BEUTH HOCHSCHULE
FÜR TECHNIK
BERLIN
University of Applied Sciences

DLR Deutsches Zentrum
für Luft- und Raumfahrt e.V.
in der Helmholtz-Gemeinschaft

Masterarbeit

zum Thema

Extraktion von senkrechten Fassadenebenen aus 3D-Punktwolken von Schrägluftbildern

vorgelegt von Magdalena Linkiewicz

BHT, Fachbereich III, Studiengang Geoinformation

Betreuer: Prof. Dr. Martin Kähler

Berlin, 30.10.2012

Danksagung

Ich möchte mich herzlich bei allen Personen bedanken, die mich bei der Arbeit unterstützt haben.

Bei der Firma DLR möchte ich mich für die Bereitstellung von den Obliquedaten bedanken.

Für die Hilfsbereitschaft und das Engagement möchte ich mich bei meinem Betreuer, Herrn Prof. Dr.-Ing. Martin Kähler, bedanken.

Besonderen Dank möchte ich dem Leiter der Abteilung Optische Informationssysteme, Sensorkonzepte und Anwendungen des DLR, Herrn Frank Lehmann, für die Ermöglichung meines Studiums aussprechen.

Weiterhin bedanke ich mich bei Herrn Alexander Wieden, dem Leiter der Photogrammetriegruppe der Abteilung Optische Informationssysteme, Sensorkonzepte und Anwendungen des DLR, für die fachliche und organisatorische Unterstützung. Den Mitgliedern der Photogrammetriegruppe: Frau Kristina Konecny, Herrn Karsten Stebner und Herrn Matthias Gessner möchte ich für die geleistete Hilfe und Unterstützungen aller Art meinen Dank aussprechen.

Bei Frau Stephanie Kaufhold bedanke ich mich für die Redaktion des Textes.

Bei Frau Gerlinde Irmscher und Herrn Reiner Petzoldt möchte ich mich für die Unterstützung während meiner Studienzeit bedanken.

Nicht zuletzt danke ich meinem Mann für seine Geduld und Ermutigungen bei der Entstehung dieser Arbeit und das große Verständnis in den letzten sechs Jahren meines Studiums.

Aufgabenstellung

Masterarbeit
für Magdalena Linkiewicz, Matr.Nr. 766752

BEUTH HOCHSCHULE
FÜR TECHNIK
BERLIN
University of Applied Sciences

Fachbereich III

**Bauingenieur- und
Geoinformationswesen
Labor für Photogrammetrie
Prof. Dr.-Ing. M. Kähler**

5.6.2012

Extraktion von senkrechten Fassadenebenen aus 3D-Punktwolken von Schrägluftbildern

In der Einrichtung *Optische Informationssysteme, Sensorkonzepte und Anwendungen*, des *Instituts für Robotik und Mechatronik* am *Deutschen Zentrum für Luft- und Raumfahrt* werden verschiedene Sensor- und Aufnahmesysteme, sowie neue Auswerteverfahren entwickelt. Eine der Entwicklungen, das *Modular Airborne Camera System (MACS)* dient beispielsweise auch zur Aufnahme von Schrägluftbildern.

Aus den Bilddaten eines solchen Aufnahmesystems entstehen 3D-Punktwolken durch Kombination von Luftbildern aus verschiedenen Blickwinkeln. Die Punktwolken beinhalten typischerweise eine große Anzahl von Messungspunkten an Gebäuden und zwar mit einer relativ hohen räumlichen Auflösung. Im Rahmen ihrer Masterarbeit soll Frau Linkiewicz Algorithmen zur Extraktion von Fassadenebenen aus einer solchen 3D-Punktwolke entwickeln.

Insbesondere ist das Ziel der Arbeit die Entwicklung von Methoden zur automatischen Extraktion von senkrechten Fassadenebenen. Dazu wird ein mehrstufiger Algorithmus benötigt. Vorgesehen ist dabei zunächst eine Filterung von Bodenelementen und Dächern. Für die verbleibenden Punkte auf senkrechten Fassaden kann die Berechnung von lokalen Orientierungen erfolgen (lineare Regression, Eigenwerte und Eigenvektoren). Anhand dieser Orientierungen wird eine Segmentierung der Punktwolke möglich. Anschließend sollen auf der Grundlage der Segmentierung Polygone erzeugt werden, die jeweils einer senkrechten Projektion von Fassadenebenen entsprechen.

Die Implementierung soll in *MatLab* erfolgen. Besonderes Augenmerk wird auf die Entwicklung eines robusten Algorithmus gelegt, der große Daten mit numerisch geringen Kosten verarbeitet. Für eine spätere industrielle Nutzung sollen möglichst nur grundlegende mathematische Verfahren verwendet werden, die sich anschließend auch in einer Programmiersprache wie C++ umsetzen lassen. Die Ergebnisse des Algorithmus werden an repräsentativen Gebäuden in Berlin Adlershof demonstriert und geprüft.

Bei der Bearbeitung der Aufgabenstellung ist darauf zu achten, dass die Vorgehensweise in allen Teilbereichen der Arbeit eingehend erläutert, alle getroffenen Entscheidungen gut begründet und die Ergebnisse übersichtlich zusammengestellt werden. Dabei sind insbesondere die zu entwickelnden Software-Module gut zu dokumentieren. Die Ausarbeitung ist in schriftlicher Form, die Unterlagen und Ergebnisse sind in digitaler Form einzureichen. Zur öffentlichen Präsentation der Masterarbeit ist sowohl ein Poster mit den wichtigsten Ergebnissen als auch eine kurze Zusammenfassung für eine Internetpräsentation anzufertigen.

Prof. Dr.-Ing. M. Kähler

Inhaltsverzeichnis

Danksagung i

Aufgabenstellung iii

Inhaltsverzeichnis v

1 Einleitung 1

2 Datengrundlage und Programmiertechnik 4

2.1 Kamerasystem 4

2.2 3D-Punktwolke und Referenzdaten 6

2.3 Prozessierung der Aufnahmen 8

2.4 Auswahl von Testgebäuden 9

2.5 Analyse der Qualität der 3D-Punktwolke 13

2.6 MatLab 16

3 Mathematisch-statistische Grundlagen 18

3.1 Grundlagen der linearen Einfachregression 18

3.2 Regression nach Pearson 19

3.3 RANSAC 22

4 Fassadenextraktion – lokaler Ansatz 24

4.1 Lokale Regression als Grundidee 24

4.2 Analyse einer Testpunktwolke 25

4.2.1 Einfluss der Fenstergröße 25

4.2.2 Einfluss der Lage der Punktwolke im Fenster 26

4.2.3 Einfluss des Rauschens 28

4.2.4 Einfluss der Form der Punktwolke im Fenster 29

4.3 Kriterium für die Fassadenerkennung 29

4.4 Quadratgitter, Zelle und Umgebung 31

4.4.1 Analyse einer realen Punktwolke 31

4.4.2 Definition der Umgebung 32
4.4.3 Beispiel für Regression mit Umgebung 33
4.4.4 Selektive Regression mit Umgebung 36
4.4.5 Testbeispiel für Gebäudeecke B2 37
4.5 Z-Filter zur Entfernung von Nicht-Fassaden-Objekten 38
4.5.1 Lokaler Z-Filter 39
4.5.2 Einsatz des lokalen Z-Filters 41
4.6 Lokale Regression auf dem Quadratgitter 43
5 Fassadenextraktion – globaler Ansatz 46
5.1 Initialisierung und Vereinigung von Gruppen von Fassadenzellen 48
5.1.1 Initialisierung von Gruppen von Fassadenzellen 48
5.1.2 Vereinigung von Gruppen von Fassadenzellen 51
5.2 Bestimmung von Fassadenstrecken durch Regression 52
5.3 Erstellung von geschlossenen Polygonzügen 54
5.3.1 Nachbarschaft von Fassadenstrecken 54
5.3.2 Vereinigung von Fassadenstrecken 56
5.3.3 Eckpunkte von Fassadenstrecken 58
5.4 Höhe von den unteren und oberen Fassadenkanten 59
5.5 Erstellung von Gebäuden 60
6 Ergebnisse und Plausibilitätsprüfung 61
6.1 Ergebnisse für das Gebäude B1 61
6.2 Ergebnisse für das Gebäude B2 63
6.3 Ergebnisse für das Gebäude B3 65
6.4 Ergebnisse für das Gebäude B4 68
6.5 Rechenzeit 73
7 Fazit und Ausblick 74
8 Quellenverzeichnis 77
Anhang [1]

A 1. Formelverzeichnis [1]

B 1. main [3]

B 2. make_cloud_in_cells [5]

B 3. z_filter [7]

B 4. eigenvectors_full_cells [9]

B 5. init_walls [11]

B 6. merge_walls [15]

B 7. make_segments [18]

B 8. find_pairs [21]

B 9. merge_gaps [25]

B 10. find_corner [29]

B 11. calculate_height [31]

B 12. make_buildings [33]

C 1. Berechnete Koordinaten der Fassadenebenen [38]

1 Einleitung

Gegenwärtig gibt es viele Beispiele für dreidimensionale Stadtmodelle, die hauptsächlich dem Zweck der Visualisierung dienen und ihre Anwendung im Bereich Tourismus, Stadtentwicklung und Marketing finden [vgl. Wieden (2012), S. 9 ff., Nagel et al. (2007), S. 151]. Es wird jedoch Bedarf an vielfältigsten Modellen beobachtet, die für Anwendungen im Bereich Städteplanung, Sicherheit, Katastrophenmanagement und Navigation geeignet sind. Diese Modelle müssen als Basis über eine genaue geometrische Beschreibung von Gebäuden und anderen urbanen Objekten verfügen. Darauf aufbauend werden semantische und topologische Eigenschaften dieser Objekte beschrieben [Gröger et al. (2005), S. 2].

Die manuelle Erstellung von 3D-Stadtmodellen ist ein aufwändiger und damit kostenintensiver Prozess. Dieser soll durch Automatisierung ökonomisch gestaltet werden. Es werden halbautomatische und vollautomatische Systeme unterschieden [Brenner (1999), S. 22-23] [Brenner et al. (2001)]. Trotz Fortschritten in der Entwicklung von vollautomatischen Systemen, werden vor allem halbautomatische Systeme eingesetzt.

Eine Technologie für die automatische Erstellung von 3D-Stadtmodellen wurde von der Firma C3-Technologies entwickelt. Diese Technologie gestattet eine detaillierte 3D-Darstellung von ausgewählten, meist urbanen Gebieten als unstrukturiertes Gitter mit Textur. Eine detaillierte Darstellung auch von komplexen Objekten ist möglich. In manchen Fällen werden Objektkanten jedoch nicht exakt oder auch fehlerhaft dargestellt. Die Hauptanwendung dieser Modelle ist die Visualisierung von urbanen Gebieten. Diese Modelle werden gegenwärtig von den Firmen Nokia und Apple verwendet. Insbesondere werden Gebäude zwar dreidimensional dargestellt, sie sind aber nicht als semantische Objekte bekannt.

In der Geowissenschaft werden die Detaillierungsgrade (Level of Detail) LoD0, LoD1, LoD2, LoD3 und LoD4 von Stadtmodellen unterschieden. Die oben genannten vollautomatisch erstellten 3D-Stadtmodelle auf Basis eines unstrukturierten Gitters mit Textur stellen ein Beispiel für LoD0 dar. Grundlage für die komplexeren Modelle LoD1 und höher sind Gebäude als semantische Objekte und deren Grundrisse. Zusammenfassend lässt sich feststellen, dass gegenwärtig verfügbare vollautomatische Techniken zur Erstellung von 3D-Stadtmodellen für die Detaillierungsgrade LoD1 und höher nicht zur Verfügung stehen. Gründe sind:

- mangelnde geometrische Genauigkeit,
- fehlende Semantik der Gebäude,
- fehlende Beziehungen zwischen den dargestellten Objekten.

In der Abteilung Sensorkonzepte und Anwendungen in der Einrichtung Optische Informationssysteme des Instituts für Robotik und Mechatronik am Deutschen Zentrum für Luft- und Raumfahrt e. V. (DLR) wird seit dem Jahr 2009 das Modulare Luftbild-Kamerasystem MACS (Modular Airborne Camera System) entwickelt, welches in einer Ausführung Obliqueaufnahmen (Schrägbildaufnahmen) bereitstellt. Im Gegensatz zum herkömmlichen Ansatz von senkrecht montierten Kameras haben Obliqueaufnahmen den Vorteil, dass Fassaden gut sichtbar sind. Eine Anwendungsmöglichkeit des MACS-Obliquesystems ist unter anderem die Erstellung von komplexen 3D-Stadtmodellen. Diese 3D-Stadmodelle sollen sowohl über eine genaue Geometrie als auch über Semantik und Topologie verfügen.

Neben der Entwicklung des neuen Sensors werden in der Abteilung Sensorkonzepte und Anwendungen Methoden der automatischen Prozessierung von den experimentell aufgenommenen Daten entwickelt. Das Ergebnis der Prozessierung der Obliqueaufnahmen ist unter anderem eine hochaufgelöste 3D-Punktwolke (50 Punkte/m^2 auf der Fassade). Diese Punktwolke soll als Grundlage für die Entwicklung von Methoden zur Erstellung von photogrammetrischen Produkten, wie zum Beispiel einem 3D-Stadmodell, das dem Bedarf der Anwender auf dem Markt entspricht und die aktuellen Standards von LoD3 erfüllt, dienen.

In der vorliegenden Masterarbeit wird eine Methode zur vollautomatischen Extraktion von senkrechten Fassadenebenen von Gebäuden aus einer 3D-Punktwolke entwickelt, implementiert und validiert. Unter einer senkrechten Fassadenebene wird hier die äußere Wand eines Gebäudes verstanden, die senkrecht zur Grundoberfläche steht. Ziel ist:

- eine hohe geometrische Genauigkeit von unter 0,5 m oder genauer,
- die Bestimmung der Lage und Höhe einzelner Fassaden,
- die Zusammenfassung von Fassaden eines Gebäudes zu dem semantischen Objekt Gebäude.

Als Ergebnis werden die Koordinaten der Fassadenecken für jedes detektierte Gebäude berechnet. Somit werden Gebäude erstens als semantische Objekte und zweitens als geometrische Objekte mit ihrem Umriss bestimmt. Die Koordinaten der extrahierten Fassadenebenen sind die Grundlage zur Bestimmung und zur geometrischen Entzerrung der Fassaden in den Obliqueaufnahmen. Damit bilden sie die Basis für die Texturierung des 3D-Modells und für weitere detaillierten Attributierungen von Objekten, wie Fenstern oder Balkonen.

In der hier entwickelten Methode werden die 3D-Punkte auf die zweidimensionale XY-Ebene projiziert, so dass die Fassadenebenen als Strecken erscheinen. Diese Strecken werden durch lineare Regression berechnet. In einem ersten Schritt werden Punkte, die sich außerhalb von Fassaden befinden, durch einen Filter entfernt. Da es nicht möglich ist, die

Fassadenstrecken durch globale Regression zu ermitteln, wird die 3D-Punktwolke durch ein regelmäßiges Quadratgitter in kleinere Bereiche aufgeteilt. Für Punkte aus jedem Bereich wird die lokale Richtung der Fassade durch lineare Regression bestimmt. Benachbarte Bereiche, die auf derselben Fassade liegen, werden zu Gruppen zusammengefasst. Im Ergebnis erhält man eine Aufteilung der 3D-Punktwolke in einzelne Fassaden. Durch die Punkte einer jeden Fassade wird eine lineare Regression durchgeführt. Diese liefert als Ergebnis, die Koordinaten der projizierten Fassade. Anschließend werden die Fassaden eines Gebäudes zusammengefasst.

Die verwendeten Algorithmen werden in MatLab implementiert. Ziel ist die Entwicklung von robusten Algorithmen mit numerisch geringen Kosten, die effizient für eine große Menge von Daten eingesetzt werden können.

Die Methodik wird an vier ausgewählten Gebäuden mit unterschiedlicher Komplexität validiert. Dazu gehören Gebäude mit komplexen Grundrissen, die teilweise mit gekrümmten Fassaden aufweisen.

Die Masterarbeit ist folgendermaßen gegliedert: In Kapitel 2 wird das Kamerasystem beschrieben und die 3D-Punktwolke analysiert. In Kapitel 3 werden die mathematisch-statistischen Grundlagen der entwickelten Methode erläutert. In Kapitel 4 wird die Grundidee der Methode und die Vorbereitung der Daten beschrieben. In Kapitel 5 wird der mehrstufige Algorithmus zur Fassadenextraktion vorgestellt. In Kapitel 6 werden die Ergebnisse der Masterarbeit erläutert und durch Vergleiche mit einem Nadir-TrueOrthobild überprüft. Abschließend werden in Kapitel 7 das Fazit und Vorschläge zur Verbesserung der entwickelten Algorithmen vorgestellt. Der Anhang enthält eine Übersicht über die verwendeten mathematischen Symbole und im Programmcode verwendeten Parameter, den implementierten Programmcode und die durch den entwickelten Algorithmus berechneten Koordinaten der Fassadenebenen der Testgebäude.

2 Datengrundlage und Programmiertechnik

In diesem Kapitel wird das vom DLR entwickelte modulare Luftbildkamerasystem MACS (Modular Airborne Camera System) beschrieben. Insbesondere wird auf das MACS-Oblique Modul eingegangen. Weiterhin wird die Erstellung einer 3D-Punktwolke auf Basis von Aufnahmen des Kamerasystems MACS-Oblique beschrieben. Danach werden vier ausgewählte Gebäude dargestellt, die zur Evaluierung der hier entwickelten Methode dienen. Dabei werden die 3D-Punktwolke kritisch analysiert und mögliche Probleme für den zu entwickelnden Algorithmus beschrieben. Abschließend wird das Programm MatLab vorgestellt.

2.1 Kamerasystem

Seit dem Jahr 2009 wird in der Abteilung Sensorkonzepte und Anwendungen in der Einrichtung Optische Informationssysteme des Instituts für Robotik und Mechatronik am Deutschen Zentrum für Luft- und Raumfahrt e. V. (DLR) das Modulare Luftbildkamerasystems (MACS) entwickelt. Dieses Kamerasystem ist durch einen flexiblen Aufbau gekennzeichnet, bei dem durch Anpassung von Sensoren und Kameramodulen verschiedenen Anforderungen erfüllt werden können. Gegenwärtig stehen fünf auf Industriekameramodulen und Industrieoptiken basierende Varianten des MACS-Systems zur Verfügung. Diese werden zur Erhebung verschiedener experimenteller Luftbilddaten verwendet.

Die Variante MACS-Jet dient der Erhebung von Luftbildaufnahmen aus schnellfliegenden Trägern. Das System MACS-RT wird im Bereich der Echtzeitkartierung eingesetzt. Das System MACS-micro wird für die digitale Kartierung kleinräumiger Gebiete genutzt und auf Klein- und Kleinstdrohnen eingesetzt. Die Variante MACS-TumbleCam wird zur 3D-Kartierung höchster Auflösung und Genauigkeit verwendet. Die Variante MACS-Oblique dient der Untersuchung perspektivischer Aufnahmegeometrien und der automatischen Ableitung semantischer 3D-Karten [Quelle: DLR (2012)].

Das System MACS-Oblique wird seit dem Frühjahr 2011 entwickelt. Es besteht aus fünf synchron auslösenden Kameraköpfen, wobei drei Kameras senkrecht und zwei Kameras schräg ausgerichtet sind (Abbildung 1).

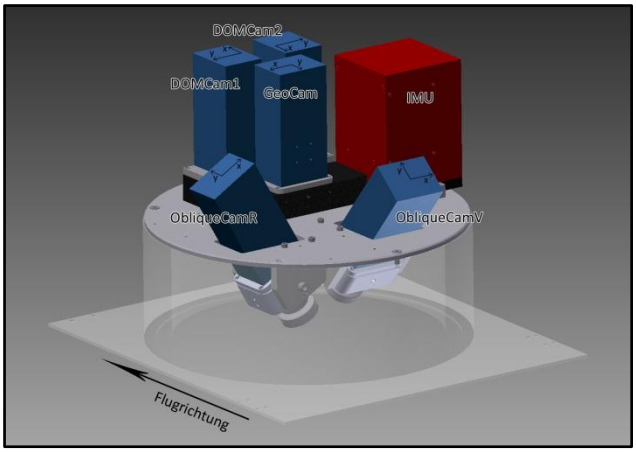

Abbildung 1: Aufbau des Kamerasystems MACS-Oblique. Quelle: DLR.

