitere Geometrieinformationen zeigt Abbildung 8.17 in Form von Distanzbildern. In diesen ist der Abstand der ursprünglichen Scanpunkte zur Fassadenebene farblich kodiert. Je nach benötigter Genauigkeit können ein oder mehrere Farbkanäle dafür genutzt werden. Diese Distanzinformation kann mittels Bumpmapping nach Blinn (1978) in das verwendete Beleuchtungsmodell einfließen. Darüber hinaus kann auch ein Displacement Mapping (Szirmay-Kalos und Umenhoffer (2008)) auf Basis der Distanzinformation erfolgen. Da ein Displacement Mapping jedoch die tatsächliche Geometrie der Ebene verändert, widerspräche dies ein wenig der Idee des Parallax Scrollings, welches rein zweidimensional ist. Darüber hinaus ist vorstellbar, dass die Geschwindigkeiten einzelner Pixel oder Regionen über die nun genaue Distanz zur Ebene angepasst werden, um die beabsichtigte Tiefenillusion weiter zu verstärken.

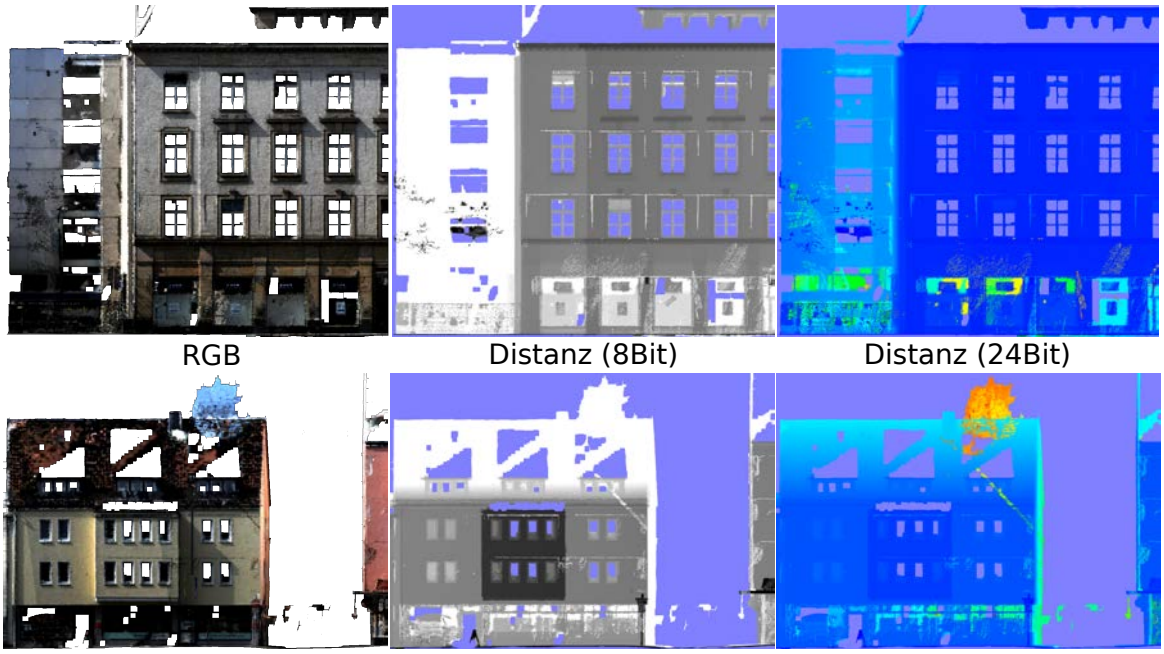


Abbildung 8.17: Distanz Bilder in 8Bit (Mitte) und 24Bit (rechts) anhand zweier Beispiele (obere und untere Reihe). Farbcodierung: Vor der Fassadenebene - Schwarz (8Bit) bzw. Blau (24Bit); Hinter der Fassadenebene - Weiß (8Bit) bzw. Rot (24Bit).

9 Schlussfolgerungen und Ausblick

Die eingangs gezeigte Zielsetzung dieser Arbeit umfasste vier Aspekte (vgl. Abschnitt 1.1):

1. Schaffung einer modularen Verarbeitungskette,
2. Effizienzbetrachtung, inklusive der Entwicklung und Umsetzung von hoch-parallelen Prozessierungsmodulen und Datentrukturen für effizienten Zugriff auf die Daten,
3. Erhöhung der Qualität der Sensordatenintegration am Beispiel der Kamera- und Laserscanner-Sensoren,
4. Entwicklung skalierbarer Visualisierungskonzepte

Das Ziel der Schaffung einer modularen Verarbeitungskette wurde durch das gezeigte modulare Framework zur Verarbeitung der Mobile Mapping Daten erreicht. Dafür wurden zunächst mögliche Anknüpfungspunkte bezüglich der proprietären Herstellerlösungen identifiziert. Ausgehend von den ermittelten Schnittstellen konnten eigene Module zur Verarbeitung der Daten entworfen und in dem in Abbildung 9.1 dargestellten Framework umgesetzt wurden. Als Austauschformat wird das offene PLY Dateiformat genutzt, welches die Erstellung weiterer eigener und die Nutzung existierender Module ermöglicht. Die entworfene Struktur, in Kombination mit geeigneten Schnittstellen, garantiert dabei eine problemlose Erweiterbarkeit und eine flexible Wiederverwendbarkeit einzelner Komponenten bzw. Module. In diesem Rahmen wurden diverse Verarbeitsungskomponenten und -module zur Farbextraktion und -anpassung, der Bestimmung von Punktmerkmalen wie Normalen, sowie Segmentierungs- und Klassifizierungsverfahren prototypisch umgesetzt.

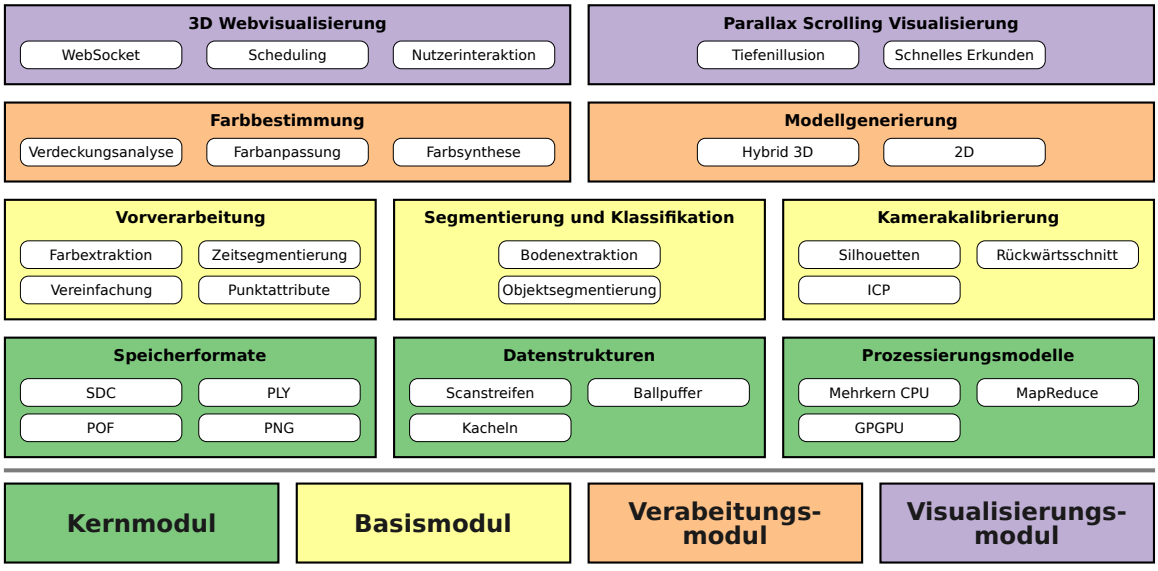


Abbildung 9.1: Übersicht über Module und Komponenten des geschaffen Frameworks.

Der zweite Aspekt der Arbeit beschäftigte sich mit der Effizienzbetrachtung der Verarbeitungsmodule. Da viele Verfahren jeweils einzelne Punkte unabhängig verarbeiten, wurden Konzepte zur Parallelisierung der Verarbeitung untersucht und umgesetzt. Als konkrete Umsetzung wurde ein Verfahren zu Bestimmung von Punktnormalen mittels dreier Parallelisierungtechniken (Mehrkern-CPU, GPGPU und Rechencluster/Hadoop) implementiert und analysiert. Die Resultate zeigten signifikante Laufzeitvorteile bei der Mehrkern-CPU Umsetzung, wohingegen die GPGPU und Hadoop Implementierungen deutlich schlechter abschnitten. Im Falle der GPGPU Umsetzung wurden über

80% der Laufzeit für Datenübertragungsoperationen von und zum Grafikartenspeicher aufgewendet, was durch eine Anpassung des Verfahrens, oder die Nutzung modernerer Hardware reduziert werden könnte. Die schlechte Laufzeit der Hadoopumsetzung liegt unter anderem am verhältnismäßig hohen Overhead, welcher mit der Nutzung des Hadoop Frameworks einhergeht. Sind im ausführenden Rechencluster nur geringe Kapazitäten (bspw. eine geringe Anzahl an Knoten, oder leistungsschwache Knoten) vorhanden, so fällt, wie im vorliegenden Fall, der zusätzliche Overhead so stark ins Gewicht, dass eine Laufzeitverbesserung nicht erzielt werden kann. Obgleich keine Laufzeitverbesserung gezeigt werden konnte, hat die Nutzung von Rechenclustern, bzw. BigData-Technologien im allgemeinen, weitere nicht zu vernachlässigende Vorteile. Hervorzuheben ist hier die inherente Skalierbarkeit. Durch das bloße Hinzufügen weiterer Knoten kann die Rechenleistung und darüber hinaus die Speicherkapazität des Clusters weiter gesteigert werden. Eine Anpassung der Implementierung ist dabei nicht notwendig. In gewissen Grenzen ist dies auch bei GPGPU Systemen möglich. Dabei kann ein und derselbe Rechner mit mehreren Grafikkarten ausgerüstet werden, um die Rechen- und Speicherkapazitäten zu erhöhen. Sind bei einem Rechencluster problemlos mehrere Hundert Knoten möglich, ist die Anzahl der möglichen Grafikkarten pro System in der Regel technisch auf einstellige Werte beschränkt. Eine solche Skalierbarkeit ist bei einer Mehrkern-CPU Umsetzung üblicherweise nicht gegeben. Neben der Parallelisierung der Verfahren wurden geeignete Datenstrukturen sowie Zugriffs- und Pufferstrategien für Mobile Mapping Daten untersucht und entwickelt. Für eine sequentielle Verarbeitung der Punktdaten wurde eine Scanstreifenbasierte Datenstruktur mit einer für typische Zugriffsmuster optimierten Pufferstrategie entwickelt und umgesetzt, sowie die jeweiligen Vorteile diskutiert. Für Verfahren, welche einen wahlfreien Zugriff auf die Daten voraussetzen, wurde eine einfache rasterbasierte Datenstruktur genutzt, welche je nach Genauigkeitsanforderungen die Speichereffizienz, durch die Verwendung geringerer Wortbreiten, erhöht. Zugriffe auf einzelne Rasterzellen erfolgen dabei über einen Cache. Darüber hinaus konnte gezeigt werden, wie geeignete Zugriffs- und Verdrängungstrategien eine signifikante Effizienzsteigerung (bspw. Senkung der Cache-Miss-Rate auf 31%) ermöglichen.