Die senkrecht ausgerichteten Kameras bestehen aus einem GeoCam-Modul und zwei hochauflösenden DOMCam-Modulen. Das GeoCam-Modul ist mit einem 50mm-Objektiv und mit einem Infrarotfilter ausgestattet. Das DOMCam-Modul besteht aus zwei Kameraköpfen mit 70mm-Objektiven. Diese Module stellen RGB-Informationen durch klassische Senkrechtaufnahmen zur Verfügung. Die zwei schräg ausgerichteten Kameramodule, ObliqueCamV und ObliqueCamR, verfügen über 70mm-Objektive. Die ObliqueCamR ist um 30° quer zur Flugrichtung verkippt und blickt in Richtung Steuerbord. Die ObliqueCamV ist um 40° in Flugrichtung verkippt [Wieden (2012), S. 42-43].

Die fünf Kameraelemente sind Teil der Serie pco.4000 vom Hersteller PCO. Dabei werden CCD-Sensoren von Kodak verwendet. Diese haben eine spektrale Empfindlichkeit von 320 nm bis 1000 nm und eine Sensorgröße von 4008 x 2672 Pixel [Lehmann et al (2011), S. 437].

Bilder der fünf MACS-Oblique-Kameras zu einem Aufnahmezeitpunkt werden in der Abbildung 2 dargestellt.

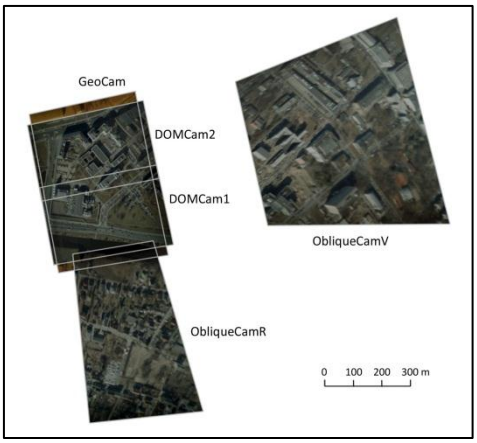

Abbildung 2: Bilder der fünf MACS-Oblique-Kameras zu einem Aufnahmezeitpunkt. Quelle: Konecny (2011).

Das MACS-System wurde mit dem GPS/IMU-System AEROcontrol der Firma IGI (Ingenieur-Gesellschaft für Interfaces) mbH ausgestattet und bestimmt die Positionen und Orientierungswinkel der Sensoren zu den Auslösezeitpunkten der Kameras [Lehmann et al. (2011), S. 440].

2.2 3D-Punktwolke und Referenzdaten

Bei den in dieser Arbeit verarbeiteten Daten handelt es sich um eine 3D-Punktwolke (Abbildung 3). Diese wurde aus den Schrägaufnahmen und Senkrechtenaufnahmen des Kamerasystems MACS-Oblique generiert. Diese Punktwolke ist ein Endprodukt der im folgenden Kapitel erläuterten Prozessierungskette der Aufnahmen des MACS-Obliquesystems.

Abbildung 3: In MeshLab visualisierte, eingefärbte 3D-Punktwolke. Quelle: DLR.

Der Einsatz des MACS-Obliquesystems, welches zur Erstellung der im Rahmen dieser Arbeit verwendeten Punktwolke genutzt wurde, fand im März 2011 statt. Es wurde ein Gebiet in Berlin-Adlershof beflogen, welches der Abteilung Sensorkonzepte und Anwendungen als Testfeld für Kameraerprobungen dient. Dieses Gebiet ist in der Abbildung 4 zu sehen. Es wurden insgesamt neun Flugstreifen aufgenommen. Sieben Flugstreifen sind in Richtung Nordost-Südwest und zwei in Richtung Nordwest-Südost ausgerichtet (Abbildung 4). Die Flughöhe betrug ca. 710 m. Es wurden 749 Bilder pro Kamera aufgenommen, also insgesamt 3745 Bilder [Konecny (2011), S. 15].

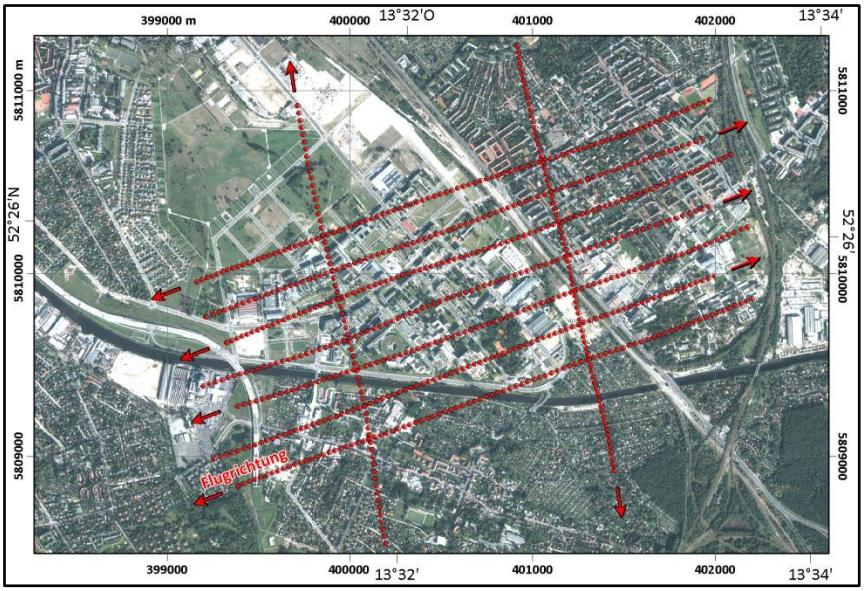

Abbildung 4: Darstellung der neun Flugstreifen der Befliegung des Kamerasystems MACS-Oblique. Quelle: DLR, Bearbeitung: Konecny (2011).

Die Befliegung wurde alternierend mit einer Längsüberlappung von ca. 90% und einer Querüberlappung von ca. 50% durchgeführt. Diese Aufnahmekonstellation hat die Aufnahme von Luftbilddaten aus vier Perspektiven gewährleistet.

Als Referenz für die Bewertung der in dieser Masterarbeit erzielten Ergebnisse dient das auf Basis der Senkrechtaufnahmen erstellte Nadir-DOM und ein Nadir-Trueorthobild (Abbildung 5). Diese Daten liegen im Rasterformat vor. Die Bodenauflösung des Nadir-TrueOrthobildes beträgt 0,15 m. Die Höhenauflösung des Nadir-DOM beträgt 0,05 m. Am Ende der Arbeit soll geprüft werden, ob die extrahierten Fassadenebenen zu dem Nadir-Trueorthobild passen. Die absolute Genauigkeit der extrahierten Fassaden wird in dieser Arbeit nicht untersucht. Die Thematik der absoluten Genauigkeit der 3D-Punktwolken aus den Obliqueaufnahmen des MACS-Systems wurde in [Konecny (2011)] besprochen.

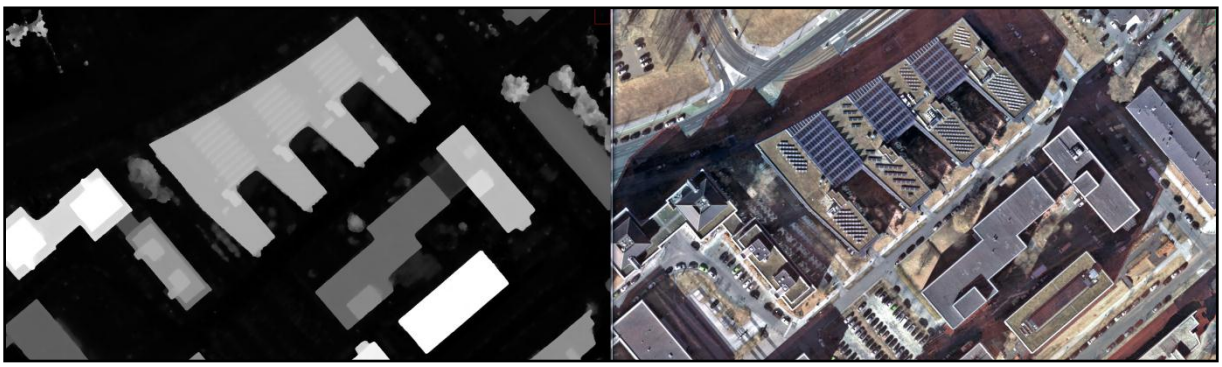

Abbildung 5: links: Nadir-DOM, rechts: Nadir-TrueOrthobild des Kamerasystems MACS-Oblique. Quelle: DLR.

2.3 Prozessierung der Aufnahmen

Die Aufnahmen einer jeden Kamera werden in separaten Aerotriangulationen verarbeitet. Als Ergebnis liefert eine Aerotriangulation die Position und die Ausrichtung der Kamera während der Aufnahme für jedes Bild im übergeordneten Raumkoordinatensystem [Kraus (2004), S. 273]. Die Aerotriangulationen werden im Modul MATCH-AT der Software Inpho von Trimbel durchgeführt. Zu jedem Auslösezeitpunkt wird die Lage des Projektionszentrums der Kamera (X, Y, Z) und die Ausrichtung des Kamerasystems (ω, φ, κ) im übergeordneten Raumkoordinatensystem mit Hilfe der GPS/IMU-Einheit näherungsweise bestimmt. Diese sechs Parameter (X, Y, Z, ω, φ, κ) werden zur Initialisierung der äußeren Orientierung bei den Aerotriangulationen genutzt. Neben den approximierten Parametern der äußeren Orientierung werden für jede Aerotriangulation die bekannten Parameter der inneren Orientierung (Bildhauptpunkt, Kamerakonstante, Verzeichnung) verwendet. Diese Parameter sind sensorspezifisch. Zusätzlich werden zur Verbesserung der absoluten Genauigkeit und zur Kontrolle der Ergebnisse der Aerotriangulationen tachymetrisch gemessene Passpunkte verwendet. Zur Kontrolle und zur Vereinigung der separaten Aerotriangulationen werden neben den Passpunkten auch manuelle, im Programm Inpho gemessene, Punkte benutzt.

Die Durchführung der Aerotriangulation von Schrägluftbildern ist schwieriger als die Aerotriangulation von Senkrechtaufnahmen. Besonders schwierig ist die Erstellung der Verknüpfung zwischen Bildern in benachbarten Flugstreifen. Obliquebilder in alternierenden Nachbarstreifen weisen zum großen Teil eine unterschiedliche Ansicht der gleichen Objekte auf. Dies erschwert die Verknüpfung der Bilder und damit die Aerotriangulation. Diese Problematik und die angewändete Lösung wurden in der Masterarbeit von [Konecny (2011)] und [Wieden (2012)] beschrieben.

Im nächsten Schritt werden die Parameter der äußeren Orientierung (also die Ergebnisse der Aerotriangulation) und die Parameter der inneren Orientierung zur Berechnung eines digitalen Oberfächenmodells (im Folgenden DOM genannt) genutzt. Das DOM beschreibt die Oberfläche unter Einbeziehung von Kunstbauten und der Vegetation [Kraus (2004), S. 354]. Ein solches Oberflächenmodell wird in der Abteilung Sensorkonzepte und Anwendungen mit Hilfe des SGM-Algorithmus (Semi-Global Matching) erstellt. Dieser Algorithmus ist in der Lage eine relative Genauigkeit im Pixelbereich zu erzielen [Hirschmüller (2008), S. 328ff.].

Der SGM-Algorithmus wurde entwickelt, um senkrecht aufgenommene Luftbilddaten verschiedenster Sensoren hochgenau prozessieren. Die Bilddaten der senkrecht ausgerichteten Kameras des MACS-Obliquesystem können ohne weitere Vorbereitung mit dem SGM-Algorithmus prozessiert werden. Im Ergebnis wird ein Nadir-DOM und ein

Nadir-TrueOrthobild, also ein geometrisch richtiges Foto, das die Kunstbauten berücksichtigt [Kraus (2004), S. 410ff.], erstellt.

Die Bilddaten, die mit den schrägblickenden Kameras des MACS-Obliquesystems aufgezeichnet wurden, können mit der gegenwärtigen Version des SGM-Algorithmus nicht ohne weiteres prozessiert werden. Um eine SGM-Prozessierung zu ermöglichen, werden die Parameter der äußeren Orientierung der Schrägaufnahmen künstlich so transformiert, als ob die Aufnahmen aus der Nadirposition aufgenommen worden wären. Die Parameter der inneren Orientierung werden nicht verändert.

Diese Transformation ist eine Rotation. Die entsprechende Rotationsmatrix wird anhand der Parameter ω, φ, κ bestimmt. Da die Schrägluftbilder aus verschiedenen Blickwinkeln und Himmelsrichtungen aufgenommen wurden, gibt es keine gemeinsame Rotationsmatrix für alle Schrägbilder. Deshalb werden die Bilder in Gruppen nach ähnlichen Werten von ω, φ, κ aufgeteilt. Es wurden insgesamt sieben Gruppen von Bildern erstellt. Jede Gruppe wird mit einer einzigen Rotationsmatrix transformiert.

Für jede Gruppe von Bildern wird nach der Transformation der Parameter der äußeren Orientierung ein Oberfächenmodell mit Hilfe des SGM-Algorithmus erstellt. Es werden insgesamt sieben DOM jeweils in einem lokalen Koordinatensystem erstellt. SGM erstellt diese DOM in Form von 2,5D-Rasterdateien. Diese werden in ASCII-Dateien mit den 3D-Koordinaten der Punkte konvertiert. Die entsprechenden Punkte werden mit der inversen Rotationsmatrix in das ursprüngliche Koordinatensystem der Aerotriangulation rücktransformiert. Im Ergebnis erhält man sieben 3D-Punktwolken im Raumkoordinatensystem. Diese sieben 3D-Punktwolken werden zusammen in einer ASCII-Datei gespeichert. Somit entsteht eine gemeinsame 3D-Punktwolke aller Schrägaufnahmen. Ihr Raumkoordinatensystem ist definiert als UTM, Zone 33 N, Datum WGS84. Die Punktdichte dieser Punktwolke beträgt ca. 50 Punkte/m^2 auf der Fassade. Die Bodenpunktdichte beträgt ca. 90 Punkte/m^2 und die Dachpunktdichte beträgt ca. 250 Punkte/m^2.

2.4 Auswahl von Testgebäuden

In der Praxis auftretende Gebäude haben unterschiedliche Formen mit unterschiedlicher Komplexität. Der zu entwickelnde Algorithmus soll für möglichst alle auftretenden Formen deren Fassaden extrahieren. Um dies zu prüfen, werden vier Gebäude B1, B2, B3, B4 mit unterschiedlicher Komplexität ausgewählt und beschrieben (Abbildung 6).

Diese Gebäude umfassen sowohl einfache als auch komplexe Grundrisse. Ein Beispiel für einen einfachen Grundriss ist ein Rechteck. Ein komplexer Grundriss zeichnet sich aus durch verschiedene Faktoren aus, wie:

- eine größere Anzahl von Fassaden (B2, B3, B4),
- kurze Fassaden, bis zu 6 m (B2, B4),
- Fassaden, die nicht senkrecht, sondern zum Beispiel in einem spitzen oder stumpfen Winkel zueinander stehen (B3),
- gekrümmte Fassaden (B2, B3),
- Fassaden, die über ihre gesamte Länge durch zurückgesetzte Fensterfronten, Außenjalousien oder spezielle Topographie nicht auf einer Ebene liegen (B3),
- Fassaden, die lokal Vorsprünge besitzen, wie Balkone, Feuertreppen oder Verbindungsgänge (B2, B3, B4),
- Fassaden mit Verglasung - die Verglasung erschwert die Prozessierung mit dem SGM-Algorithmus (B1, B3),
- Fassaden mit geringer Höhe, bis zu 6 m (B4).

Ein Faktor, der nicht von der Fassade abhängig ist, aber die Darstellung der Fassade in der Punktwolke bestimmt, ist die Sichtbarkeit bei Schrägbildaufnahmen. Deshalb wird ein zusätzlicher Punkt berücksichtigt:

- Vegetation oder andere Objekte in der Nähe einer Fassade (B4).

Abbildung 6: Ausgewählte Gebäude: B1, B2, B3 sowie B4 mit Gebäudeteilen a, b, c in Berlin Adlershof. Diese Abbildung und alle weiteren Draufsichten in dieser Arbeit sind genordet.

Das Gebäude B1 besitzt mit der Form eines Quaders die geringste Komplexität (Abbildung 6). Die südöstliche Fassade des Gebäudes ist zu einem großen Teil verglast (Abbildung 7).

Abbildung 7: Gebäude B1 mit der teilweise verglasten Fassade.

Das Gebäude B2 weist einen komplexen Gebäudeumriss auf (Abbildung 6). Die Fassadenebenen schneiden sich nicht unter 90°. Die horizontalen Fassadenlängen sind unterschiedlich lang und variieren von 2 m bis zu 55 m. Der Eingang ist halbrund (Abbildung 8, links). Das Gebäude ist mit einem Nachbargebäude durch ein verglastes Verbindungsgebäude verbunden (Abbildung 8, rechts).

Abbildung 8: Gebäude B2 mit einem halbrunden Eingang (links) und einem verglasten Verbindungsgebäude (rechts).

Das Gebäude B3 weist ebenfalls einen komplexen Gebäudeumriss auf (Abbildung 6). Die Fassadenebenen schneiden sich unter verschiedenen Winkeln. Die nördliche Fassade ist gekrümmt (Abbildung 9, Mitte). Die Fassaden sind teilweise voll verglast (Abbildung 9, rechts). Aus den Fassaden ragen Elemente wie Rettungstreppen oder Balkone heraus (Abbildung 9, links). Das Gebäude ist von einer dichten Bepflanzung umgeben. Diese war zum Zeitpunkt der Aufnahme im März noch nicht laubtragend. Die horizontalen Fassadenlängen weisen Längen von 12 m bis zu 108 m auf.

Abbildung 9: Gebäude B3 mit Balkonen und Rettungstreppen an den Fassaden (links), gekrümmter vorderer Fassade (mitte) und vollständig verglasten Fassaden zwischen den vier Gebäudeteilen.

Die drei Teile des Gebäudes B4 haben unterschiedliche Höhen. Der Gebäudeumriss weist eine große Anzahl von Fassadenebenen mit unterschiedlichen horizontalen Längen von 4 m bis zu 52 m auf (Abbildung 10, Bilder 1-3 von links). Vor manchen Fassaden befindet sich Vegetation (Abbildung 10, rechts).

Abbildung 10: Gebäude B4 mit verschiedenen Fassadenhöhen und unterschiedlichen horizontalen Fassadenlängen (die drei Bilder links) und mit dichter Vegetation (rechts).

Für jedes der Gebäude B1, B2, B3 und B4 wird jeweils eine lokale 3D-Punktwolke aus der 3D-Punktwolke durch Beschnitt erstellt. Die Entwicklung und Evaluierung des Algorithmus zur Extraktion der Fassadenelemente wird auf den lokalen Punktwolken, die im Folgenden als Punktwolken B1, B2, B3 und B4 bezeichnet werden, durchgeführt. Dadurch wird die Berechnungszeit reduziert und die Analyse von Problemen erleichtert.

Für das Gebäude B4 wurde der Beschnitt zunächst mit einem umgebenden Polygon in ArcGIS von ESRI durchgeführt (Abbildung 11). Da die Verarbeitungszeit in ArcGIS sehr hoch ist, wird für die Gebäude B1, B2 und B3 eine andere Methode benutzt. Hier wird der Beschnitt mit einem umgebenden achsenparallelen Rechteck in MatLab erstellt.