Die dritte Zielsetzung war die Qualitätssteigerung der Sensordatenintegration. Hier wurde ein Verfahren zur Feinkalibrierung der Kamerapositionen und -orientierungen konzipiert und umgesetzt. Das realisierte Verfahren beruht dabei auf dem automatischen Finden von korrespondierenden Scan- und Bildpunkten und einem anschließenden Rückwärtsschnitt zur Ermittlung der genauen Kameraposition und -orientierung. Dafür wurden zunächst, mittels initialer Kameraparameter, die erfassten Scanpunkte in Scanpunktbilder transformiert. Anschließend wurden sowohl aus den Kamerabildern, als auch aus den erzeugten Scanpunktbildern Silhouetten extrahiert. Zur Ermittlung der Bild- und Scanpunktkorrespondenzen wurden die extrahierten Silhouetten mittels des Iterative-Closest-Point (ICP) Verfahrens zur Deckung gebracht. Ein entscheidender Aspekt für die Genauigkeit war die Modellierung eines Zeitversatzes zwischen Auslösen der Bildaufnahme und des tatsächlichen Bildaufnahmezeitpunkts. Das Ergebnis ist eine deutliche Steigerung der Genauigkeit bei der Integration der Kamera- und Laserscannersensoren. Hier konnte an mehreren Beispieldatensätzen eine signifikante Reduktion der Residuen des Rückwärtsschnitts gezeigt werden. Die mittels des vorgestellten Verfahrens bestimmten präziseren Kameraparameter erlauben eine genauere Ermittlung der Lage der erfassten Scanpunkte in den jeweiligen Kamerabildern. Nur eine möglichst genaue Lage des Scanpunktes im Kamerabild stellt sicher, dass der jeweilige Farbwert im Kamerabild auch zum jeweiligen Scanpunkt gehört. Daher erhöhen genauere Kameraparameter die Qualität der Farbzuordnung der Scanpunkte, was wiederum die visuelle Qualität der abgeleiteten Modelle erhöht.

Die Entwicklung und Umsetzung skalierbarer Visualisierungskonzepte für die erfassten Mobile Mapping Daten stellte das abschließende Ziel dar. Ausgehend von unterschiedlichen Szenarien wurden zwei Visualisierungsansätze umgesetzt. Beide Visualisierungen erfordern die Erstellung von speziellen Punktwolkenmodellen. Die Module zur Modellgenerierung wurden mittels der Kern- und Basismodule des geschaffenen Frameworks umgesetzt. Da es sich um Modelle zur Visualisierung handelt, wurde besonderes Augenmerk auf eine korrekte Einfärbung der Punkte unter Berücksichtigung von Verdeckungen und dem Ausgleich von ungewollten Farbübergängen gelegt. Die entwickelte geometrische Verdeckungsanalyse, in Kombination mit der dafür angepassten Ballpufferdatenstruktur, weist dabei deutliche Vorteile, in Bezug auf Flexibilität, Genauigkeit und Verarbeitungseffizienz, gegenüber alternativen Verfahren wie der semantischen Analyse (vgl. Abschnitt 2.5) auf. Darüber hinaus fanden die

vorgestellten Konzepte zur Steigerung der Verarbeitungseffizienz in Form von Parallelisierung und der Nutzung von effizienten Datenstrukturen und Zugriffsstrategien bei der Umsetzung der Modellgenerierungsmodule Anwendung. Die gezeigte webbasierte 3D Visualisierung erlaubt die Darstellung nahezu beliebig umfangreicher Mobile Mapping Daten in hybrider Repräsentation, bestehend aus eingefärbten Scanpunkten und texturierten Flächen, in einem gewöhnlichen Internetbrowser. Kern der Skalierbarkeit der Visualisierung bildet eine, für die gewählte Umgebung entworfene, kontinuierliche Level-Of-Detail Technik. Sie nutzt zum einen eine geschickte Partitionierung der Modelldaten und zum anderen ein blickwinkelabhängiges Scheduling der Datenanfragen. Bestehende LOD Techniken beschränken sich in der Regel auf einen Modelltyp, Punkte oder Polygone, wohingegen das entwickelte Verfahren hybride Modelle bestehend aus Punkten und texturierten Flächen unterstützt. Neben der bloßen Darstellung wurden auch Konzepte zur Navigation und Interaktion mit den Punktwolkendaten aufgezeigt und umgesetzt. Eine realisierte Interaktion ermöglicht die Bestimmung von beliebigen Strecken in der dargestellten Szene. Potentielle Nutzer dieser Interaktionsmöglichkeit könnten Behörden sein, aber auch Transportdienstleister (Schwerlasttransport) oder Umzugsunternehmen, also alle, welche auf möglichst exakte Rauminformationen angewiesen sind. Im Gegensatz zu etablierten Systemen wie Google Streetview erlaubt die entwickelte 3D Visualisierung eine detaillierte und uneingeschränkt begehbare Abbildung urbaner Räume. Der zweite gezeigte Visualisierungsansatz beruht auf der Parallax Scrolling Technik. Diese stark komprimierte Form der Darstellung in Kombination mit einer intuitiven gestenbasierten Steuerung ermöglicht das schnelle Erkunden von umfangreichen Mobile Mapping Daten. Mittels des implementierten Prototyps konnte gezeigt werden, dass diese Darstellungsform, selbst auf mobilen Endgeräten mit geringer Leistung, die Visualisierung und Exploration umfangreicher Punktwolken in Form von Fassadenmodellen erlaubt. Neben dem geringen Ressourcenbedarf weist diese Visualisierung deutliche Vorteile gegenüber alternativen Ansätzen, wie beispielsweise Streetslide, auf. Zum einen lässt die erzeugte Tiefenillusion die Darstellung wesentlich realistischer wirken, zum anderen entstehen keine Probleme mit weiter entfernten Objekten.

9.1 Ausblick

Das im Rahmen dieser Arbeit entworfene modulare Framework hat sich bereits bei der Umsetzung der eigenen Verfahren bewährt. Die logische Unterteilung in vier aufeinander aufbauende Schichten, sowie in einzelne Module, welche wiederum mehrere Komponenten umfassen, garantiert eine leichte Wiederverwendbarkeit. Somit lassen sich problemlos neue Verarbeitungsketten zur Beantwortung beliebiger Fragestellungen definieren und mit minimalem Aufwand umsetzen. Im Rahmen zukünftiger Bemühungen sollte das vorhandene Framework um weitere Module und Komponenten erweitert werden, um die Verwendbarkeit für die Erforschung aktueller oder zukünftiger Fragestellungen mittels neuer Verfahren und Analysen im wissenschaftlichen Kontext weiter zu erhöhen.

Die Resultate der Feinkalibrierung der Kameraposition und -orientierung sind ebenfalls sehr vielversprechend. Verbesserungspotential besteht hier bei der automatischen Auswahl der Featurepunkte. Bei der vorliegenden Messanordnung und den daraus resultierenden Kamerabildern befinden sich die extrahierten Silhouetten vorwiegend in der oberen Bildhälfte. Dies stellt jedoch keine optimale geometrische Ausgangslage für den durchgeführten Rückwärtsschnitt dar. Um korrespondierende Punkte aus allen Bildbereichen, insbesondere der unteren Bildhälfte, zu erhalten, könnten weitere Features wie Kanten oder Ecken herangezogen werden. Diese müssten dann sowohl in den Kamerabildern, als auch in den Scanpunktdaten identifiziert und darüber hinaus einander zugeordnet werden. Dies würde die Genauigkeit und Stabilität der Rückwärtsschnittberechung weiter verbessern. Wie in Abschnitt 5.7 diskutiert, lässt sich die Laufzeit des vorgestellten Verfahrens signifikant reduzieren, indem nicht alle Bilder und Silhouetten zur Bestimmung der Kameraparameter herangezogen werden. Eine gezielte Selektion relevanter Bilder auf Basis gewisser Randbedingungen, wie der Geschwindigkeit des Messfahrzeugs, kann zu ähnlich robusten Ergebnissen führen, wobei die Menge an zu verarbeitenden Daten drastisch reduziert würde. Weiteres Verbesserungspotential liegt in der Aufhebung der strikten Trennung zwischen der Ermittlung der Korrespondenzen und der finalen Berechnung der gesuchten Parameter. Hier kann eine erneute Ermittlung der Korrespondenzen mittels temporärer Kameraparameter die Qualität der finalen Parameter weiter steigern.

Die gezeigten Konzepte zur Parallelisierung bergen ebenfalls weiteres Potential. Obgleich sich im Rahmen dieser Arbeit die Mehrkern-CPU Variante als Mittel der Wahl herausgestellt hat, ist eine bessere Skalierbarkeit fast ausschließlich bei der Verwendung eines Rechenclusters gegeben. Hier gilt es den gezeigten Ansatz weiter zu verbessern und die erhoffte Skalierbarkeit bei noch umfangreicheren Daten zu zeigen. Hierfür ist jedoch eine vollständige Anpassung aller Module und Komponenten auf die beabsichtigte BigData-Umgebung (wie bspw. Hadoop) erforderlich. Im Rahmen dieser Arbeit wurde lediglich die Vorverarbeitungskomponente zur Bestimmung der Punktnormalen für eine Hadoopumgebung umgesetzt. Die dafür notwendigen Maßnahmen, wie die Spezifikation eines Avro-Schemas bei der Verarbeitung von Binärdaten, machen dieses Unterfangen jedoch verhältnismäßig aufwendig. Im Gegenzug erlaubt eine entsprechende Umsetzung die parallele Verarbeitung der Mobile Mapping Daten über hunderte von Rechenknoten hinweg, was einen signifikanten Laufzeitvorteil bieten würde.

Der entworfene Ansatz zur Verdeckungsanalyse weist, wie gezeigt, einige Unzulänglichkeiten auf. Hier werden lediglich Verdeckungen berücksichtigt, die auch vom Laserscanner erfasst wurden, was zu Problemen bei komplexeren Verdeckungssituationen aber auch bei Glas- und Spiegelflächen führt, da diese nur eingeschränkt durch den Laserscanner erfasst werden. Besonders problematisch sind dabei dynamische Verdecker, bspw. andere, sich bewegende Verkehrsteilnehmer. Eine korrekte Berücksichtigung dynamischer Verdecker erfordert die Adressierung gleich mehrerer Herausforderungen. Zunächst müssen dynamische Verdecker als solche identifiziert werden. Dies könnte über einen Vergleich von segmentierten Objekten aus den Scanpunkten jeweils eines Laserscanners erfolgen. Wurde ein dynamischer Verdecker erkannt, muss dessen Position zum Zeitpunkt der Bildaufnahme bestimmt werden. Dies könnte durch eine Interpolation zwischen den zwei zuvor ermittelten Objektsegmenten erfolgen. Abschließend müssen die erfassten Punkte des dynamischen Verdeckers von den ursprünglichen Positionen der Erfassungszeitpunkte, an die Stelle des relevanten Aufnahmezeitpunktes verschoben werden.

Die umgesetzten Visualisierungstechniken können bezüglich der benötigten Modellgenerierung erweitert werden. Hier ist vorstellbar, dass die Modelle von dynamischen Objekten, wie Fahrzeugen, Passanten, usw. befreit werden. Findet eine mehrmalige Erfassung desselben Gebietes statt, so könnten alle Objekte entfernt werden, welche nicht in der Mehrheit der Aufnahmen zu finden sind. Dies würde auch etwaige Datenschutzbedenken mildern. Des Weiteren könnten so auch potentiell vorhandene Lücken, bspw. durch Verdeckungen verursacht, aus mehreren Aufzeichnungen rekonstruiert werden. Abschließend würde dies auch ein Monitoring bzw. die Erfassung von Änderungen des aufgezeichneten Gebietes ermöglichen, was jedoch eine eigenständige Fragestellung darstellt und über die Erzeugung von Punktwolkenmodellen und deren Visualisierung hinaus geht. Die vorgestellte kontinuierliche LOD Technik ließe sich ebenfalls erweitern. Zum einen könnte die Blickwinkelabhängigkeit auf Teile der texturierten Fläche ausgeweitet und somit die Granularität des Detailgrades weiter verfeinert werden. Darüber hinaus ließe sich diese feine Granularität auch auf die Punktdaten ausweiten, so dass die kleinste betrachtete Einheit ein einzelner Punkt darstellt. Zur Zeit sind dies, aus Effizienzgründen, noch kleinere Punktmengen. Das Herunterbrechen auf einzelne Punkte erfordert jedoch andere Abfrage- und Übertragungsstrategien, als sie im Rahmen dieser Arbeit vorgestellt wurden. Da die erfassten urbanen Gebiete typischerweise eher flach sind (die Ausdehung in z-Richtung beträgt lediglich wenige Meter, wohingegen die Ausdehnung in xy-Richtung mehrere Kilometer betragen kann), wird bei der Auswertung des Sichtkegels die Höhe aus Effizienzgründen ignoriert. Je nach darzustellendem Modell und verwendeter Navigationtechnik könnte dies zur Übertragung und Darstellung unnötiger Modelldaten führen. Für eine Darstellung nicht flacher Modelldaten müsste die LOD Datenstruktur, als auch das Abfrageschema entsprechend angepasst werden. Die implementierten Navigationstechniken sind ebenfalls eher rudimentärer Natur und könnten für weitere Nutzergruppen entsprechend erweitert werden. Beispielsweise könnte die Beschränkung der Position virtuellen Kamera auf die Trajektorie des Messfahrzeugs aufgehoben oder zumindest aufgeweicht werden. Dies erfordert jedoch weiterführende Navigationseingaben durch den Nutzer, was unbedarfte Nutzer unter Umständen überfordern könnte. Bezüglich der realisierbaren Nutzerinteraktionen wurden lediglich die Möglichkeiten aufgezeigt und anhand eines Beispiels (Messen von Entfernungen innerhalb der dargestellten Punktwolke) prototypisch umgesetzt. Hier sind je nach Anwendungsszenario vielfältige Realisierungen denkbar. Neben der Erweiterung der vorgestell-