Die lokalen Punktwolken enthalten

- Punktwolke B1: 940.000 Punkte,
- Punktwolke B2: 2.208.000 Punkte,
- Punktwolke B3: 4.385.000 Punkte,
- Punktwolke B4: 1.611.000 Punkte.

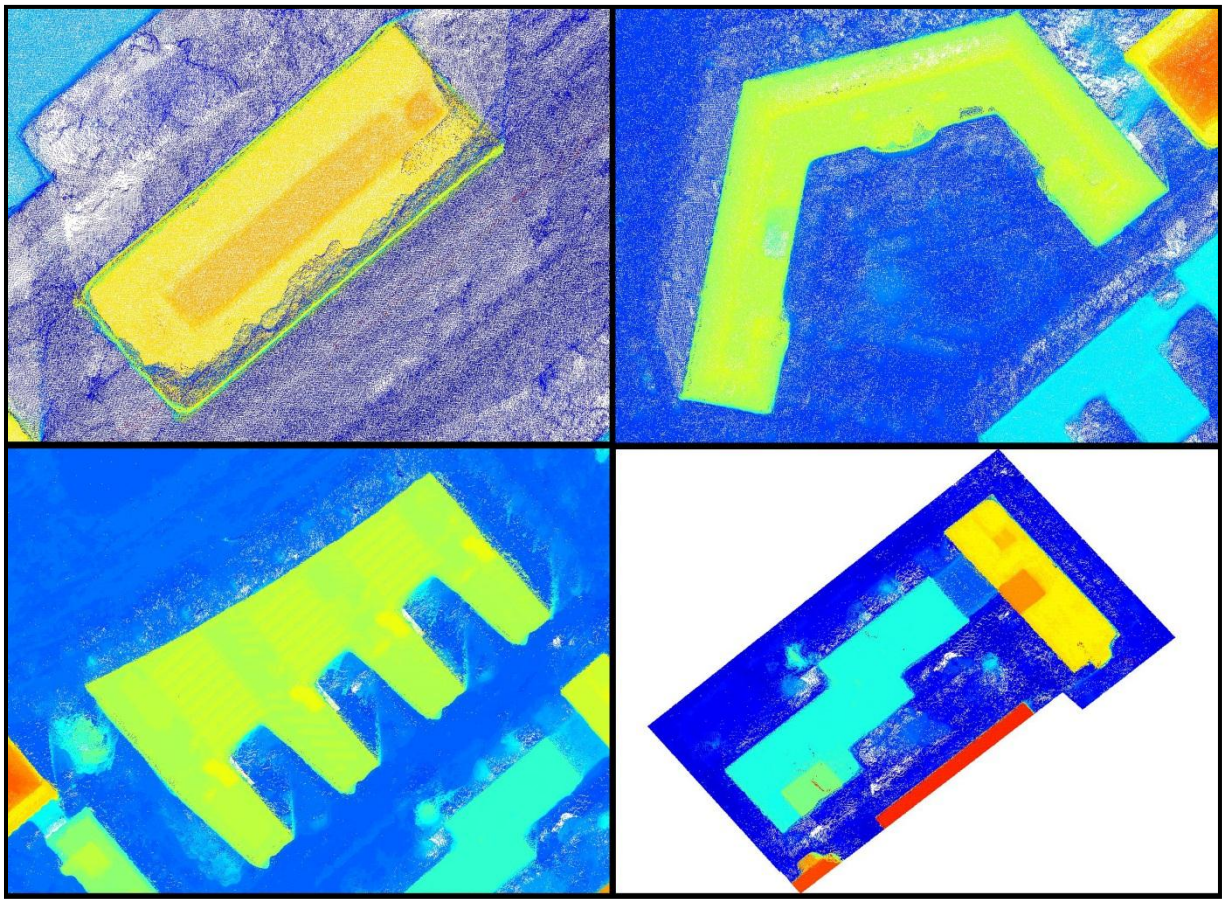

Abbildung 11: Auf die XY-Ebene projizierte 3D-Punktwolken (Farbe entspricht Z-Koordinate, jeweils unterschiedliche Farbskala): Gebäude B1 (oben links), B2 (oben rechts), B3 (unten links) und B4 (unten rechts).

2.5 Analyse der Qualität der 3D-Punktwolke

In diesem Kapitel wird die Qualität der 3D-Punktwolke analysiert. Ziel ist es Erfahrungen über die Qualität und mögliche Fehler der Datengrundlage zu sammeln, da diese die Extraktion von Fassadenelementen erschweren. Mit diesem Wissen sollen die Algorithmen zu Fassadenextraktion entwickelt werden. Alle 2D-Abbildungen in dieser Arbeit sind genordet.

Zunächst soll die horizontale Abweichung der 3D-Punkte von der Fassade untersucht werden. Dazu werden verschiedene Regionen der Gebäude untersucht und die Punkte, farbig entsprechend ihrer Z-Koordinate, dargestellt.

Ein Beispiel für eine Fassade ohne Fehler ist in Abbildung 12 dargestellt. Eine grobe Messung zeigt, dass der maximale Abstand zur gedachten Fassadenebene ca. 20 – 40 cm beträgt (vgl. Abschnitt 4.4.5 für Werte zur Standardabweichung). Diese Abweichung ist typisch für die betrachteten Gebäude und wird bei keinem der Gebäude unterschritten.

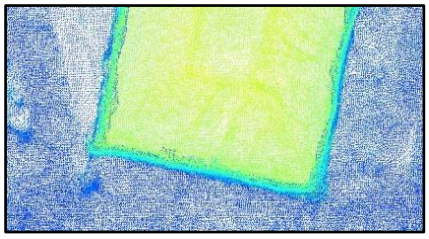

Abbildung 12: Ausschnitt aus Gebäude B2.

Es gibt Fassaden, bei denen die horizontale Abweichung der 3D-Punkte von der Fassade größer ist. In Abbildung 13 wird das Gebäude B1 dargestellt. Es fällt auf, dass die Bodenpunkte (blau) deutlich unterhalb des Daches liegen. Die Verschiebung beträgt bis zu 4,5 m. Bei einer Begehung ist festgestellt worden, dass die entsprechende Fassade zum großen Teil verglast ist. Es wird vermutet, dass hierdurch Prozessierungsfehler verursacht werden. Die Konsequenz ist die falsche Lage der Bodenpunkte.

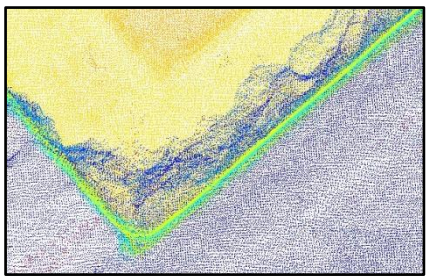

Abbildung 13: Ausschnitt aus dem Gebäude B1. Verglaste südliche Ecke und südöstliche Fassade.

Das Gebäude B1 weist weitere Abweichungen auf. Die nord-westliche Fassade wird durch zwei Reihen von Punkten repräsentiert (Abbildung 14). Der Abstand dieser Reihen beträgt ca. 1 m.

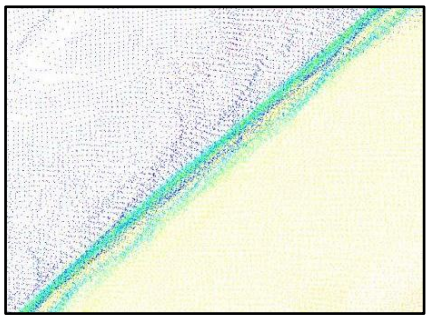

Abbildung 14: Ausschnitt aus Gebäude B1. Doppelte Punktreihe (türkisfarbene Punkte) repräsentiert eine Fassade.

Hohe Vegetation, die sich nahe an den Fassaden befindet, kann die Fassadenextraktion durch Verdeckung erschweren. Eine teilweise verdeckte Fassade ist für die Luftbildkamera nicht vollständig sichtbar (Abbildung 15). Da der verdeckte Teil nicht auf den Aufnahmen vorhanden ist, können dort auch keine 3D-Informationen erzeugt werden. In der

Abbildung 15 ist sichtbar, dass die Streuung im Bereich der kurzen Fassade deutlich größer ist als für die beiden langen Fassaden.

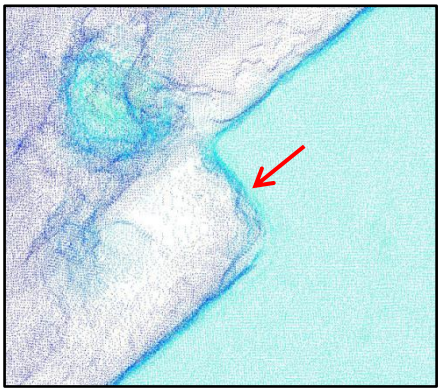

Abbildung 15: Ausschnitt aus Gebäude B4. Teilweise verdeckte Fassade (roter Pfeil).

Ein weiteres Problem sind scheinbare linienhafte Objekte (Abbildung 16). Hier ist deutlich die Ecke eines Gebäudes zu erkennen, sowie eine unerwartete Struktur. Ein Vergleich mit anderen Ecken des Gebäudes zeigt, dass sich diese Struktur dort wiederholt. Die Analyse des Orthobildes zeigt, Schatten im Bereich dieser Strukturen. Wahrscheinlich wurden diese Schatten vom SGM-Algorithmus als Objekte erkannt und prozessiert.

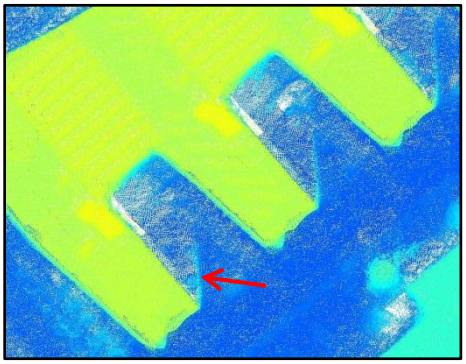

Abbildung 16: Ausschnitt aus Gebäude B3. Schattenkante (roter Pfeil) kann mit Fassaden verwechselt werden.

Ebenfalls problematisch ist die Verdeckung von Fassaden durch Objekte wie zum Beispiel Rettungstreppen oder Balkone. In Abbildung 17 ist ein Ausschnitt der 3D-Punktwolke dargestellt, bei dem eine Rettungstreppe aus der Fassade herausragt. Punkte, welche die Rettungstreppe repräsentieren, werden mit den Fassadenpunkten vermischt. Es ist deshalb schwierig festzustellen, welche Punkte zur Fassade und welche zur Rettungstreppe gehören.

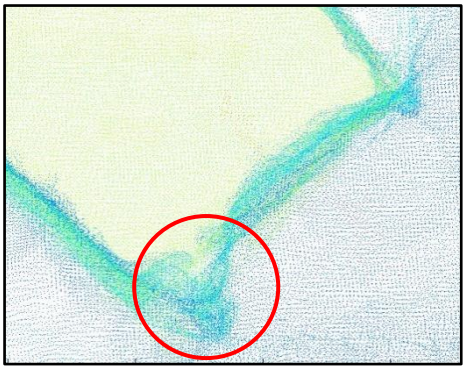

Abbildung 17: Ausschnitt aus Gebäude B3. Streuung an der Fassade mit der herausragenden Rettungstreppe (roter Kreis).

2.6 MatLab

Für die in dieser Arbeit implementierten Algorithmen wird MatLab angewändet (Abbildung 18). MatLab ist eine kommerzielle Software der Firma The MathWorks Inc. und wird unter anderem für die Lösung von mathematischen Problemen verwendet. MatLab wird seit Ende der siebziger Jahre des vergangenen Jahrhunderts entwickelt und war ursprünglich auf Matrizen-Berechnungen spezialisiert. Heutzutage bietet MatLab eine große Menge von verschiedenen Funktionalitäten und wird in vielen Bereichen der Industrie, Forschung und Lehre als Standardwerkzeug genutzt [Schweizer (2009), S .2]. Zu den nützlichen Funktionalitäten von MatLab gehört die grafische Darstellung der Ergebnisse. Diese ermöglicht durch die Visualisierung von Teilergebnissen und somit die Kontrolle des entwickelten Programmes.

Dadurch, dass in MatLab viele Konzepte zur Matrizen-Berechnungen implementiert sind, ist die Bearbeitung von 2D- und 3D-Punktwolken komfortabel. Dies soll am Beispiel einer 3D-Punktwolke `cloud` erläutert werden, deren Punkte zeilenweise aufgelistet sind. Die Koordinaten befinden sich in den Spalten 1 bis 3. Zur Bestimmung des maximalen Z-Werts, dient der Aufruf `max(cloud(:,3))`. Hierbei steht der Operator ‚:' für alle Zeilen der Variable `cloud` und die `3` für die dritte Spalte.

Der hier entwickelte Algorithmus zur Fassadenextraktion ist ein Prototyp. Dieser soll eventuell in der Zukunft in C++-Code umgewandelt werden. Die Konvertierung ist beispielsweise mit Hilfe des Programms MatLab Coder möglich, das ebenfalls zur Produktpalette der Firma MathWorks gehört. Die Konvertierung kann auch vollständig automatisch durchgeführt werden und ist umso einfacher, je einfacher die Funktionen sind, die in MatLab verwendet werden. Die in dieser Arbeit verwendeten MatLab-Funktionen und Datenstrukturen sind zum überwiegenden Teil einfach. Zu den komplexeren Funktionen gehören die Erstellung eines

Histogramms und die Berechnung von Eigenwerten und Eigenvektoren einer 2x2-Matrix. Diese beiden Funktionen lassen sich ohne größeren Aufwand in anderen Programmiersprachen implementieren.

Es wird darauf geachtet, dass der im MatLab geschriebene Code nicht numerisch aufwändig ist, d.h. dass alle Funktionen, soweit möglich nur lineare Kosten verursachen. Das bedeutet, dass der Rechenaufwand höchstens proportional mit der Anzahl der Variablen steigt und nicht quadratisch.

Der Code und alle Kommentare werden auf Grund der Anforderungen des DLR auf Englisch geschrieben.

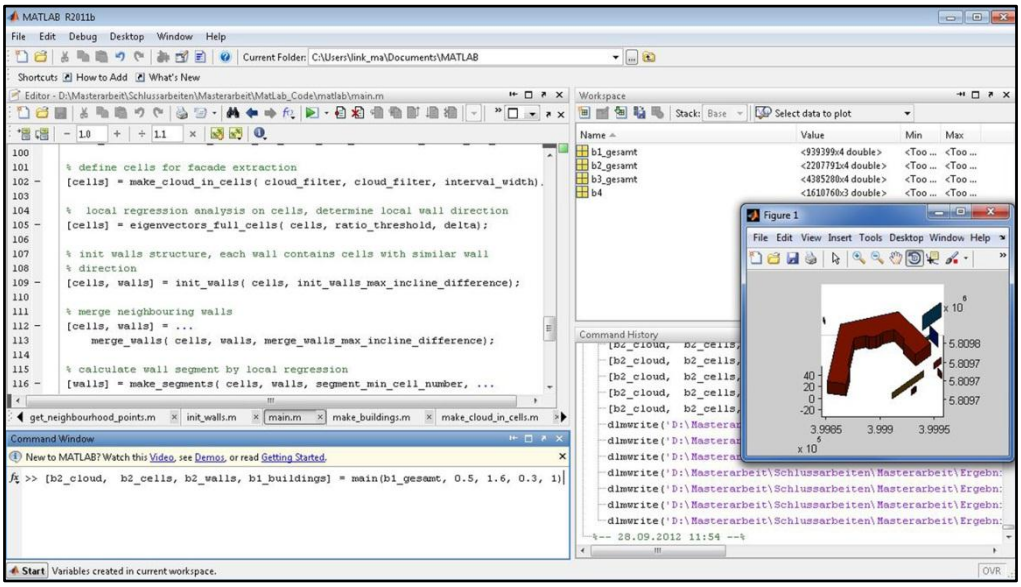

Abbildung 18: GUI vom MatLab besteht aus mehreren Fenstern u.a. aus dem Fenster zum Editieren des Codes (oben links) und aus dem Befehlsfenster (unten links) zum Ausführen des Codes.

3 Mathematisch-statistische Grundlagen

In diesem Kapitel werden die mathematisch-statistischen Grundlagen für diese Arbeit beschrieben. Wie Eingangs beschrieben ist die Aufgabe in dieser Arbeit, die Extraktion von senkrechten Fassadenebenen aus der 3D-Punktwolke umzusetzen. Dazu werden Punkte dieser Punktwolke werden auf die XY-Ebene projiziert, so dass die Fassadenebenen als linienhafte Objekte dargestellt werden. Wir betrachten also im Folgenden eine Punktwolke mit den Punkten $P_i(x, y), i = 1, \dots, n$ in einem zweidimensionalen kartesischen Koordinatensystem. Um die projizierten Fassadenebenen zu beschreiben, soll unter anderem eine Regressionsmethode angewändet werden.

Zunächst werden zwei Methoden der linearen Regression beschrieben: die lineare Einfachregression [Bahrenberg (1990), S. 135ff.] und die Regression nach Pearson [Pearson (1901), S. 559ff.]. Anschließend wird die modernere Regressionsmethode RANSAC erläutert.

3.1 Grundlagen der linearen Einfachregression

Die lineare Einfachregression ist eine Methode zur Beschreibung einer mittleren Gerade durch eine 2D-Punktwolke. Diese Gerade wird im Folgenden Ausgleichsgerade genannt und wird durch die Geradengleichung beschrieben:

$$y = a \cdot x + b, x \in \mathbb{R}$$

Hierbei sind a und b die zu bestimmenden Parameter der Geraden.

Die Ausgleichsgerade erfüllt die Bedingung, dass die Summe der Quadrate der vertikalen Entfernungen von Punkten zu dieser Gerade minimal ist (Abbildung 19). Diese Bedingung ist in der Literatur als least square principle bekannt [Bahrenberg (1990), S. 135 ff.].

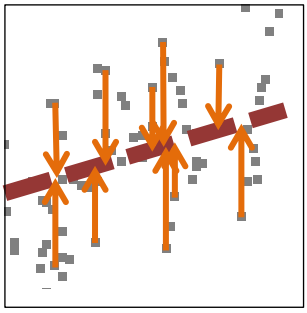

Abbildung 19: Regression nach dem Minimum Least Square Prinzip, die Summe der Quadrate der Abstände in Y-Richtung wird minimiert.

Diese Methode hat die Eigenschaft, dass sie von der Wahl des Koordinatensystems bzw. der Lage der Punktwolke im Koordinatensystem, abhängt (Abbildung 20 links). Bei einer Drehung der 2D-Punktwolke ändert sich die Lage der Ausgleichsgerade zur Punktwolke [Pearson (1901), S. 559 ff.]. Weiterhin ist diese Methode für die Punkte, die parallel zur y-Achse verteilt sind nicht anzuwenden, weil Geraden, die parallel zur y-Achse sind, nicht in der Form $y = ax + b$ beschreiben werden können.

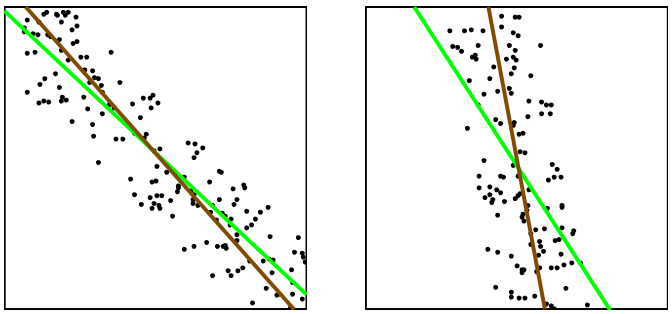

Abbildung 20: Regression nach dem Minimum Least Square Prinzip mit Regression nach x (grün) und nach y (braun), unterschiedliche Ergebnisse der Regression, links: geringe Abweichung, rechts: große Abweichungen.