ten Visualisierungskonzepte sind weitere Möglichkeiten vorstellbar. Ein interessanter Ansatz stellt hier bspw. die Kombination der erfassten Mobile Mapping Daten mit einem Virtuellen Globus, wie Google Earth, dar. Dies erlaubt unter Umständen eine noch immersivere Darstellung und sorgt für mehr Kontext durch die zusätzliche Darstellung der Satellitenbilder.

Literaturverzeichnis

Alsabti, K., Ranka, S., Singh, V., 1997. An efficient k-means clustering algorithm. *Electrical Engineering and Computer Science*.

Baarda, W., 1968. A testing procedure for use in geodetic networks. Nr. 5 In Bd. 2. Netherlands Geodetic Commission, Delft.

Bay, H., Tuytelaars, T., Van Gool, L., 2006. Surf: Speeded up robust features. *Computer vision–ECCV 2006*, S. 404–417.

Belton, D., Lichti, D. D., 2006. Classification and segmentation of terrestrial laser scanner point clouds using local variance information. *IAPRS, XXXVI* 5.

Bernardini, F., Mittleman, J., Rushmeier, H., Silva, C., Taubin, G., 1999. The Ball-Pivoting Algorithm for Surface Reconstruction. *IEEE Transactions on Visualization and Computer Graphics* 5 (4), S. 349–359.

Blinn, J. F., 1978. Simulation of Wrinkled Surfaces. *SIGGRAPH Comput. Graph.* 12 (3), S. 286–292.

Bock, F., Eggert, D., Sester, M., 2015. On-street parking statistics using lidar mobile mapping. In: *Intelligent Transportation Systems (ITSC), 2015 IEEE 18th International Conference on*. IEEE, S. 2812–2818.

Borrmann, D., Elseberg, J., Lingemann, K., Nüchter, A., 2011. The 3D Hough Transform for Plane Detection in Point Clouds: A Review and a New Accumulator Design. *3D Res.* 2 (2), S. 32:1–32:13.

Bouvier, D. J., Gordon, C., McDonald, M., 2011. An Approach for Occlusion Detection in Construction Site Point Cloud Data. In: *Computing in Civil Engineering: Proceedings of the 2011 Asce International Workshop on Comupting in Civil Engineering*. Miami, Florida, S. 234–241.

Carey, R., Bell, G., Marrin, C., 1997. ISO/IEC 14772-1: 1997 VirtualReality Modeling Language (VRML97). Tech. rep.

CCITT, 1992. Recommendation T.81 - JPEG Specification.

Chen, B., Nguyen, M. X., 2001. POP: A hybrid point and polygon rendering system for large data. In: *Proceedings of the conference on Visualization'01*. IEEE Computer Society, S. 45–52.

Chum, O., Matas, J., Obdržálek, Š., 2004. Enhancing RANSAC by Generalized Model Optimization. In: Hong, K.-S., Zhang, Z. (Hrsg.), *Proc. of the Asian Conference on Computer Vision (ACCV)*. Bd. 2. Asian Federation of Computer Vision Societies, Seoul, Korea South, S. 812–817.

Clark, J. H., 1976. Hierarchical geometric models for visible surface algorithms. *Communications of the ACM* 19 (10), S. 547–554.

Csuri, C., Hackathorn, R., Parent, R., Carlson, W., Howard, M., 1979. Towards an Interactive High Visual Complexity Animation System. In: *Proceedings of the 6th Annual Conference on Computer Graphics and Interactive Techniques*. SIGGRAPH '79. ACM, New York, NY, USA, S. 289–299.

Dean, J., Ghemawat, S., 2008. MapReduce: simplified data processing on large clusters. *Communications of the ACM* 51 (1), S. 107–113.

Decoret, X., Durand, F., Sillion, F. X., Dorsey, J., 2003. Billboard Clouds for Extreme Model Simplification. *ACM Transactions on Graphics* 22, S. 689–696.

Demantke, J., Mallet, C., David, N., Vallet, B., 2011. DIMENSIONALITY BASED SCALE SELECTION IN 3D LIDAR POINT CLOUDS. In: *ISPRS Archives - ISPRS Workshop Laser Scanning*. Bd. XXXVIII-5/W12. S. 0 – 0.

Deutsch, L. P., 1996. DEFLATE Compressed Data Format Specification version 1.3. Internet RFC 1951.

Disney, 1937. Multiplane Camera. http://www.waltdisney.org/content/multiplane-camera.

Douglas, D. H., Peucker, T. K., 1973. Algorithms for the reduction of the number of points required to represent a digitized line or its caricature. *Cartographica: The International Journal for Geographic Information and Geovisualization* 10 (2), S. 112–122.

Edelsbrunner, H., Kirkpatrick, D., Seidel, R., 1983. On the shape of a set of points in the plane. *IEEE Transactions on information theory* 29 (4), S. 551–559.

Eggert, D., Dalyot, S., 2012. Octree-based SIMD strategy for ICP registration and alignment of 3D point clouds. *ISPRS Ann. Photogramm. Remote Sens. Spat. Inf. Sci* 3, S. 105–110.

Eggert, D., Hücker, D., Paelke, V., 2014. Augmented reality visualization of archeological data. In: *Cartography from Pole to Pole*. Springer, S. 203–216.

Eggert, D., Paelke, V., 2010. Relevance-driven acquisition and rapid on-site analysis of 3d geospatial data. In: *International Archives of the Photogrammetry, Remote Sensing and Spatial Information Sciences:[Joint International Conference On Theory, Data Handling And Modelling In Geospatial Information Science] 38 (2010), Nr. Part 2*. Bd. 38. Göttingen: Copernicus GmbH, S. 118–123.

Eggert, D., Schulze, E. C., 2014. Visualization of Mobile Mapping Data via Parallax Scrolling. *The International Archives of Photogrammetry, Remote Sensing and Spatial Information Sciences* 40 (3), S. 89.

Eggert, D., Sester, M., 2013. Multi-layer visualization of mobile mapping data. *ISPRS Annals of Photogrammetry, Remote Sensing and Spatial Information Sciences* (2), S. 73–78.

El-Hakim, S. F., Brenner, C., Roth, G., 1998. A multi-sensor approach to creating accurate virtual environments. *ISPRS Journal of Photogrammetry and Remote Sensing* 53 (6), S. 379–391.

Elmqvist, N., Tsigas, P., 2008. A taxonomy of 3d occlusion management for visualization. *Visualization and Computer Graphics, IEEE Transactions on* 14 (5), S. 1095–1109.

Fairchild, M. D., 2013. Color appearance models. John Wiley & Sons.

Fette, I., Melnikov, A., 2011. The WebSocket Protocol. Internet RFC 6455.

Fischler, M. A., Bolles, R. C., 1981. Random Sample Consensus: A Paradigm for Model Fitting with Applications to Image Analysis and Automated Cartography. *Commun. ACM* 24 (6), S. 381–395.

Freeman, H., 1961. On the Encoding of Arbitrary Geometric Configurations. *Electronic Computers, IRE Transactions on* EC-10 (2), S. 260–268.

G. Randers-Pehrson, e. a., 1999. PNG (Portable Network Graphics) Specification, Version 1.2.

Garg, R. P., Sharapov, I. A., Sharapov, I., 2002. Techniques for optimizing applications: high performance computing. Sun Microsystems Press.

Google, 2015. Mobile Input Patterns - Drag, Swipe and Fling Gestures. https://www.google.com/design/spec/patterns/gestures.html.

Gross, H., Jutzi, B., Thoennessen, U., 2007. Segmentation of tree regions using data of a full-waveform laser. *International Archives of Photogrammetry, Remote Sensing and Spatial Information Sciences* 36 (part 3), S. W49A.

Grossman, J., Dally, W., 1998. Point Sample Rendering. In: Drettakis, G., Max, N. (Hrsg.), *Rendering Techniques '98*. Eurographics. Springer Vienna, S. 181–192.

Hachet, M., Declec, F., Knodel, S., Guitton, P., 2008. Navidget for Easy 3D Camera Positioning from 2D Inputs. In: *3D User Interfaces, 2008. 3DUI 2008. IEEE Symposium on*. S. 83–89.

Hammoudi, K., Dornaika, F., Soheilian, B., Vallet, B., McDonald, J., Paparoditis, N., 2012. Recovering Occlusion-Free Textured 3D Maps of Urban Facades by a Synergistic Use of Terrestrial Images, 3D Point Clouds and Area-Based Information. *Procedia Engineering* 41 (0), S. 971 – 980, international Symposium on Robotics and Intelligent Sensors 2012 (IRIS 2012).

Hammoudi, K., Dornaika, F., Soheilian, B., Vallet, B., McDonald, J., Paparoditis, N., 2013. A Synergistic Approach for Recovering Occlusion-Free Textured 3D Maps of Urban Facades from Heterogeneous Cartographic Data. *CoRR* abs/1308.6401.

Hickson, I., 2012. Web Workers - W3C Candidate Recommendation. http://www.w3.org/TR/workers/.

IGI, 2014. IGI mbH und 3D Laser Mapping Ltd., StreetMapper 360 Datasheet. Tech. rep.

ISTI-CNR, 2014. http://meshlab.sourceforge.net/. http://meshlab.sourceforge.net/.

Jackson, D., Gilbert, J., 2015. WebGL Specification.

Jenks, G. F., Caspall, F. C., 1971. Error on Choroplethic Maps. Definition, Measurement, Reduction. *Annals of the Association of American Geographers*, S. 217–244.

Kazhdan, M., Bolitho, M., Hoppe, H., 2006. Poisson surface reconstruction. In: *Proceedings of the fourth Eurographics symposium on Geometry processing*. Bd. 7.

Kopf, J., Chen, B., Szeliski, R., Cohen, M., 2010. Street Slide: Browsing Street Level Imagery. In: *ACM SIGGRAPH 2010 Papers*. SIGGRAPH '10. ACM, New York, NY, USA, S. 96:1–96:8.

Korf, R. E., 2003. Optimal Rectangle Packing: Initial Results. In: *ICAPS*. S. 287–295.

Korte, B., Vygen, J., 2008. Kombinatorische Optimierung. Springer.

Kraus, K., Jansa, J., 1996. Photogrammetrie: Verfeinerte Methoden und Anwendungen. De Gruyter Lehrbuch Series. Dümmler.

Leica, 2014. Leica Geosystems, Datasheet Leica Pegasus: Two. Tech. rep.

Levoy, M., Whitted, T., 1985. The Use of Points as a Display Primitive. Tech. Rep. 85-022, Computer Science Department, University of North Carolina.

Lipchak, B., 2014. OpenGL ES Version 3.0.4.

Lorensen, W. E., Cline, H. E., 1987. Marching Cubes: A High Resolution 3D Surface Construction Algorithm. *SIGGRAPH Comput. Graph.* 21 (4), S. 163–169.

Lowe, D. G., 2004. Distinctive image features from scale-invariant keypoints. *International journal of computer vision* 60 (2), S. 91–110.

Luebke, D., Reddy, M., Cohen, J. D., Varshney, A., Watson, B., Huebner, R., 2002. Level of Detail for 3D Graphics. Elsevier.