Die Methode zur Bestimmung der Fassadenebene soll nicht davon abhängen, in welchem Winkel eine Fassade im übergeordneten Koordinatensystem liegt. Wünschenswert ist eine Methode zur Bestimmung der Ausgleichsgerade, die vom Koordinatensystem unabhängig ist. Da die lineare Einfachregression von der Wahl des Koordinatensystems abhängig ist, wird sie für die Bestimmung der projizierten Fassadenebenen nicht verwendet.

3.2 Regression nach Pearson

Die Regression nach Person ist eine weitere Methode zur Beschreibung einer Ausgleichsgerade durch die Punktewolke. Diese Methode ist unabhängig von der Wahl des Koordinatensystems, das heißt unabhängig von der Lage der Punktwolke in der 2D-Ebene.

Um diese Methode zu beschreiben, sind folgenden Begriffe aus der Statistik notwendig: Mittelwert, Schwerpunkt, Varianz, Streuung, Kovarianz und Kovarianzmatrix. Diese werden nachfolgend definiert.

Die Mittelwert einer Menge von reellen Zahlen $a = \{ a_i \in \mathbb{R},\ i = 1, \dots, n\}$ ist definiert als:

$$\bar{a} \coloneqq \frac{1}{n} \sum_{i=1}^{n} a_i \,.$$

Der Schwerpunkt einer Punktwolke $p = \{ p_i = (x_i, y_i) \in \mathbb{R}^2,\ i = 1, \dots, n\}$ wird definiert als zweidimensionaler Vektor der Mittelwerte der einzelnen Koordinaten:

$$\bar{p} \coloneqq (\bar{x}, \bar{y}) \ \in \mathbb{R}^2 \,.$$

Der Varianz σ^2 einer Menge von reellen Zahlen $a = \{ a_i \in \mathbb{R},\ i = 1, \dots, n\}$ ist definiert als:

$$\sigma^2 \coloneqq \frac{1}{n} \sum_{i=1}^{n} (a_i - \bar{a})^2 .$$

Die Standardabweichung σ (in dieser Arbeit auch kurz Streuung genannt) einer Menge von reellen Zahlen $a = \{ a_i \in \mathbb{R},\ i = 1, \dots, n\}$ berechnet sich als Quadratwurzel aus der Varianz. In dieser Arbeit wird zur Vereinfachung der Notation keine Unterscheidung zwischen der Standardabweichung σ einer Zufallsvariable und dem Schätzwert der Standardabweichung s_0 auf Basis einer Stichprobe [Bahrenberg (1990), S. 110] gemacht.

Die Kovarianz für zwei Mengen reeller Zahlen mit gleicher Anzahl von Elementen $a = \{ a_i \in \mathbb{R}, i = 1, \dots, n\},\ b = \{b_i \in \mathbb{R},\ i = 1, \dots, n\}$ ist eine reelle Zahl und ist definiert als:

$$\mathrm{cov}(a, b) \coloneqq \frac{1}{n} \sum_{i=1}^{n} (a_i - \bar{a})\big(b_i - \bar{b}\big).$$

Die Kovarianzmatrix für eine 2D-Punktwolke $p = \{ p_i = (x_i,\ y_i) \in \mathbb{R}^2, i = 1, \dots, n\}$ ist eine 2x2-Matrix:

$$M_{\mathrm{cov}}(p) \coloneqq \begin{pmatrix} \mathrm{cov}(x,x) & \mathrm{cov}(x,y) \\ \mathrm{cov}(x,y) & \mathrm{cov}(y,y) \end{pmatrix}.$$

Jeder Eintrag der Kovarianzmatrix ist die Kovarianz von entsprechenden Paaren von Koordinaten.

Pearson verwendet zur Beschreibung einer Ausgleichsgerade durch die Punktewolke die Begriffe: Eigenwerte und Eigenvektoren. Diese sollen hier kurz erläutert werden. Für die Kovarianzmatrix gibt es zwei sogenannte Eigenwerte und zwei Eigenvektoren. Eigenwert und Eigenvektor bilden jeweils ein Paar. Der Eigenwert ist eine reelle Zahl und der Eigenvektor ein zweidimensionaler Vektor. Im Fall einer 2x2 Matrix können die Eigenwerte als Wurzeln

einer quadratischen Gleichung und die Eigenvektoren durch Lösung eines Gleichungssystems mit zwei Gleichungen und zwei Unbekannten bestimmt werden [Gellert et al. (1986), S. 412ff]. Die Eigenvektoren werden mit $v_{\min}$ und $v_{\max}$ und die entsprechenden Eigenwerte mit $\lambda_{\min}$ und $\lambda_{\max}$ bezeichnet.

Die Eigenwerte und Eigenvektoren beschreiben wichtige Eigenschaften der Punktwolke. Sie beschreiben die Richtung der kleinsten und der größten Streuung der Punktwolke (Abbildung 21). Die beiden Vektoren $v_{\min}$ und $v_{\max}$ stehen senkrecht aufeinander. Die Eigenwerte stellen die kleinste und die größte Varianz, also die Quadrate der kleinsten und größten Streuung, der Punktwolke dar. Für den kleineren Eigenwert gilt also: $\lambda_{\min} = \sigma_{\min}^2$ und für den größeren: $\lambda_{\max} = \sigma_{\max}^2$. Die kleinste Streuung $\sigma_{\min}$ und die größte Streuung $\sigma_{\max}$ ergeben sich als Quadratwurzel aus den Eigenwerten. Das Verhältnis von $\sigma_{\min}$ zu $\sigma_{\max}$ wird im Folgenden als Koeffizient ϱ bezeichnet:

$$\varrho := \frac{\sigma_{\min}}{\sigma_{\max}}.$$

Die minimale und maximale Streuung ändern sich nicht, wenn die Punktwolke $p_i = (x_i, y_i)$, $i = 1, \dots, n$ verschoben oder gedreht wird, da die Richtungen der minimalen und maximalen Streuung sich mit drehen.

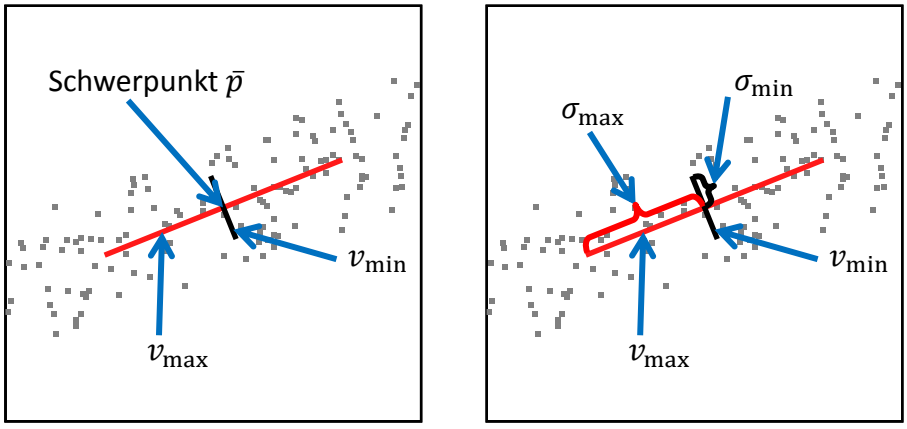

Abbildung 21: Vektoren $v_{\min}$ (schwarz) und $v_{\max}$ (rot) und Schwerpunkt $\bar{p}$, die Vektoren $v_{\min}$ und $v_{\max}$ werden mit der entsprechenden Standardabweichung skaliert.

Die Richtung der Ausgleichsgerade nach Pearson ist gegeben durch die Richtung des Eigenvektors $v_{\max}$. Um die Ausgleichsgerade vollständig zu definieren, wird der Schwerpunkt $\bar{p}$ der Punktwolke verwendet. Dieser liegt auf der Ausgleichsgerade. Die Ausgleichsgerade ist somit definiert als (Abbildung 22, links):

$$r \in \mathbb{R}^2: \quad r = \bar{p} + \lambda \cdot v_{\max}, \lambda \in \mathbb{R}.$$

Für den Fall der auf die XY-Ebene projizierten Punkte $P_i = (x_i, y_i, z_i)$ die zur Fassadenebenen gehören, approximiert die Ausgleichsgerade nach Pearson die projizierte Fassadenebe-

ne. Die Richtung der Ausgleichsgerade wird weiter im Folgenden als Richtung der Fassade bezeichnet. Die Lage der Fassade wird durch den Schwerpunkt bestimmt. Zusammen stellen die Richtung und die Lage der Fassade die Orientierung der Fassade dar.

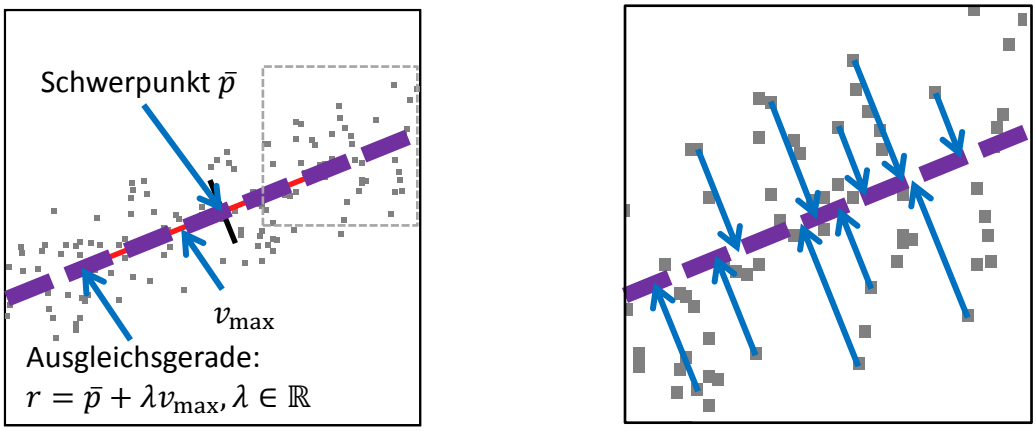

Abbildung 22: links: Die Ausgleichsgerade (violett, gestrichen) mit der Regression nach Pearson durch den Schwerpunkt entlang $v_{\max}$. rechts: Bei der Regression nach Pearson wird die Summe der Quadrate der senkrechten Abstände zur Ausgleichsgerade minimiert.

Die Richtung der Ausgleichsgerade kann durch den Winkel zur x-Achse beschrieben werden. Hierbei ist der Winkel mit einem Vorzeichen versehen. Er liegt im Intervall $[-90°, 90°]$.

An dieser Stelle ist anzumerken, dass die kleinste Streuung $\sigma_{\min}$ die mittlere Abweichung der Punkte von Ausgleichsgerade beschreibt. Diese Abweichung wird als senkrechte Projektion der Punkte auf die Ausgleichsgerade gemessen (Abbildung 22, rechts). Je kleiner der Eigenwert $\lambda_{\min}$ desto näher liegen die Punkte an der Ausgleichsgerade.

3.3 RANSAC

Eine weitere Methode zur Bestimmung einer Gerade durch eine 2D-Punktwolke ist der RANSAC-Algorithmus (random sample consensus). Dieser wurde 1981 von M. Fischler und R. Bolles vorgestellt [Fischler et al., (1981)].

Im Gegensatz zu den oben beschriebenen Methoden verwendet der RANSAC-Algorithmus zur Berechnung der bestpassenden Gerade nicht die ganze Punktwolke. Bei dem RANSAC-Algorithmus werden zufällig zwei Testpunkte aus der Punktwolke ausgewählt und die Gerade durch diese berechnet (Abbildung 23, links). Danach wird bestimmt, wie viele der Punkte aus der Punktewolke einen Abstand zur Gerade haben, der kleiner als ist ein festgelegter Schwellenwert. Dieser Schritt wird für weitere Testpunktpaare wiederholt (Abbildung 23, Mitte). Das Testpunktpaar mit der größten Anzahl von Punkte innerhalb der Toleranz definiert die am besten passende Gerade (Abbildung 23, rechts).

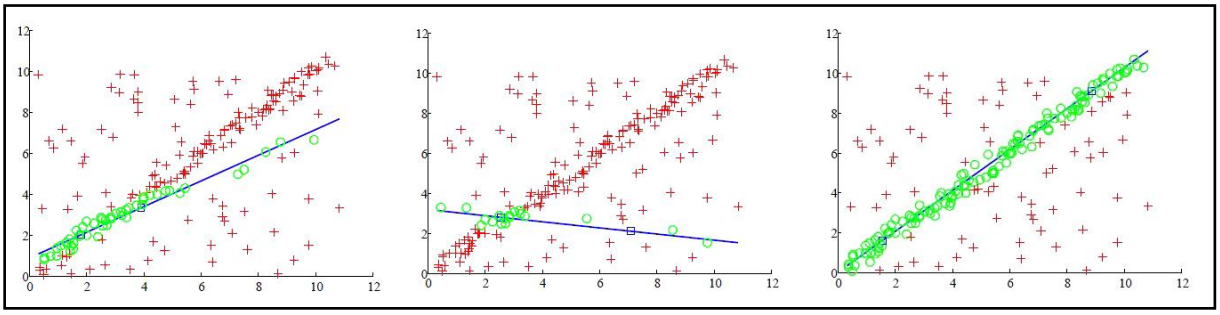

Abbildung 23: Bestimmung einer Ausgleichsgeraden mit RANSAC. Quelle: Laible (2008).

Dadurch weist RANSAC einen Vorteil gegenüber der Regression auf Basis der gesamten Punktwolke auf, wie bei der linearen Einfachregression oder der Regression nach Pearson. Er ist unempfindlich gegenüber groben Fehlern und Ausreißern [Derpanis (2010), S. 1ff]. In Abbildung 24 (links) ist zu sehen, dass bei der Regression nach Pearson ein einzelner Ausreißer, die Ausgleichsgerade stark beeinflusst. Die Regression mit RANSAC dagegen zeigt, dass der Ausreißer keinen Einfluss auf die Ausgleichsgerade hat (Abbildung 24, rechts)

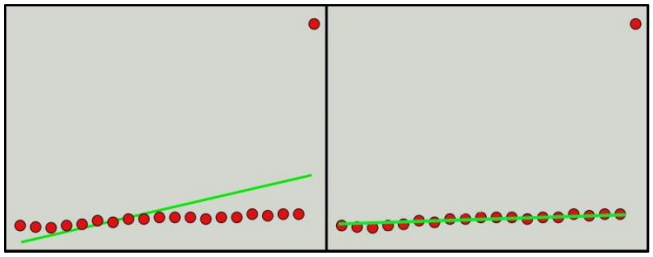

Abbildung 24: links: Bestimmung einer Ausgleichsgerade mit der Regression nach Pearson; rechts: Bestimmung einer Ausgleichsgerade mit RANSAC., Wikipedia Seite „RANSAC-Algorithmus“, eigene Bearbeitung.

Es gibt verschiedene Varianten des RANSAC-Algorithmus. Ein Nachteil wird in der hohen Berechnungszeit gesehen [Dorninger et al. (2010), S. 193]. Vor allem ist die Implementierung komplexer als bei der Regression nach Pearson. Aus diesem Grund wird in dieser Arbeit der Regression nach Pearson verwendet.

4 Fassadenextraktion – lokaler Ansatz

4.1 Lokale Regression als Grundidee

Die oben beschriebenen Methoden der Regression sind anwendbar für eine Punktwolke, deren Punkte annährend auf einer Gerade liegen. Liegen die Punkte der Punktwolke auf zwei oder mehr Geraden sind diese Methoden nicht anwendbar. Zwar kann eine Ausgleichsgerade berechnet werden. Die Punkte der Punktwolke haben aber einen großen Abstand von der Ausgleichsgerade, so dass nicht mehr von einer Approximation gesprochen werden kann (Abbildung 25).

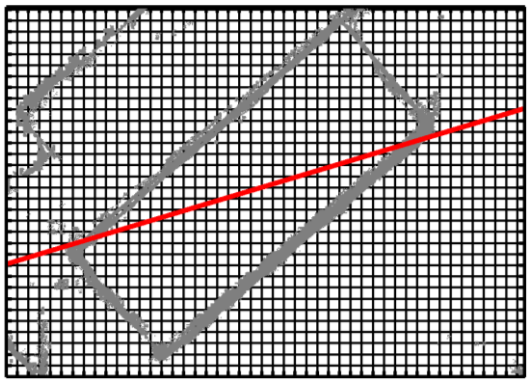

Abbildung 25: Globale lineare Regression für in grau dargestellte Fassadenpunkte, Regressionsgerade liegt nicht auf einer Fassade.

In dieser Arbeit sollen die Fassadenebenen durch lineare Regression bestimmt werden. Da diese nicht global durchzuführen ist, soll sie lokal durchgeführt werden. Die Überlegungen gehen dahin, die Punktwolke in kleine räumliche Bereiche aufzuteilen. Dieser Ansatz ist in der Literatur bekannt. Eine solche Methode der lokalen Regression haben beispielsweise Hoppe [Hoppe et al., (1992)] und Mitra [Mitra et al., (2004)] entwickelt. Bei Hoppe wird für jeden Punkt x_i aus einer Punktwolke eine Anzahl von k nächstliegenden Punkten, als k-Nachbarschaft definiert. Für diese Punkte wird Regression nach Pearson durchgeführt. Mitra entwickelt die Methode von Hoppe weiter, indem er optimale Wert von für k findet.

In dieser Arbeit wird die Punktwolke durch ein regelmäßiges Quadratgitter in quadratische Bereiche aufgeteilt. Jeder Bereich definiert ein Fenster in dem die Regression nach Pearson lokal durchgeführt wird und so die lokale Fassadenrichtung bestimmt. Dieses Fenster wird im Folgenden Regressionsfenster genannt. In Kapitel 5 wird beschrieben, wie benachbarte Bereiche mit ähnlicher Fassadenrichtung zusammengefasst und weiter verarbeitet werden.

4.2 Analyse einer Testpunktwolke

In diesem Kapitel soll die Größe des Fensters für die lokale Regression und die Lage der Punktwolke im Fenster variiert werden, um die Einflüsse auf die Ergebnisse der Regression nach Pearson zu untersuchen. Speziell werden die Streuungen und deren Richtungen analysiert. Mit diesen Untersuchungen sollen Erfahrungen gesammelt werden, wie die Parameter zur Detektion von linear verteilten Punktwolken zu definieren sind. Zu diesem Zweck wird eine zweidimensionale, in jede Richtung jeweils uniform verteilte, Punktwolke als Testpunktwolke konstruiert.

Für jedes Beispiel wird die minimale Streuung, die maximale Streuung und der Koeffizient ϱ dieser Streuungen berechnet. Weiterhin werden die Richtungen von $v_{\min}$ und $v_{\max}$ dargestellt. Dabei werden $v_{\min}$ und $v_{\max}$ mit den entsprechenden Standardabweichungen skaliert. Dadurch ist der Wert der Streuung leicht einzuschätzen. Zur Visualisierung des Koeffizienten ϱ wird die Richtung $v_{\max}$ farbkodiert dargestellt:

- $\varrho < 0{,}3$: Farbe von $v_{\max}$ rot,
- $0{,}3 \leq \varrho \leq 0{,}5$: Farbe von $v_{\max}$ violett,
- $0{,}5 < \varrho$: Farbe von $v_{\max}$ grau.

4.2.1 Einfluss der Fenstergröße

Zunächst wird der Einfluss der Fenstergröße untersucht. Dazu wird dieselbe Punktwolke mit zwei Quadraten der Kantenlänge 2,0 und 1,0 beschnitten (Abbildung 26). Ein Vergleich zeigt, dass sich die minimale Streuung, wie zu erwarten, nicht verändert. Die maximale Streuung hat sich halbiert. Dies steht in Übereinstimmung mit der Beobachtung, dass sich die Länge der Punktwolke halbiert hat.