Luhmann, T., 2000. Nahbereichsphotogrammetrie. Wichmann Verlag.

Maciel, P. W., Shirley, P., 1995. Visual navigation of large environments using textured clusters. In: *Proceedings of the 1995 symposium on Interactive 3D graphics*. ACM, S. 95–ff.

MacQueen, J., 1967. Some methods for classification and analysis of multivariate observations. In: *Proceedings of the Fifth Berkeley Symposium on Mathematical Statistics and Probability, Volume 1: Statistics*. University of California Press, Berkeley, Calif., S. 281–297.

Mazzoleni, A., 2014. Advance Projects - AdvanceCOMP: AdvPNG. http://advancemame.sourceforge.net/.

Microsoft, 2016. Bing Maps Streetside. https://www.microsoft.com/maps/streetside.aspx.

Mine, M., 1996. Working in a virtual world: Interaction techniques used in the chapel hill immersive modeling program. *University of North Carolina*.

Monnier, F., Vallet, B., Soheilian, B., 2012. Trees detection from laser point clouds acquired in dense urban areas by a mobile mapping system. *Proceedings of the ISPRS Annals of the Photogrammetry, Remote Sensing and Spatial Information Sciences (ISPRS Annals), Melbourne, Australia* 25, S. 245–250.

Muerle, J. L., Allen, D. C., 1968. Experimental Evaluation of Techniques for Automatic Segmentation of Objects in a Complex Scene. In: Cheng, G. C., Ledley, R. S., Pollock, D. K., ROSENFELD, A. (Hrsg.), *Pictorial Pattern Recognition*. Pergamon Press, UK, S. 3–13.

Munshi, A., Leech, J., 2010. OpenGL ES - Common Profile Specification 2.0.25 (Full Specification).

Nguyen, Q.-D., Bredif, M., Richard, D., Paparoditis, N., 2016. Progressive streaming and massive rendering of 3D city models on web-based virtual globe. In: *Proceedings of 24th ACM SIGSPATIAL International Conference on Advances in Geographic Information Systems*. San Francisco, USA.

Nikon, 2008. Nikon Corp., Datasheet D700. Tech. rep.

Nüchter, A., Gutev, S., Borrmann, D., Elseberg, J., 2011. Skyline-based registration of 3D laser scans. *Geospatial Information Science* 14 (2), S. 85–90.

Open-Geospatial-Consortium, 2012. OGC12-019: OGC City Geography Markup Language (CityGML) Encoding Standard. Tech. rep.

Open-GIS-Consortium-Inc., 2002. OpenGIS Geography Markup Language (GML) Implementation Specification V2.1.2. Tech. rep.

Opheim, H., 1981. Smoothing a digitized curve by data reduction methods. *Eurographics '81*, S. 127—-135.

Opheim, H., 1982. Fast data reduction of a digitized curve. *GeoProcessing* 2, S. 33–40.

Otepka, J., Ghuffar, S., Waldhauser, C., Hochreiter, R., Pfeifer, N., 2013. Georeferenced Point Clouds: A Survey of Features and Point Cloud Management. *ISPRS International Journal of Geo-Information* 2 (4), S. 1038.

Pearson, K., 1901. On Lines and Planes of Closest Fit to Systems of Points in Space. *Phil. Mag.* 6 (2), S. 559–572.

Perdeck, M., 2011. Fast Optimizing Rectangle Packing Algorithm for Building CSS Sprites. http://www.codeproject.com/Articles/210979/Fast-optimizing-rectangle-packing-algorithm-for-bu.

Reumann, K., Witkam, A. P. M., 1974. Optimizing curve segmentation in computer graphics. *Proceedings of International Computing Symposium*, S. 467—-472.

Richter, R., Döllner, J., 2010. Out-of-core real-time visualization of massive 3D point clouds. In: *Proceedings of the 7th International Conference on Computer Graphics, Virtual Reality, Visualisation and Interaction in Africa*. ACM, S. 121–128.

Riegl, 2012. Riegl Laser Measurement Systems GmbH, Datasheet RIEGL VQ-250. Tech. rep.

Riegl, 2015. Riegl Laser Measurement Systems GmbH, VMX-450 Datasheet. Tech. rep.

Rosenfeld, A., Pfaltz, J. L., 1966. Sequential Operations in Digital Picture Processing. *J. ACM* 13 (4), S. 471–494.

Rublee, E., Rabaud, V., Konolige, K., Bradski, G., 2011. ORB: An efficient alternative to SIFT or SURF. In: *Computer Vision (ICCV), 2011 IEEE international conference on*. IEEE, S. 2564–2571.

Rusinkiewicz, S., Levoy, M., 2000. QSplat: A Multiresolution Point Rendering System for Large Meshes. In: *Proceedings of the 27th Annual Conference on Computer Graphics and Interactive Techniques*. SIGGRAPH '00. ACM Press/Addison-Wesley Publishing Co., New York, NY, USA, S. 343–352.

Sainz, M., Pajarola, R., 2004. Point-based rendering techniques. *Computers and Graphics* 28 (6), S. 869 – 879.

Shi, W., Cheung, C., 2006. Performance evaluation of line simplification algorithms for vector generalization. *The Cartographic Journal* 43 (1), S. 27–44.

Silverman, K., 2015. Ken Silvermans's Utility Page: PNGOUT. http://advsys.net/ken/utils.htm#pngout.

Ström, J., 2008. OpenGL ES Extension #5 - OES compressed ETC1 RGB8 texture.

Ström, J., Akenine-Möller, T., 2005. i PACKMAN: High-quality, low-complexity texture compression for mobile phones. In: *Proceedings of the ACM SIGGRAPH/EUROGRAPHICS conference on Graphics hardware*. ACM, S. 63–70.

Szirmay-Kalos, L., Umenhoffer, T., 2008. Displacement Mapping on the GPU—State of the Art. *Computer graphics forum* 27 (6), S. 1567–1592.

Tanenbaum, A. S., Van Steen, M., 2002. Distributed systems: principles and paradigms. Bd. 2. Prentice hall Englewood Cliffs.

Tang, R., Halim, S., Zulkepli, M., 2013. Surface Reconstruction Algorithms: Review and Comparison. *8th International Symposium on Digital Earth*.