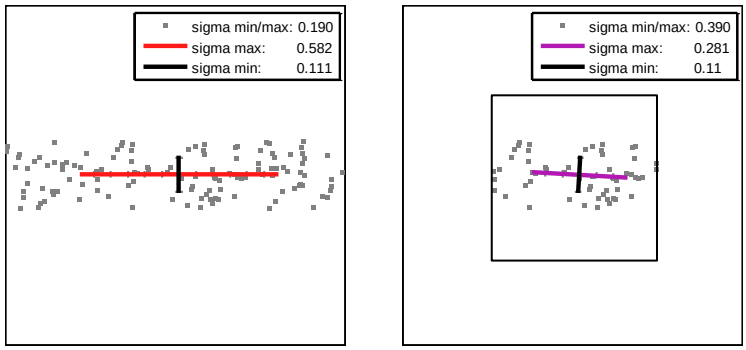

Abbildung 26: Einfluss der Fenstergröße. links: Fenstergroße 2,0, rechts: Fenstergröße 1,0.

Der Koeffizient ϱ von minimaler und maximaler Streuung hat sich entsprechend verdoppelt.

4.2.2 Einfluss der Lage der Punktwolke im Fenster

In diesem Kapitel wird analysiert, wie sich die Ergebnisse der Regression nach Pearson verändern, wenn die Punktwolke im Fenster verschoben oder verdreht wird.

Zur Analyse wird jeweils dieselbe Testpunktwolke verwendet. Diese wird gegenüber der Ausgangslage (Abbildung 27, links) um ein Viertel der Fensterbreite in Y-Richtung verschoben (Abbildung 27, rechts).

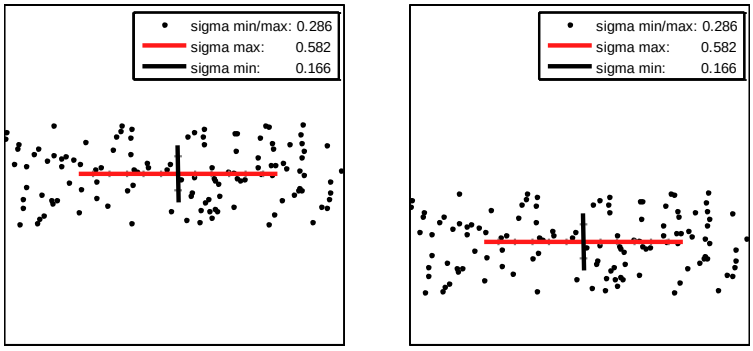

Abbildung 27: Einfluss der Lage der Punktwolke im Fenster. Links: Testpunktwolke, rechts: um ein Viertel in Y-Richtung verschobene Testpunktwolke.

Die Streuungen haben sich nicht verändert, da sich die Anzahl und die relative Lage der Punkte nicht verändert haben.

Nun wird die Punktwolke um die halbe Fensterbreite in die negative Y-Richtung verschoben (Abbildung 28). Dadurch werden manche Punkte dieser Punktwolke durch die Fensterkante abgeschnitten und werden in der lokalen Regression nicht verwendet.

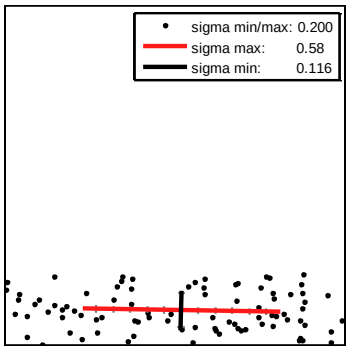

Abbildung 28: Die Testpunktwolke wurde um die halbe Fensterbreite in negative Y-Richtung verschoben.

Der Wert der minimalen Streuung hat sich in diesem Fall verringert. Das war zu erwarten, da die Punktwolke in Y-Richtung schmaler geworden ist. Der Wert für die maximale Streuung hat sich nicht verändert.

Nun wird analysiert, welchen Einfluss auf die Ergebnisse der Regression nach Pearson eine Drehung bzw. eine kombinierte Drehung und Verschiebung der Punktwolke im Fenster haben.

Es werden Beispiele konstruiert, wo die Testpunktwolke um 22,5° und 45° gedreht wird (Abbildung 29). Die minimalen Streuungen ändern sich nicht. Die maximalen Streuungen nehmen zu, da die Länge der Punktwolke im Fenster größer wird.

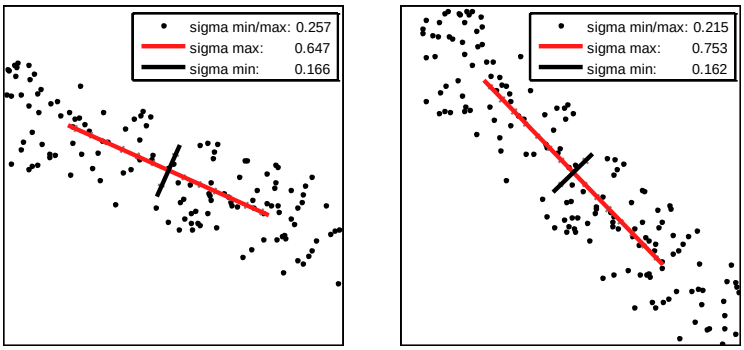

Abbildung 29: Links: Testpunktwolke um 22,5° gedreht, rechts: Testpunktwolke um 45° gedreht.

In der Abbildung 30 wird eine um 45° gedrehte Punktwolke zusätzlich in Normalenrichtung verschoben.

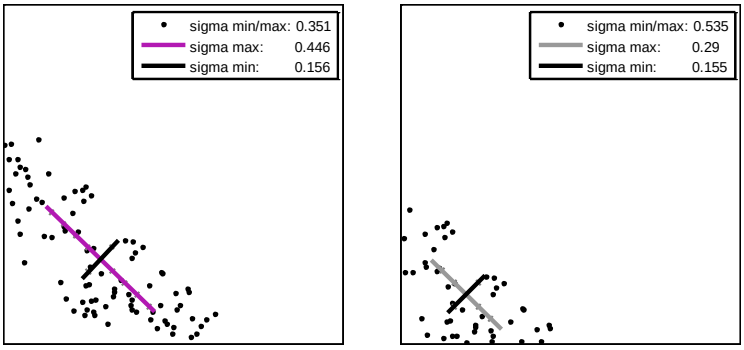

Abbildung 30: Punktwolke um 45° gedreht und zusätzlich in Normalenrichtung der Punktwolke verschoben.

Die minimale Streuung hat sich nach der Drehung und Verschiebung der Punktwolke kaum verändert. Die maximale Streuung weist im Vergleich zu der ursprünglichen Punktwolke verringerte Werte auf. Dadurch steigt der Koeffizient ϱ auf einen Wert größer 0,3. Die größte Veränderung findet im vierten Beispiel statt (Abbildung 30, rechts). Hier ist die maximale Streuung im Vergleich zur ursprünglichen um 45° gedrehte Punktwolke (Abbildung 29, rechts) um den Faktor 2,5 kleiner. Der Koeffizient ϱ steigt auf 0,54.

Diese Ergebnisse zeigen, dass die Lage der Punktwolke im Fenster einen Einfluss auf die Regression nach Pearson hat.

4.2.3 Einfluss des Rauschens

In Kapitel 3.3 wurde erwähnt, dass klassische Regressionsmethoden wie Minimum Least Square oder die Regression nach Pearson bei vorhandenem Rauschen schlechte Ergebnisse liefern können. Deshalb soll in diesem Kapitel anhand einer Testpunktwolke, der Einfluss des Rauschens auf die Regression nach Pearson untersucht werden. Das Rauschen wird simuliert, indem zu den vorhandenen Punkten eine bestimmte Anzahl vom zusätzlichen Punkten hinzugefügt wird. Die Koordinaten der zusätzlichen Punkte werden unabhängig voneinander mit einer uniformen Verteilungsfunktion hinzugefügt. Dazu wird die MatLab Funktion `rand` benutzt.

Auf der Abbildung 31 oben links werden zehn Punkte hinzugefügt. Die minimale Streuung ist um ca. 30% größer geworden. Die maximale Streuung hat sich kaum verändert. Der Koeffizient ϱ hat einen Wert von 0,36. Im Fall von 100 zusätzlich zufällig verteilten Punkten beträgt die minimale Streuung 0,393 (Abbildung 31 unten rechts). Der Koeffizient ϱ beträgt 0,655 und ist im Vergleich mit der Punktwolke ohne Rauschen ist 2,3-mal größer geworden. In allen Fällen ist der Koeffizient ϱ größer als 0,3.

Das Ergebnis zeigt, dass sich die minimale Streuung mit zunehmendem Rauschen erhöht, während die maximale Streuung sich nicht verändert.

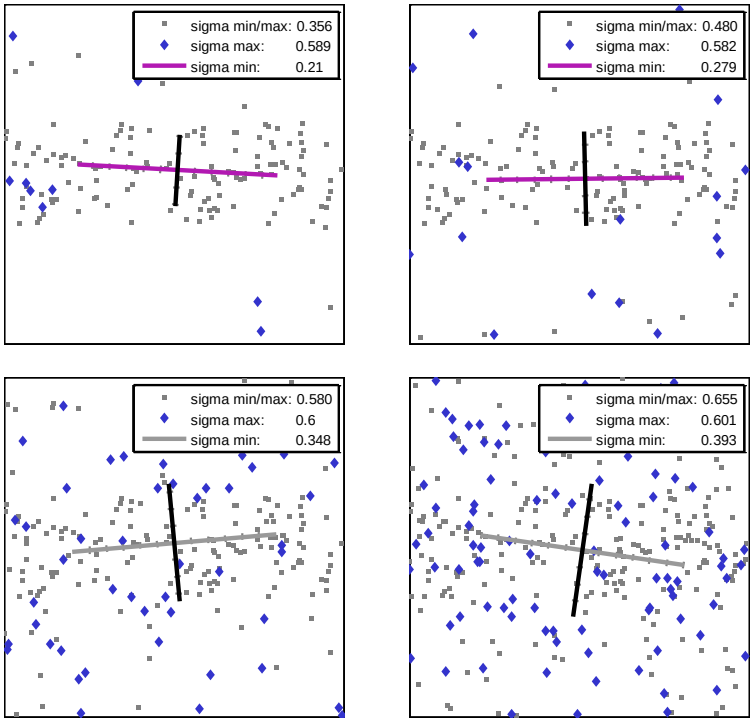

Abbildung 31: Regression nach Pearson mit unterschiedlicher Anzahl von zusätzlichen Punkten die das Rauschen visualisieren (große blaue Rauten), links oben: 10 Punkte, rechts oben: 20 Punkte, links unten: 50 Punkte, rechts unten 100 Punkte.

4.2.4 Einfluss der Form der Punktwolke im Fenster

Bisher wurde analysiert, wie sich die Regression nach Pearson für eine in etwa linear verteilte Punktwolke auswirkt. Da sich unter den projizierten 3D-Punkten nicht nur lokal linear verteilte Punkte befinden, sondern auch Punkte auf Fassadenecken, soll in diesem Abschnitt untersucht werden, wie sich die Regression nach Pearson hier auswirkt.

Auf der Abbildung 32 links wird eine parallel zu der x-Achse liegende Gebäudeecke simuliert. Auf der Abbildung 32 rechts wird diese Ecke rotiert. Die maximale Streuung ist in den beiden Fällen gleich groß und beträgt ca. 0,53. Die minimale Streuung beträgt in den beiden Fällen ca. 0,38. Der Koeffizient ϱ ist in beiden Fällen größer als 0,7.

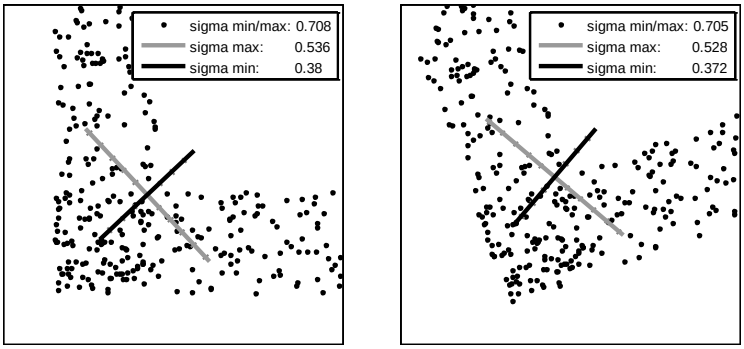

Abbildung 32: Punktwolke, die eine Gebäudeecke simuliert und Ergebnisse der Regression.

4.3 Kriterium für die Fassadenerkennung

Anhand der obigen Untersuchungen wird vorgeschlagen, dass der Koeffizient ϱ als Parameter zur Unterscheidung zwischen linear verteilten Punktwolken und Punktwolken, die nicht linear verteilt sind, verwendet wird. Er ist definiert als das Verhältnis zwischen der minimalen und der maximalen Streuung der Punktwolke. Je kleiner der Wert von ϱ ist, desto mehr liegen die Punkte auf einer Gerade.

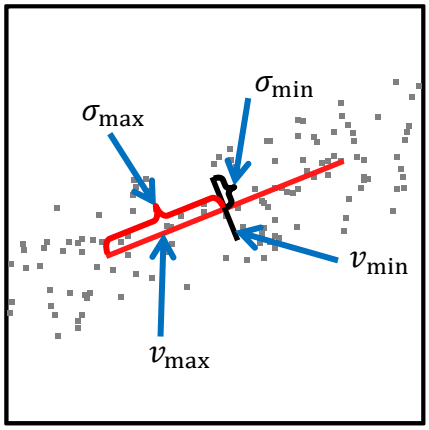

Abbildung 33: Das Verhältnis von $\sigma_{\min}$ zu $\sigma_{\max}$ ist eine Zahl zwischen 0 und 1 und wird als Koeffizient ϱ bezeichnet. Es wird visualisiert durch das Verhältnis der Längen der mit der jeweiligen Standardabweichung skalierten Vektoren $v_{\min}$ und $v_{\max}$.

Unter Verwendung eines Schwellenwertes $\varrho_{\max}$ lautet das Kriterium für die Fassadenerkennung:

Wenn für eine auf die XY-Ebene projizierte Punktwolke in einem Regressionsfenster gilt $\varrho < \varrho_{\max}$, dann gehört dieser Teil der Punktwolke zu einer Fassadenebene.

Für einen Schwellenwert von $\varrho_{\max} = 0{,}3$ wird dieses Kriterium erfüllt für die Punktwolken in den Abbildungen 26, 27, 28, 29. Das ist ein Hinweis, dass das Kriterium zur Fassadenerkennung sinnvoll gewählt ist.

In den Beispielen in Abbildung 28 und in Abbildung 30 ist zu sehen, dass für Punktwolken, die nicht durch den Mittelpunkt des Fensters gehen, zwei gegensätzliche Effekte auftreten können. Einerseits kann durch den Beschnitt des Fensters eine erhöhter Wert von ϱ auftreten, andererseits aber auch ein verringerter.

Das Kriterium für die Fassadenerkennung ist nicht erfüllt für die Punktwolke mit Rauschen (Abbildung 31). Das stellt ein Problem dar, das im Weiteren berücksichtigt werden muss.

Das Kriterium für die Fassadenerkennung wird nicht erfüllt für Punkte auf einer Fassadenekke (Abbildung 32). Dies ist erwünscht.

Dieses Kriterium für die Fassadenerkennung ist streng und es kann sein, dass es nicht erfüllt ist, obwohl der Teil der Punktwolke im Fenster zu einer Fassadenebene gehört. Das ist beispielsweise für die Punktwolke aus Abbildung 30 zu sehen. Für diese Punktwolken ist anhand der Regressionsanalyse nicht sicher feststellbar, dass die Punkte zu einer Fassade gehören. Unter Anwendung des Kriteriums zur Fassadenerkennung, würden diese Punkte nicht bei der Fassadendetektion berücksichtigt. Damit diese Punkte für die Analyse nicht verlorengehen, falls sie zu einer Fassadenebene gehören, wird das Konzept der Umgebung eingeführt. Dieses wird in dem folgenden Abschnitt erläutert.

4.4 Quadratgitter, Zelle und Umgebung

Zu den Ausgangsdaten gehört eine 3D-Punktwolke, die auf die XY-Ebene projiziert wurde. Die aus programmiertechnischen Gründen einfachste Aufteilung der Punktwolke ist die mittels eines regelmäßigen Quadratgitters. Jeder quadratischer Teil dieses Gitters wird im Folgenden als Zelle bezeichnet. Die Breite einer Zelle wird im Programm durch den Parameter l_{interval} bestimmt.

Für jede Zelle soll die Regressionsanalyse nach Pearson durchgeführt werden, um die Zugehörigkeit der Zelle zu einer Fassadenebene (vgl. Kapitel 4.3) festzustellen und die lokale Richtung der Fassade zu berechnen. Insbesondere soll untersucht werden, welche Einstellungen zur Breite des Regressionsfensters und zur Wahl des Schwellenwertes $\varrho_{\max}$ sinnvoll sind.

4.4.1 Analyse einer realen Punktwolke

In der Abbildung 34 wird ein Ausschnitt aus dem Gebäude B2 dargestellt, dass durch verschiedene Quadratgitter in Zellen der Breite 1 m, 2 m, 3 m und 4 m unterteilt wurde. Das Regressionsfenster entspricht zunächst einer Zelle. In Abbildung 34 sind die Ergebnisse der lokalen Regression dargestellt. Die Größe des Koeffizienten ϱ wird wie weiter oben durch die Farbe des Eigenvektors $v_{\max}$ kodiert.

Bei einer Fenstergröße von 1 m (Abbildung 34, oben links) sind die für die meisten Zellen Koeffizienten ϱ größer als $\varrho_{\max} = 0{,}5$ und werden durch schwarze Darstellung der Richtungen $v_{\max}$ markiert. Das bedeutet, dass Punkte aus diesen Zellen als nicht zu einer Fassade zugehörig identifiziert werden. Erst ab einer Fenstergröße von 3 m und 4 m (Abbildung 34, unten), werden alle Fassadenteile identifiziert (rote Striche). Die Gebäudeecke wird wie beabsichtigt nicht als Fassade identifiziert. In der Abbildung 34 wird die Zuordnung des Vektors $v_{\max}$ zur Zelle durch einen dünnen grauen Strich vom Zellmittelpunkt zum Schwerpunkt der lokalen Punktwolke dargestellt.

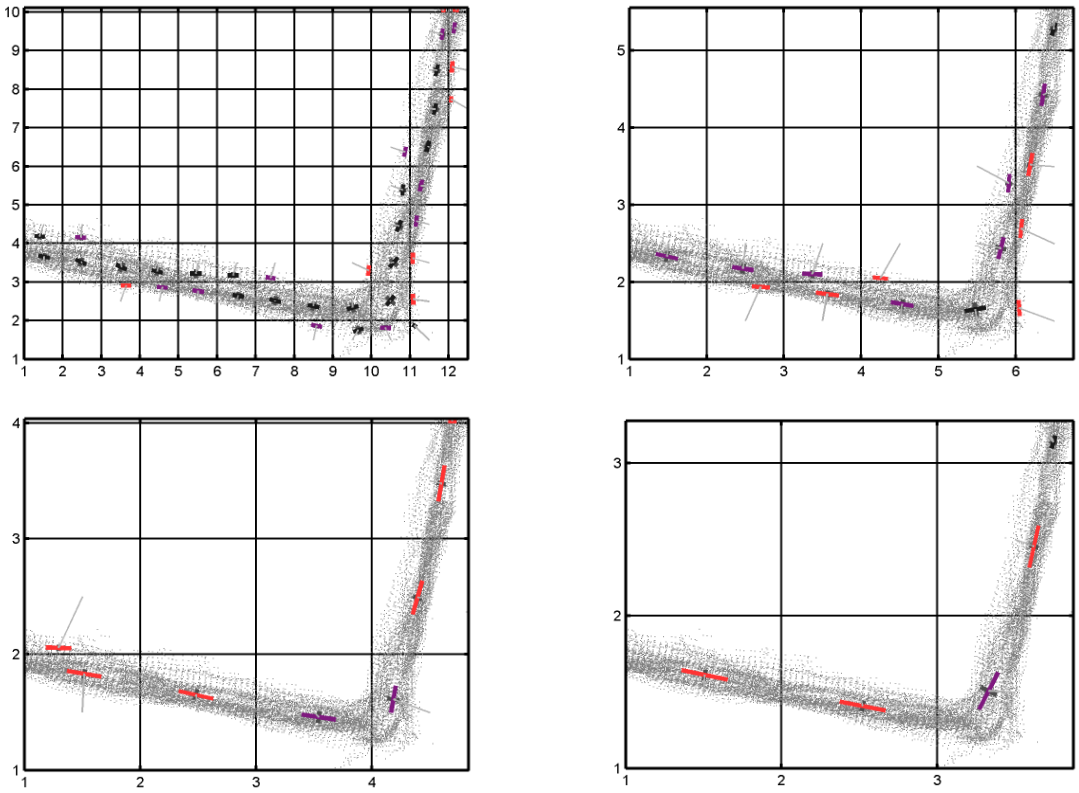

Abbildung 34: Ausschnitt aus Gebäude B2, Regression nach Pearson mit unterschiedlicher Regressionsfenstergröße, Regressionsfenster entspricht einer Zelle, Zellbreiten oben links: 1 m, oben rechts: 2 m, unten links: 3 m, unten rechts: 4 m, $v_{\max}$ rot: $\varrho < 0{,}3$, violett: $0{,}3 < \varrho < 0{,}5$, schwarz $0{,}5 < \varrho$.