Tao, C. V., Li, J., 2007. Advances in mobile mapping technology. CRC Press.

Tarjan, R. E., 1979. A class of algorithms which require nonlinear time to maintain disjoint sets. *Journal of Computer and System Sciences* 18 (2), S. 110 – 127.

Trimble, 2013. Trimble MX8, Datasheet. Tech. rep.

Turk, G., 1994. The PLY Polygon File Format Specification.

Vandevenne, L., 2013. https://github.com/google/zopfli.

Vincent, L., 2007. Taking Online Maps Down to Street Level. *Computer* 40 (12), S. 118–120.

Wahl, R., Guthe, M., Klein, R., 2005. Identifying Planes in Point-Clouds for Efficient Hybrid Rendering. In: *The 13th Pacific Conference on Computer Graphics and Applications*. S. 0–0.

Ware, C., Osborne, S., 1990. Exploration and Virtual Camera Control in Virtual Three Dimensional Environments. *SIGGRAPH Comput. Graph.* 24 (2), S. 175–183.

Web3D-Consortium, 2005. ISO/IEC 19775:2004: Extensible 3D (X3D) Abstract Spec. Tech. rep.

Williams, L., 1983. Pyramidal Parametrics. *SIGGRAPH Comput. Graph.* 17 (3), S. 1–11.

Wimmer, M., Wonka, P., Sillion, F., 2001. Point-based impostors for real-time visualization. Springer.

Zitova, B., Flusser, J., 2003. Image registration methods: a survey. *Image and vision computing* 21 (11), S. 977–1000.

Wissenschaftliche Arbeiten der Fachrichtung Geodäsie und Geoinformatik der Leibniz Universität Hannover

(Eine vollständige Liste der Wiss. Arb. ist beim Geodätischen Institut, Nienburger Str. 1, 30167 Hannover erhältlich.)

Nr. 310	SEATOVIC, Dejan:	Methods for Real-Time Plant Detection in 3-D Point Clouds (Diss. 2013)
Nr. 311	SCHUNERT, Alexander:	Assignment of Persistent Scatterers to Buildings (Diss. 2014)
Nr. 312	GUERCKE, Richard:	Optimization Aspects in the Generalization of 3D Building Models (Diss. 2014)
Nr. 313	ZIEMS, Marcel:	Automatic verification of road databases using multiple road models (Diss. 2014)
Nr. 314	DINI, Gholam Reza:	Toward an automatic solution for updating building databases using high resolution space-borne stereo imaging (Diss. 2014)
Nr. 315	KERSTEN, Tobias:	Bestimmung von Codephasen-Variationen bei GNSS-Empfangsantennen und deren Einfluss auf Positionierung, Navigation und Zeitübertragung (Diss. 2014)
Nr. 316	BISKUPEK, Liliane:	Bestimmung der Erdorientierung mit Lunar Laser Ranging (Diss. 2015)
Nr. 317	STEINER, Christina:	Highspeed Stereo-Endoskopie für eng begrenzte Messvolumina (Diss. 2015)
Nr. 318	BANDIKOVA, Tamara:	The role of attitude determination for inter-satellite ranging (Diss. 2015)
Nr. 319	LIN, Miao:	Regional gravity field recovery using the point mass method (Diss. 2015)
Nr. 320	ZHANG, Lijuan:	Mining GPS-Trajectory Data for Map Refinement and Behavior Detection (Diss. 2015)
Nr. 321	ZADDACH, Sebastian:	Zum Beitrag Bayesscher Schätzverfahren in der Vergleichswertermittlung (Diss. 2016)
Nr. 322	SMYRNAIOS, Marios:	Carrier-phase Multipath in Satellite-based Positioning (Diss. 2016)
Nr. 323	MENZE, Moritz:	Object Scene Flow (Diss. 2016)
Nr. 324	WU, Hu:	Gravity field recovery from GOCE observations (Diss. 2016)
Nr. 325	XU, Xiangyang:	Terrestrial Laser Scanning for the Generation and Calibration of Finite Element Models (Diss. 2016)
Nr. 326	SAYYAD, Muhammad Naeem Shahzad:	Joint use and mutual control of terrestrial laser scans and digital images for accurate 3D measurements (Diss. 2016)
Nr. 327	SCHACK, Lukas:	Object-based matching of Persistent Scatterers to Optical Oblique Images (Diss. 2016)
Nr. 328	REICH, Martin:	Global Image Orientation from Pairwise Relative Orientations (Diss. 2016)
Nr. 329	KLINGER, Tobias:	Probabilistic multi-person localisation and tracking (Diss. 2016)
Nr. 330	SCHMIDT, Alena:	Markierte Punktprozesse für die automatische Extraktion von Liniennetzen in Rasterdaten (Diss. 2016)
Nr. 331	HOFMANN, Franz:	Lunar Laser Ranging - verbesserte Modellierung der Monddynamik und Schätzung relativistischer Parameter (Diss. 2017)
Nr. 332	BRIEDEN, Phillip:	Validierung von GOCE-Gravitationsgradienten in Kreuzungspunkten und Zukunftsperspektiven der Satellitengradiometrie (Diss. 2017)
Nr. 333	VON GÖSSELN, Ilka:	Simulationsbasierte Effizienzoptimierung von Messprozessen am Beispiel der tachymetrischen Netzmessung (Diss. 2017)
Nr. 334	HOFMANN, Sabine:	Potential von LiDAR Mobile Mapping für hochgenaue Karten (Diss. 2017)
Nr. 335	ALBERT, Lena:	Simultane Klassifikation der Bodenbedeckung und Landnutzung unter Verwendung von Conditional Random Fields (Diss. 2017)
Nr. 336	NIEMEYER, Joachim:	Verwendung von Kontext zur Klassifikation luftgestützter Laserdaten urbaner Gebiete (Diss. 2017)
Nr. 337	EGGERT, Daniel:	Effiziente Verarbeitung und Visualisierung von Mobile Mapping Daten (Diss. 2017)

Die Arbeiten werden im Rahmen des wissenschaftlichen Schriftenaustausches verteilt und sind nicht im Buchhandel erhältlich. Der Erwerb ist zu einem Stückpreis von € 25,00 bei den herausgebenden Instituten möglich.