4.4.2 Definition der Umgebung

Für die Regression wird vorgeschlagen, das Regressionsfenster nicht auf eine einzelne Zelle zu beschränken, sondern auf Basis der Zelle und ihrer benachbarten Zellen zu definieren. Das Regressionsfenster enthält damit Punkte aus der Zelle und ihrer Umgebung (Abbildung 35). Die Größe der Umgebung kann dabei unterschiedlich definiert werden. Beispielsweise können zu der Umgebung nur die direkt benachbarte Zellen (Abbildung 35) gehören, so dass die Umgebung aus insgesamt 9 Zellen besteht. Diese Umgebung wird im Folgenden „3x3-Umgebung“ der Zelle genannt, da sie insgesamt $3^2 = 9$ Zellen enthält.

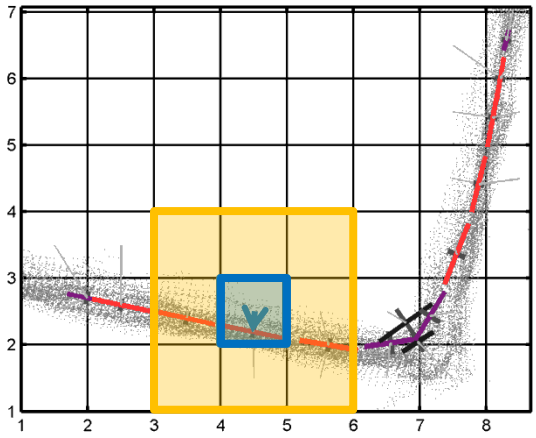

Abbildung 35: Konzept der Umgebung, Fenster für Regression besteht aus einer Zelle mit allen maximal 8 benachbarten Zellen.

Erweitert man die ursprüngliche Zelle um zwei Reihen von Nachbarzellen, wird die Umgebung aus insgesamt 25 Zellen bestehen, dies entspricht einer 5x5-Umgebung der Zelle. Im Programmcode wird die Wahl der Breite der Umgebung über einen Parameter $n_{\text{nbhd\_regression}}$ gesteuert, so dass verschiedenen Varianten ausprobiert werden können. Die Werte 1 bzw. 2 entsprechen der 3x3-Umgebung bzw. 5x5-Umgebung. Im Besonderen kann dieser Parameter mit 0 auch so gewählt werden, dass die Umgebung genau die Ausgangszelle enthält und keine weiteren benachbarten Zellen. Wenn eine Zelle am Rand liegt, enthält die 3x3-Umgebung entsprechend weniger Zellen.

Umgebungen von benachbarten Punkten überlappen sich (Abbildung 36).

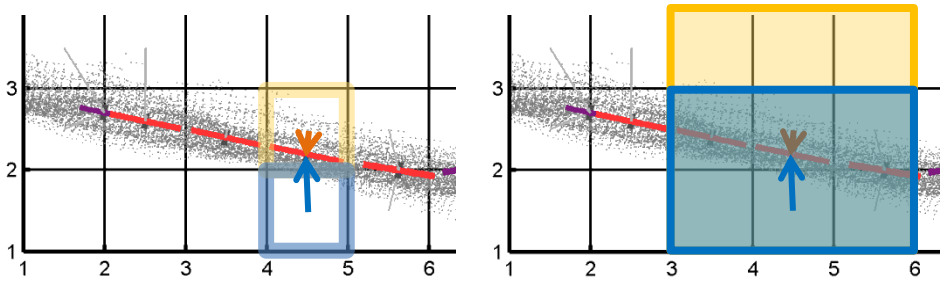

Abbildung 36: Konzept der Umgebung, links: gelb und blau markierte Zellen sind benachbart, rechts: die Regressionsfenster für die gelb und blau markierten Zellen überlappen sich, jedes der beiden Fenster enthält dieselbe Teilpunktwolke, Pfeile zeigen von Zellmittelpunkt auf Regressionsgerade.

4.4.3 Beispiel für Regression mit Umgebung

In Abbildung 37 wird ein Ausschnitt aus Gebäude B4 dargestellt. Dieser zeichnet sich durch eine kurze Wand aus. Es soll der Einfluss der Umgebung auf die lokale Regression untersucht werden. In der oberen Reihe werden Ergebnisse der lokalen Regression für verschie-

dene Regressionsfensterbreiten von 1,5 m, 3 m, 4,5 m und 6 m ohne Verwendung einer Umgebung abgebildet. In der mittleren Reihe werden Ergebnisse der lokalen Regression bei denselben Regressionsfensterbreiten aber unter Verwendung der 3x3-Umgebung dargestellt. Die Größe der Zelle wird hier entsprechend um den Faktor 3 verkleinert und beträgt 0,5 m, 1 m, 1,5 m und 2 m. Beispielsweise hat das Bild in der oberen Reihe (links) eine Zellenbreite, und somit ein Regressionsfensterbreite, von 1,5 m. Das entsprechende linke Bild aus der mittleren und unteren Reihe wird in Zellen mit 0,5 m Breite aufgeteilt, die zusammen mit der 3x3 Umgebung ein Regressionsfenster von 1,5 m Breite bilden.

Vergleicht man die Abbildungen aus der oberen Reihe mit denen aus der mittleren Reihe sieht man deutlich, dass in der mittleren Reihe mehr Richtungen $v_{\max}$ dargestellt sind. Dies illustriert, dass die Regressionsfenster aus der oberen Reihe auch für die Regression unter Verwendung der 3x3-Umgebung genutzt werden.

Zweitens ist zu bemerken, dass sich die Anzahl der Eigenvektoren mit einem kleinen Koeffizienten ϱ erhöht hat. Das hat folgende Ursache: Bei der Regression unter Verwendung einer 3x3-Umgebung überlappen sich die einzelnen Regressionsfenster und bedecken die Punktwolke dichter, als bei der Regression ohne Umgebung. Die Regression wird also für dieselben Punkte mehrmals durchgeführt. Das hat den Vorteil, dass Punkte, die in einem Regressionsfenster ungünstig liegen, beispielsweise wenn sich die Fassade nur zu einem kleinen Teil im Regressionsfenster befindet, in einem benachbarten Regressionsfenster günstig liegen können. Dadurch werden die ungünstig liegenden Punkte nicht verlorengehen und durch das benachbarte Regressionsfenster als Fassadenpunkte identifiziert.

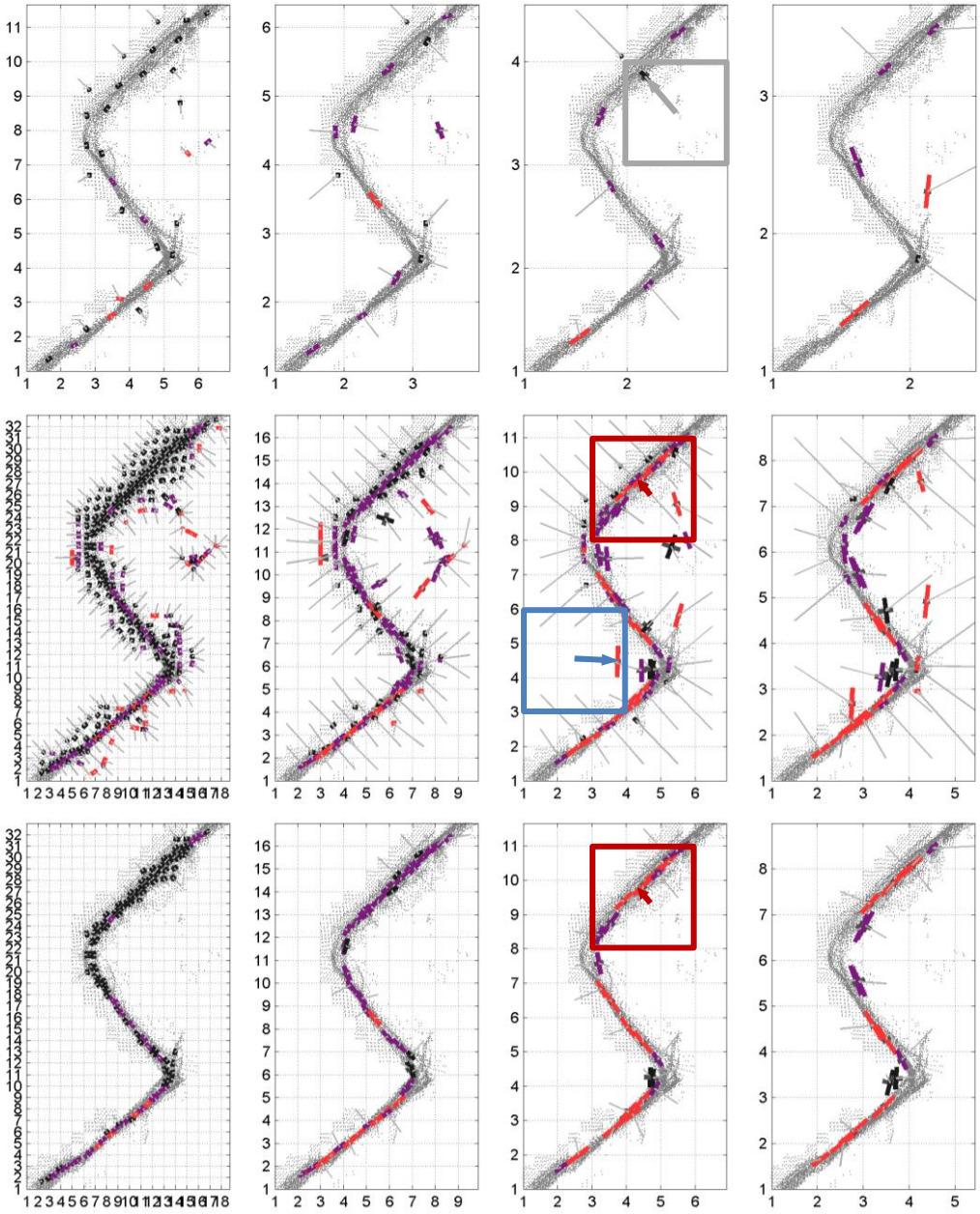

Abbildung 37: oben: lokale Regression ohne Umgebung bei Zellenbreite 1,5 m, 3 m, 4,5 m, 6 m, Mitte: lokale Regression mit 3x3-Umgebung und Zellenbreite 0,5 m, 1 m, 1,5 m, 2 m , unten: lokale Regression mit 3x3-Umgebung und Zellenbreite: 0,5 m, 1 m, 1,5 m, 2 m und Mindestanzahl von Punkten in der Zelle. Die Farben entsprechen dem Koeffizienten ϱ wie in Abbildung 34.

Als Beispiel soll die Regression mit einer Fensterbreite von 4,5 m betrachtet werden (Abbildung 37, dritte Spalte von links). In der oberen Reihe beobachtet man, dass die nördliche Fassade ungünstig in den Regressionsfenstern liegt. Die Koeffizienten ϱ sind beispielsweise in der Zelle [2,3] größer als 0,5 und die entsprechende Richtung $v_{\max}$ ist schwarz markiert (Abbildung 37, grau umrandetes Regressionsfenster). Die Regression unter Verwendung der 3x3-Umgebung für die Zelle [4,9] zeigt dagegen einen Koeffizienten ϱ, der kleiner als 0,5 ist, und detektiert somit die nördliche Fassade (Abbildung 37, rot umrandetes Re-

gressionsfenster). Generell werden wie gewünscht bei Verwendung der 3x3-Umgebung mehr Fassaden extrahiert als ohne.

4.4.4 Selektive Regression mit Umgebung

Allerdings ist bei Verwendung der Regression mit 3x3-Umgebung zu beobachten, dass manche Zellen als Fassadenzellen identifiziert werden, obwohl sie keine Punkte enthalten (Abbildung 37, blau umrandetes Regressionsfenster). Dieser Fehler tritt auf, wenn eine Zelle keine Punkte enthält, aber die Zellen aus der Umgebung dieser Zelle Punkte enthalten. Um diesen Fehler zu beseitigen, wird ein Parameter $n_{\text{min\_points}}$ eingeführt. Ist die Anzahl von Punkten in der Zelle kleiner als der gewählte Parameter $n_{\text{min\_points}}$, wird die Regression für diese Zelle und deren Umgebung nicht durchgeführt. Dieser Parameter hat standardmäßig den Wert 10. Das Ergebnis der Regression nach der Einführung von dem Parameter wird in der unteren Reihe der Abbildung 37 dargestellt.

Es ist zu sehen, dass mit dieser Bedingung weniger Regressionen durchgeführt werden als ohne diese Bedingung (vgl. mittlere und untere Zeilen von Abbildung 37). Insbesondere ist zu sehen, dass der Abstand von Zellenmittelpunkt zum Schwerpunkt der Ausgleichsgerade (dargestellt durch dünne graue Pfeile) gering ist.

Das Regressionsfenster darf nicht zu klein und nicht zu groß gewählt werden. In der Abbildung 37 (linke Spalte) ist zu sehen, dass ein Regressionsfenster von 1,5 m zu klein ist um alle Fassaden zu identifizieren. Nur die südlichen Fassaden werden für einen Schwellenwert $\varrho_{\max} = 0{,}5$ detektiert. Ein Regressionsfenster mit einer großen Breite von 6 m (Abbildung 37, rechte Spalte) ist zur Identifizierung der mittleren Fassade schlecht geeignet. Man vergleiche hierzu die mittlere und untere Reihe, das erste Bild von rechts, Zelle [2, 5]. Im Fall von horizontal kurzen Fassaden beinhaltet ein großes Regressionsfenster sowohl die Fassadenpunkte als auch die Eckpunkte. In diesen Fällen ist der Koeffizient ϱ größer als $\varrho_{\max} = 0{,}3$ und die Punkte aus diesen Zellen werden nicht als Fassadenpunkte identifiziert.

Zusammenfassend lässt sich feststellen, dass für das untersuchte Beispiel (Abbildung 37) die Regression mit einer Zellenbreite von ca. 1,5 m mit einer 3x3-Umgebung für das Regressionsfenster und einer Bedingung für die Mindestanzahl von Punkten in der Zelle zu guten Ergebnissen führt.

Die Aufteilung der Punktwolke in quadratische Zellen wird in der Funktion `make_cloud_in_cells` (Anhang, S. [5]) realisiert. Dazu werden zunächst die minimalen und maximalen Werte der X- und Y-Koordinaten, sowie die Anzahl der Zellen in der X- und Y-Richtung bestimmt. Anschließend, wird für jeden Punkt der Index der entsprechenden Zel-

le in X- und Y-Richtung berechnet. Danach wird der Punkt zu den Punkten dieser Zelle hinzugefügt. Die Regression nach Pearson wird in der Funktion `eigenvectors_full_cells` (Anhang, S. [9]) durchgeführt (vgl. Kapitel 4.6).

4.4.5 Testbeispiel für Gebäudeecke B2

In diesem Abschnitt werden quantitative Ergebnisse der lokalen Regressionsanalyse für ein Beispiel dargestellt.

In der Abbildung 38 wird eine Ecke des Gebäudes B2 untersucht. Punkte, die diese Ecke bilden, werden durch ein regelmäßiges Gitter in Zellen mit Breite 1,6 m aufgeteilt. Für jede Zelle und deren 3x3-Umgebung wird die Regression nach Pearson durchgeführt. In Abbildung 38 (links) wird die minimale Standardabweichung $\sigma_{\min}$ für jede Zelle dargestellt. Die minimale Standardabweichung ist mit 0,25 m in etwa konstant. Die Überlegungen aus Kapitel 3.2 zeigen, dass dieser Wert auch unabhängig von der Breite des Regressionsfensters ist, so lange das Regressionsfenster nur Teile derselben Fassade enthält.

Ein Vergleich mit Werten der Standardabweichung aus der Masterarbeit [Konecny (2011), S. 76] zeigt dort deutlich geringere Werte aus dem Bereich von 0,12 m bis 0,16 m. Hierbei ist anzumerken, dass die Daten dort sich auf Punktwolken beziehen, die aus einem digitalen Oberflächenmodell stammen, das aus Aufnahmen erstellt wurde, die eine gemeinsame Perspektive haben. In dieser Arbeit wird die vereinigte Punktwolke aus sieben digitalen Oberflächenmodellen verwendet (siehe Kapitel 2.3). Dadurch kann die Standardabweichung größer werden.

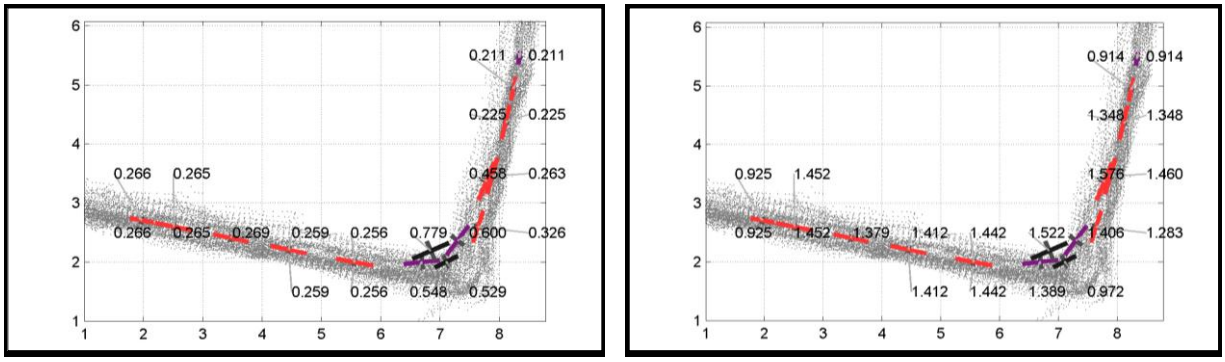

Abbildung 38: Werte der Standardabweichung $\sigma_{\min}$ (links) und $\sigma_{\max}$ (rechts) für Quadratgitter mit Kantenlänge 1,6 m und 3x3-Umgebung.

Für ausgewählte Zellen ergeben sich folgende Werte des Koeffizienten ϱ.

Zelle [2,2]	ϱ = 0,19 = 0,27/1,45
Zelle [7,4]	ϱ = 0,17 = 0,23/1,35
Zelle [7,1]	ϱ = 0,55 = 0,53/0,97

Tabelle 1: Werte für Koeffizient ϱ für ausgewählte Zellen.

Wird der Schwellenwert von $\varrho_{\max}$ auf 0,3 eingestellt, werden auf Grund von der Bedingung

$$\frac{\sigma_{\min}}{\sigma_{\max}} < \varrho_{\max}$$

die Zellen [2,2] und [7,4] korrekt als Fassadenzellen interpretiert. Die Zelle [7,1] gehört zu einer Ecke und wird korrekt nicht als Fassadenzelle detektiert.

In der vorliegenden Arbeit ist die Wahl der Parameter $\varrho_{\max}$ und der Fenstergröße ein empirisch geprüfter Prozess. Gute Ergebnisse werden bei dem Wert $\varrho_{\max} = 0{,}3$ und der Breite des Regressionsfensters von 4,8 m, d.h. für den Parameter der Zellenbreite l_{interval} gleich 1,6 m und die Breite der Nachbarschaft von $n_{\text{nbhd\_regression}} = 1$ erzielt.

4.5 Z-Filter zur Entfernung von Nicht-Fassaden-Objekten

In Kapitel 4.2.3 und 4.3 wurde gezeigt, dass das Rauschen einen negativen Einfluss auf lineare Regression hat. Für die Extraktion von Fassadenrichtungen haben die Dach- und Bodenpunkte einen ähnlichen Einfluss wie das Rauschen, da sie außerhalb der Fassade liegen.

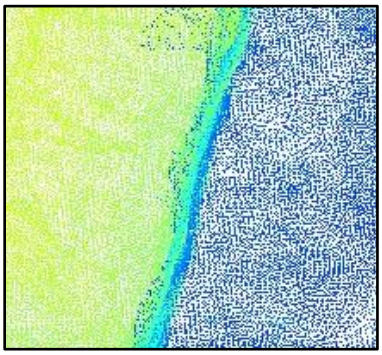

Abbildung 39: Ausschnitt aus 3D-Punktwolke in Umgebung einer Fassade (farbkodiert nach Höhe).

Das Ziel der Filterung besteht darin, die Boden- und Dachpunkte, sowie alle weiteren unerwünschten Objekte aus der 3D-Punktwolke zu entfernen, so dass nur die Fassadenpunkte übrigbleiben. Für diese Filterung soll die Z-Koordinate der Punkte verwendet werden.

Eine einfache Methode, könnte folgendermaßen aussehen. Bei bekannter und konstanter Geländehöhe h_0 und maximaler Höhe der Vegetation l_{veg} können die Punkte für Objekte auf dem Boden und auf der Vegetation mit folgender Bedingung entfernt werden:

$$z > h_0 + l_{\text{veg}}\,.$$

Dachpunkte könnten bei bekannter Höhe der oberen Fassadenkante des kleinsten Gebäudes l_{min} mit dieser Bedingung entfernt werden:

$$h_0 + l_{\text{min}} > z.$$

Dabei muss gewährleistet sein, dass die Höhe der oberen Fassadenkante des kleinsten Gebäudes höher ist als die der Vegetation: $l_{\text{veg}} < l_{\text{min}}$, da sonst jeder Punkt beide Bedingungen erfüllen würde.

Für diese einfache Methode müssen die Werte für h_0, l_{veg} und l_{min} bekannt sein und manuell eingegeben werden, was die Anwendung des Algorithmus erschwert. Haben die Gebäude aber unterschiedliche Höhen, was typischerweise vorkommt, werden mit dieser Methode niedrigere Gebäude als $h_0 + l_{\text{veg}}$ teilweise oder vollständig entfernt. Als weiterer Nachteil werden von den Gebäuden, die höher als $h_0 + l_{\text{min}}$ sind, Fassadenpunkte unterhalb von l_{min} entfernt.

Diese Methode ist nicht optimal und im Folgenden wird eine Lösung der lokalen Filterung vorgestellt. Wünschenswert ist die Entwicklung eines Filters, der keine absoluten Werte benötigt und somit keine manuelle Steuerung erfordert und außerdem möglichst alle Punkte auf den Fassaden detektiert.

4.5.1 Lokaler Z-Filter

Der Ausgangspunkt der Filterung ist, die Projektion der Punkte auf die XY-Ebene und die Unterteilung der projizierten Punkte durch ein Quadratgitter in der XY-Ebene. Die Kantenlänge dieses Quadratgitters kann anders gewählt werden als die Kantenlänge des Quadratgitters für die Regression (siehe Abschnitt 4.4) und wird von dem Parameter $l_{\text{z\_filter}}$ gesteuert. Für jede Zelle des Quadratgitters wird die 3x3-Umgebung bestimmt. Für die Punkte, deren Projektion in dieser Umgebung liegt, wird das Histogramm der Z-Werte erstellt. Auf der x-Achse des Histogramms werden die Z-Höhen, die in 10 Klassen gruppiert sind, abgelegt. Auf der y-Achse wird die Anzahl von Punkten abgelegt (Abbildung 42). Einfache geometrische Überlegungen zeigen, dass bei Zellen, die Fassadenpunkte enthalten ein typisches Muster für das Histogramm zu erwarten ist. Dies wird in der Abbildung 40 illustriert.

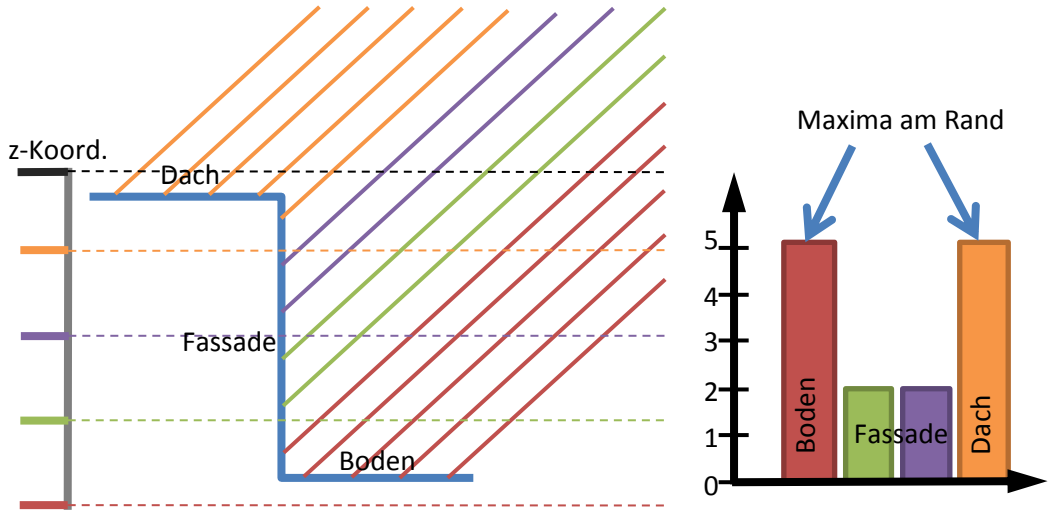

Abbildung 40: links: schematische Abbildung eines Gebäudeteils mit Boden, Fassade und Dach bei einer Schrägaufnahme, rechts: Histogramm der Z-Werte für diesen Gebäudeteil.

Dieses Muster kann man ab einer gewissen Größe der Umgebung erwarten. Für ausreichend große Umgebungen wird der Anteil der Punkte auf dem Dach und dem Boden ähnlich groß oder größer sein, als der Punkte auf der Fassade. Für diesen Fall ist zu vermuten, dass es zwei Histogrammklassen gibt, jeweils an dem unteren und oberen Rand des Histogramms, deren Punktanzahl größer als die der benachbarten Klassen ist. In der Klasse am linken Rand liegen die Punkte aus dem Bodenbereich. Diese haben in etwa dieselbe Z-Koordinate und werden deshalb in derselben bzw. benachbarten Histogrammklasse liegen. Ähnliches gilt für die Punkte im Bereich des Daches. Somit besteht das typische Muster für Histogramme von Zellen im Fassadenbereich darin, dass das Histogramm zwei Maxima aufweist, die sich jeweils am unteren und oberen Rand des Histogramms befinden. Eine Verteilung mit 2 Maxima wird bimodal genannt [Bahrenberg (1990), S. 37].

Der untere Schwellenwert $z_{\min}$ wird pro Zelle als Mittelwert der Grenzen der Histogrammklasse des unteren Maximums bestimmt. Ähnlich wird der obere Schwellenwert $z_{\max}$ bestimmt. Die Entfernung zwischen den Maxima des Histogramms ist abhängig von der Fassadenhöhe. Es wird gefordert, dass diese mindestens 3 m beträgt, damit alle Nicht-Fassaden-Objekte mit einer Höhe unter 3 m entfernt werden. Diese Entfernung wird als Parameter $h_{\text{z\_filter}}$ eingestellt**.** Die neue Bedingung zur Filterung von Fassadenpunkte lautet nun:

1. für eine Zelle des Quadratgitters, hat das Histogramm der Z-Werte der Punkte aus der 3x3-Umgebung dieser Zelle jeweils ein Maximum am linken und rechten Rand,
2. Berechnung der Schwellenwerte: $z_{\min}$, $z_{\max}$,
3. Zelle wird übersprungen, wenn: $z_{\max} - z_{\min} < h_{\text{z\_filter}}$,
4. 3D-Punkte innerhalb der Zelle mit der folgenden Eigenschaft werden, als Fassadenpunkte verwendet: $z_{\min} < \mathrm{z} < z_{\max}$.

Diese Überlegungen gelten für Flachdächer und möglicherweise für Dächer, wie sie sich in Altbauten in der Großstadt befinden. Für Giebeldächer wird kein deutliches Maximum im Histogramm erwartet. Da in dem betrachteten Aufnahmegebiet Gebäude alle größeren Gebäude Flachdächer aufweisen, wird im Folgenden der oben beschriebenen Ansatz als ausreichend angesehen.

4.5.2 Einsatz des lokalen Z-Filters

Die Entfernung von Punkten, die nicht auf Fassaden liegen, wird durch einen Filter im MatLab in der Funktion `z_filter` (Anhang, S. [7]) implementiert.

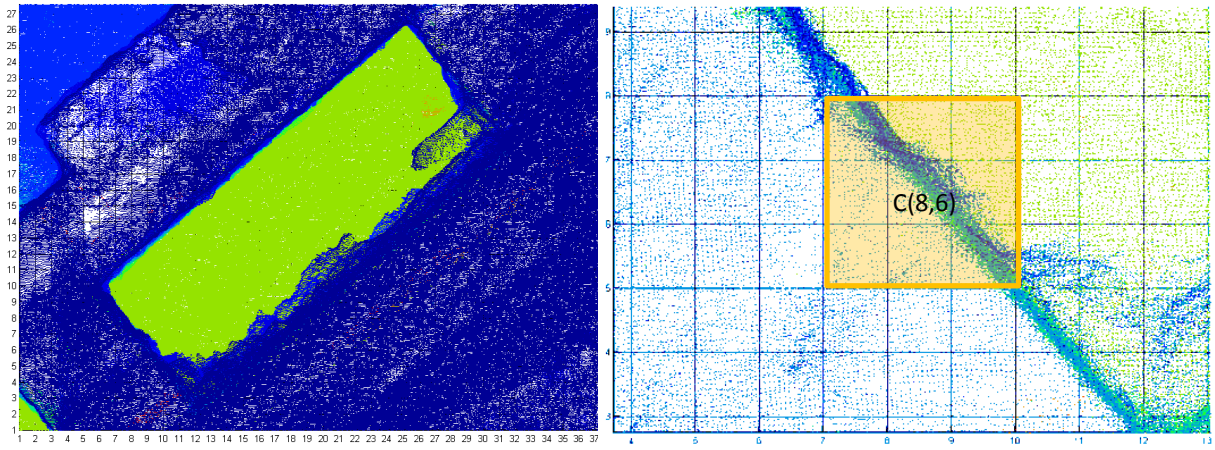

Abbildung 41: links: Gebäude B1 mit Einfärbung nach Z-Koordinate, deutlich zu sehen sind falsch zugeordnete Punkte unterhalb des Daches, Quadratgitter mit Breite 2m, rechts: Umgebung der Zelle C[8,6].

Die Funktionsweise des Filters soll am Beispiel des Gebäudes B1 demonstriert werden. Dabei wird ein Quadratgitter mit Breite 2 m und 3x3 Umgebung der Zellen verwendet. In Abbildung 41 ist das Gebäude B1 (links) und ein Ausschnitt (rechts) dargestellt. In Abbildung 42 (links, Mitte) wird das Histogramm für die Zellen [2,2] und [8,6], die auf einer Fassade liegen, dargestellt. Es treten wie erwartet Maxima an den Rändern des Histogramms auf.

Um Punkte, die fehlerhafte Daten in Gebäude B1 repräsentieren, identifizieren zu können, muss eine weitere Bedingung hinzugefügt werden. Bei den fehlerhaften Daten, handelt es sich um scheinbare Bodenpunkte direkt unterhalb des Daches. Diese Punkte erzeugen somit ebenfalls das oben beschriebene typische Muster für das Histogramm (Abbildung 42 rechts).

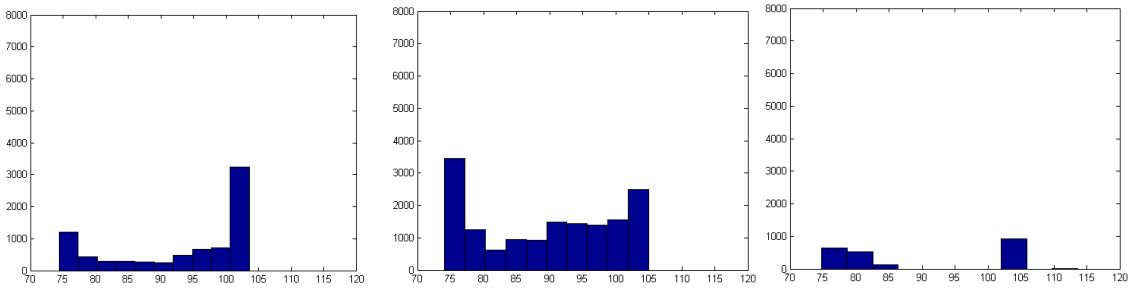

Abbildung 42: links, Mitte: typisches Muster für Histogramm der Z-Werte für Zelle C[2,2] und C[8,6], rechts: Histogramm der Umgebung der Zelle C[12,5], die fälschlicherweise Bodenpunkte unterhalb des Daches enthält.

Um diese fehlerhaften Daten zu entfernen, wird gefordert, dass die Histogrammklassen, die sich zwischen den Maxima befindenden nicht leer sind. Das entspricht der natürlichen Forderung, dass eine Fassade durchgehend über die gesamte Höhe aus Punkten enthält. Die Anzahl der nicht leeren Klassen wird im Algorithmus über den Parameter $n_{\text{z\_filter}}$ bestimmt.

Nach der ersten Implementierung dieses Filters stellte sich heraus, dass zu viele Fassadenpunkte, gelöscht werden. Es wird analysiert, welche Bedingung des implementierten Filters diese Tatsache verursacht. Es stellte sich heraus, dass das linke Maximum nicht immer in den Klassen 1-2 liegt, also das das linke Maximum nicht immer von Bodenpunkten verursacht wird. Es wird auch festgestellt, dass das rechte Maximum nicht immer in den Klassen 9-10 liegt, also nicht immer von Dachpunkten verursacht wird. In der endgültigen Version der Funktion `z_filter` wird das linke Maximum in den Klassen 1-5 gesucht und das rechte Maximum in den Klassen 6-10. Nach dieser Veränderung funktioniert der Z-Filter so, dass fast alle Punkte auf den Fassaden als Fassadenpunkte identifiziert werden (Abbildung 46). Allerdings nimmt die Anzahl der Punkte, die nicht auf Fassaden liegen, aber als Fassadenpunkte identifiziert werden, zu. Beispielsweise werden die Punkte eines über 3 m hohen Baumes, als Fassadenpunkte interpretiert.

Die optimale Breite der Zellen des Quadratgitters $l_{\text{z\_filter}}$ (vgl. Kapitel 4.5.1) wird durch Tests ermittelt und beträgt 0,5 m.

Der vorgestellte Algorithmus wird trotz dieser Schwäche verwendet, da ausreichend viele Fassadenpunkte in der gefilterten Punktwolke vorhanden sind und da erwartet wird, dass falsch detektierte Fassadenpunkte durch das Kriterium zur Fassadenerkennung bei der linearen Regression entfernt werden.

Abbildung 43: Gefilterte Punktwolke B1 (oben links), B2 (oben rechts), B3 (unten links) und B4 (unten rechts).

4.6 Lokale Regression auf dem Quadratgitter

Die Berechnung der lokalen Orientierung von Fassaden durch lokale Regression (vgl. Kapitel 4.1) ist die Hauptaufgabe der in diesem Kapitel beschriebenen Funktion `eigenvectors_full_cells` (Anhang, S. [9]).

Ausgangsbasis ist die gefilterte 3D-Punktwolke (Kapitel 4.5). Diese Punktwolke enthält vor allem Fassadenpunkte. Die Punkte dieser Punktwolke werden in ein regelmäßiges Quadratgitter aufgeteilt (Kapitel 4.4). Die Breite jeder Zelle beträgt 1,6 m. Das Fenster für die lokale Regression hat bei Verwendung der 3x3-Umgebung eine Kantenlänge von 4,8 m. Für die Umgebung einer jeden Zelle wird die lokale Regression nach Pearson durchgeführt (siehe Kapitel 3.2). Die Zellen werden in einem Array `cells` verwaltet und besitzen zwei Indizes, jeweils für die X- und die Y-Richtung des Gitters. Jede Zelle besitzt ein Feld `is_wall` und ein Feld `incline`.

Eine Zelle wird als einer Fassade zugehörig betrachtet, wenn gilt: $\varrho < \varrho_{\max} = 0{,}3$ (siehe Kapitel 4.3). Eine Zelle, die diese Bedingung erfüllt, wird im Folgenden Fassadenzelle genannt. Sie wird markiert, in dem der Parameter `is_wall` den Wert `true` erhält.

Für alle Fassadenzellen wird deren Richtung durch den Eigenvektor $v_{\max}$ berechnet. Diese Richtung wird als Fassadenrichtung bezeichnet. Dazu wird der Winkel des Eigenvektors $v_{\max}$ zur x-Achse berechnet. Dieser nimmt Werte im Intervall $[-90°, 90°]$ an und wird als Feld `incline`.

Die lokale Regression wird nur berechnet, wenn die betreffende Zelle eine Mindestanzahl von Punkten enthält. Damit sollen Zellen ausgeschlossen werden, die keine Fassadenpunkte enthalten (vgl. Kapitel 4.4).

Die Ergebnisse der lokalen Regression für Gebäude B2 werden in der Abbildung 44 dargestellt.

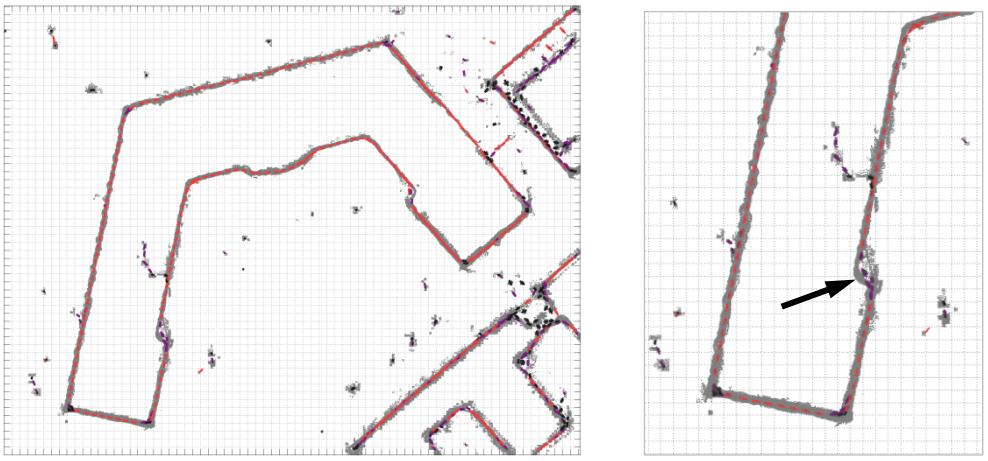

Abbildung 44: Ergebnisse der lokalen Regression für das Gebäude B2, links: gesamtes Gebäude, rechts: kurze Fassade wurde nicht detektiert (schwarzer Pfeil).

Die Fassadenrichtungen werden korrekt für die langen Fassaden und die gekrümmten Fassaden des Gebäudes B2 bestimmt. Dies wird durch die Darstellung der skalierten Eigenvektoren $v_{\max}$ sichtbar, die auf den Fassaden rot eingefärbt sind. In Abbildung 44 rechts wird deutlich, dass die kurze Fassade nicht detektiert wird (Eigenvektoren $v_{\max}$ sind violett dargestellt). Außerdem ist sichtbar, dass sich in der gefilterten Punktwolke in der Nähe der kurzen Fassade auch Punkte außerhalb der Fassade befinden. Dies erschwert die Detektion der kurzen Fassade zusätzlich.

Die Ergebnisse für die anderen Gebäude sind in Abbildung 45 dargestellt.

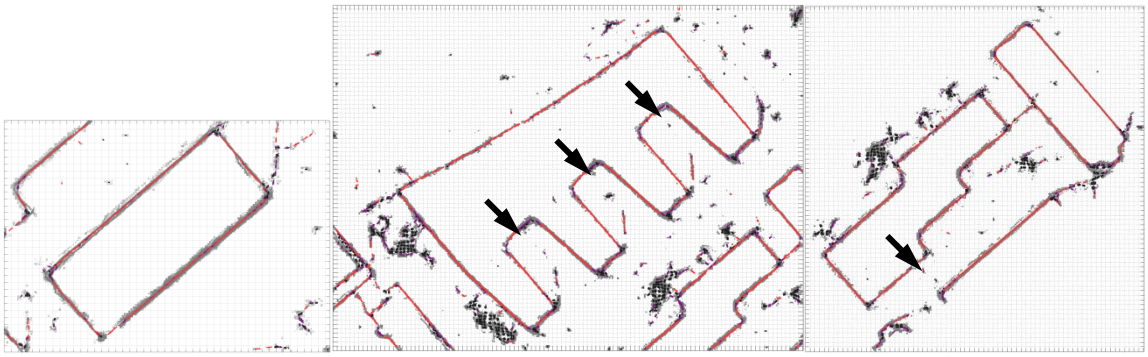

Abbildung 45: Ergebnisse der lokalen Regression für die Gebäude B1, B3, B4. Nicht detektierte Bereiche sind mit einem Pfeil markiert.

Deutlich sichtbar sind in der Abbildung 45 Zellen außerhalb von Fassaden, die fälschlicherweise als Fassadenzellen detektiert werden.

5 Fassadenextraktion – globaler Ansatz

Im Kapitel 4 wurde beschrieben, wie aus der 3D-Punktwolke Punkte in Fassadennähe extrahiert und durch ein regelmäßiges Quadratgitter in Zellen aufgeteilt werden. Durch lokale Regression wurde für jede Zelle bestimmt, ob sie zu einer Fassade gehört oder nicht. Für jede Fassadenzelle wird die Fassadenrichtung berechnet.

In diesem Kapitel wird ein mehrstufiger Algorithmus beschrieben, um Fassaden als geschlossenen Polygonzug mit zusätzlichen Informationen über die untere und obere Fassadenkante zu extrahieren. Dieser Algorithmus wurde durch Tests an den Gebäuden B1 - B4 schrittweise entwickelt. Die Entwicklung des Algorithmus wird am Beispiel des Gebäudes B2 illustriert. Dieses Gebäude hat durch kurze Fassaden, den gekrümmten Eingangsbereich und einen nordöstlich gelegenen Verbindungsgang eine komplexe Form. Der mehrstufige Algorithmus umfasst folgende Schritte:

- Vereinigung von benachbarten Fassadenzellen mit gleicher Fassadenrichtung zur Gruppen von Fassadenzellen: Abbildung 46 b, c (Kapitel 5.1.1, 5.1.2),
- Regression durch Gruppen von Fassadenzellen: Abbildung 46 d (Kapitel 5.2),
- Bestimmung von Fassadenstrecken durch Berechnung von Anfangs- und Endpunkten der Ausgleichsgeraden pro Fassade: Abbildung 46 d (Kapitel 5.2),
- Bestimmung von Nachbarschaftsbeziehungen zwischen Fassadenstrecken: Abbildung 46 e (Kapitel 5.3.1),
- Vereinigung von Fassadenstrecken: Abbildung 46 f (Kapitel 5.3.2),
- Berechnung von Fassadenecken: Abbildung 46 g (Kapitel 5.3.3),
- Zusammenfassen von benachbarten Fassadenstrecken zu einem Gebäude: Abbildung 46 h (Kapitel 5.5),
- Berechnung der unteren und oberen Fassadenkanten, Erstellung von Gebäuden: Abbildung 46 h (Kapitel 5.4, 5.5).

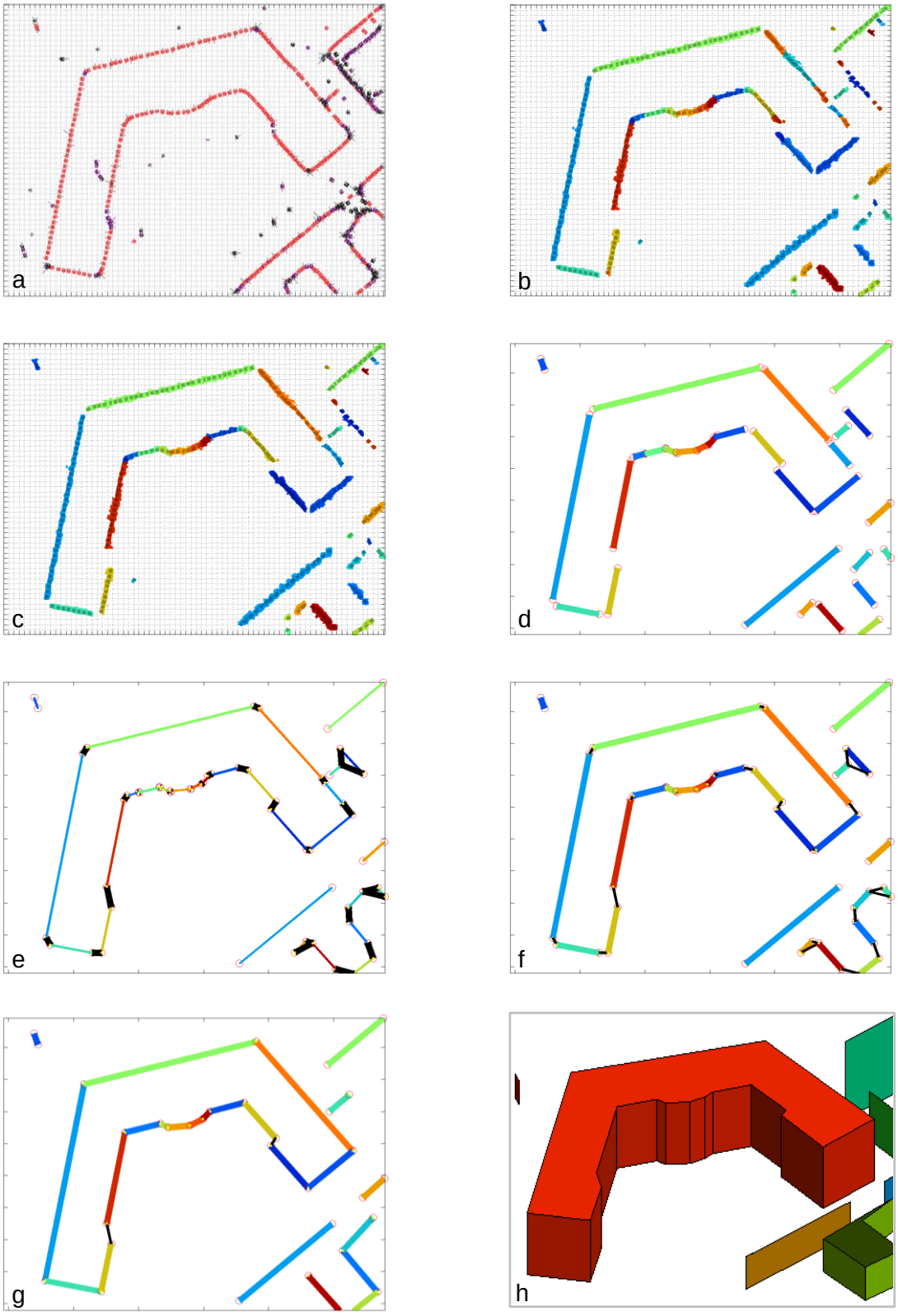

Abbildung 46: Folge von Algorithmen zur Fassadenextraktion, zeilenweise von oben nach unten, a) Richtungen aus lokaler Regression, b) Initialisierung von Gruppen von Fassadenzellen, c) Vereinigung von Gruppen von Fassadenzellen, d) Fassadenstrecken durch Regression, e) Nachbarschaftsbeziehungen, f) Vereinigung von Fassadenstrecken, g) Eckpunkte von Fassadenstrecken, h) Gebäude.

5.1 Initialisierung und Vereinigung von Gruppen von Fassadenzellen

5.1.1 Initialisierung von Gruppen von Fassadenzellen

In der Funktion `init_walls` (Anhang, S. [11]) werden Gruppen von Fassadenzellen geschaffen, welche benachbarte Fassadenzellen mit ähnliche Fassadenrichtung enthalten. Zwei Richtungen werden als ähnlich betrachtet, wenn ihre Differenz vom Betrag kleiner als ein zu wählender Parameter $\delta_{\text{init\_walls}}$ ist. Dieser hat beispielsweise einen Wert von 15°.

Beim Vergleich von zwei Richtungen α_1 und α_2 aus dem Intervall [-90°,90°] ist zu beachten, dass es zu Fehlern führen kann, wenn nur geprüft wird ob: $|\alpha_1 - \alpha_2| < \delta = 15°$. Als Beispiel werden die Richtungen α_1= 85° und α_2= - 85° betrachtet. Für ihre Differenz gilt: |85°-(-85°)| = 170° > δ , obwohl die Abweichung der Richtungen nur 10° beträgt. Eine Lösung des Problems besteht darin, im Fall von $|\alpha_1 - \alpha_2| > 90°$, den Winkel α_1 neu zu definieren als: $\alpha_1 := \alpha_1 - 180°$. Somit erhält man $\alpha_1 = -95°$ und |-95°-(-85°)| = 10° < 15°. Falls α_2 ein positives Vorzeichen hat, wird definiert α_1 als $\alpha_1 := \alpha_1 + 180°$. Diese Vorgehensweise wird im Folgenden immer angewändet, wenn zwei Winkel verglichen werden.

Jeder Fassadenzelle soll einer Gruppe von Fassadenzellen zugeordnet werden. Dazu wird folgende Struktur verwendet. Eine Gruppe von Fassadenzellen F wird über ihre ID, die sogenannte Fassaden-ID identifiziert, und enthält die Indizes aller enthaltenen Fassadenzellen $C_{k,l}$. Weiterhin enthält jede Fassadenzelle die Fassaden-ID der zugeordneten Gruppe von Fassadenzellen als Feld der Struktur: Gruppe von Fassadenzellen.

Das Quadratgitter wird spaltenweise von links nach rechts durchlaufen. Für jede aktuell untersuchte Basiszelle B werden die Nachbarzellen N aus der 3x3-Umgebung untersucht. Dabei werden nur Zellen betrachtet, die Fassadenzellen sind. In einer ersten Version des Algorithmus wird für jede Nachbarzelle N, die folgende Bedingung B1 geprüft:

$$|B_{\text{incline}} - N_{\text{incline}}| < \delta_{\text{init\_walls}} .$$

Hier sind B_{incline} und N_{incline} die Richtung der Zelle B und N als Winkel zur x-Achse.

Falls eine Nachbarzelle, die die Bedingung erfüllt, und bereits einer Gruppe F von Fassadenzellen zugeordnet ist, erhält die Basiszelle die Fassaden-ID der Nachbarzelle. Außerdem wird die Basiszelle der Gruppe F hinzugefügt.

Falls es keine Nachbarzelle gibt, die die Bedingung B1 erfüllt und einer Gruppe von Fassadenzellen zugeordnet ist, wird eine neue Fassaden-ID erzeugt. Die Basiszelle erhält diese

Fassaden-ID und wird als erstes Element der neuen Gruppe von Fassadenzellen hinzugefügt.

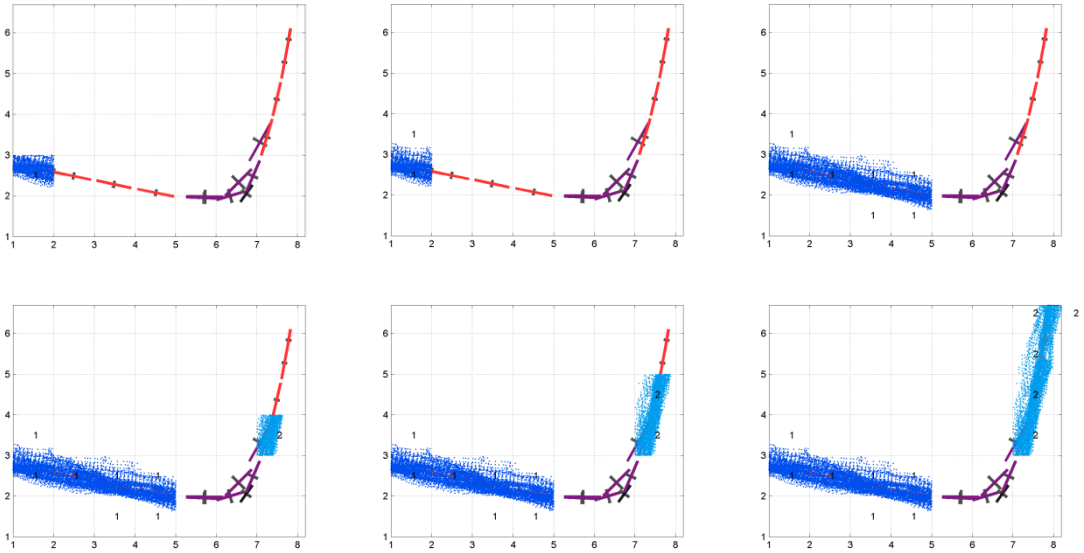

Abbildung 47: Generierung von 2 Gruppen von Fassadenzellen in verschiedenen Stufen (zeilenweise von oben nach unten, nicht alle Stufen dargestellt). Die erste Gruppe von Fassadenzellen hat die Fassaden-ID 1, die zweite die Fassaden-ID 2. Fassadenzellen werden mit der Fassaden-ID markiert. Außerdem werden die Punkte innerhalb der Zellen dargestellt.

Die Bedingung B1 hat einen Nachteil bei gekrümmten Fassaden. Als Beispiel wird eine Fassade betrachtet, deren Projektion ein Kreisbogen ist (Abbildung 48). Es kann der Fall eintreten, dass für jede Basiszelle eine Nachbarzelle existiert, die die Bedingung B1 erfüllt. In diesem Fall würden alle Zellen der gekrümmten Fassade einer einzigen Gruppe von Fassadenzellen zugeordnet. Das ist nicht erwünscht, da in einem späteren Schritt geplant ist, die Punkte aus der Gruppe der Fassadenzellen durch eine Gerade zu beschreiben. In diesem Fall würde die gekrümmte Fassade durch eine einzige Gerade approximiert und somit schlecht dargestellt.

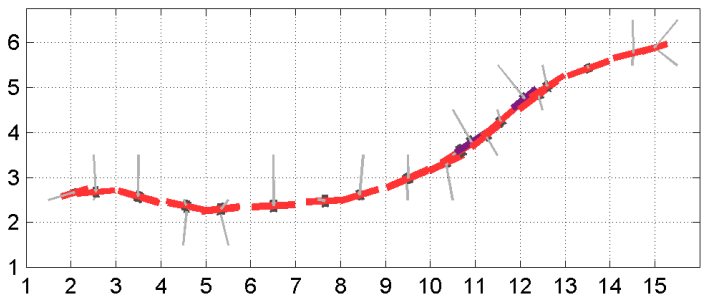

Abbildung 48: Eingangsbereich von Gebäude B2 mit leicht gekrümmter Fassade, die Abweichungen der Richtungen (rot) zwischen benachbarten Zellen sind gering.

Um diesen Nachteil der Bedingung B1 zu beheben, wird vorgeschlagen, dass die Gruppe der Fassadenzellen F die Eigenschaft der Richtung besitzt. Die Richtung F_{incline} einer Gruppe von Fassadenzellen wird als Mittelwert der Richtungen der enthaltenen Fassadenzellen bestimmt. Ziel der Verwendung des Mittelwertes ist es, Einflüsse durch eventuelle Ausreißer auszugleichen. Empirische Tests haben gezeigt, dass es ausreicht, den Mittelwert anhand der drei zuerst hinzugefügten Fassadenzellen zu berechnen.

Die aktuell verwendete Bedingung B2 lautet folgendermaßen. Für jede Basiszelle B gibt es eine Nachbarzelle N, die Teil einer Gruppe von Fassadenzellen F ist, mit:

$$|B_{\text{incline}} - F_{\text{incline}}| < \delta_{\text{init\_walls}}.$$

Durch diese Bedingung wird sichergestellt, dass die Richtungen innerhalb einer Gruppe von Fassadenzellen ähnlich sind. Die vorherige Bedingung B1 hat nur sichergestellt, dass die Richtungen zwischen benachbarten Fassadenzellen ähnlich sind.

In Abbildung 49 ist ein Bereich von Gebäude 2 mit einem halbrunden Eingang dargestellt. Unter Verwendung der Bedingung B2 wurde sie wie gewünscht in mehrere kleine Gruppen aufgeteilt.

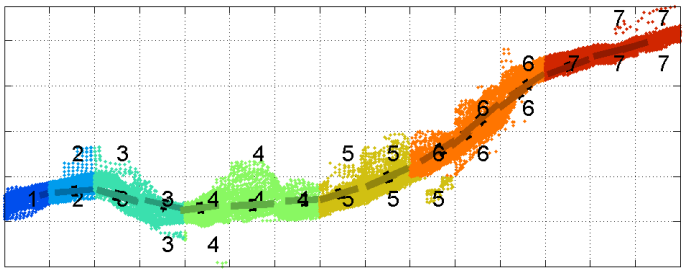

Abbildung 49: Eingangsbereich von Gebäude B2 mit gekrümmter Fassade, die Punktwolke wurde unter Verwendung von Bedingung B2 in kleine Bereiche mit ähnlicher Richtung aufgeteilt.

In Abbildung 50 ist das Ergebnis des Algorithmus `init_walls` dargestellt. Benachbarte Fassadenzellen mit ähnlicher Richtung werden für viele Bereiche erfolgreich zusammengefasst.

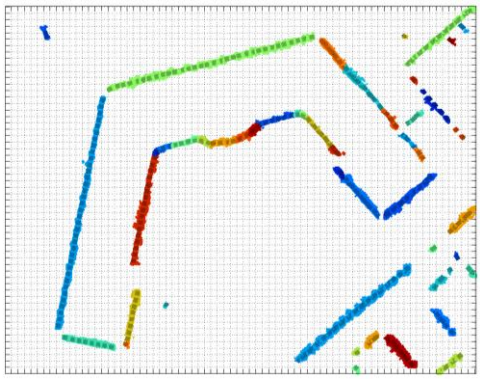

Abbildung 50: Darstellung von Gruppen von Fassadenzellen durch Farbkodierung.

Hier sind allerdings auch Bereiche auf der nordöstlichen Fassade zu sehen, wo Fassadenzellen mit ähnlicher Richtung nicht zusammengefasst wurden. Deshalb wird folgender Algorithmus entwickelt (Kapitel 5.1.2).

5.1.2 Vereinigung von Gruppen von Fassadenzellen

Die Aufgabe der Funktion `merge_walls` (Anhang, S. [15]) besteht darin, benachbarte Gruppen von Fassadenzellen, die eine ähnliche Richtung haben, zu vereinigen. Zwei Richtungen gelten als ähnlich, wenn ihre Differenz vom Betrag kleiner als der Parameter $\delta_{\text{merge\_walls}}$ ist. Dieser ist standardmäßig auf 10° eingestellt.

Das Quadratgitter wird spaltenweise von links nach rechts durchsucht. Für jede Zelle, Basiszelle genannt, werden die Nachbarzellen der 3x3-Umgebungen betrachtet. Für jedes Paar von Basiszelle und Nachbarzelle wird geprüft, ob beide Zellen in verschiedenen Gruppen von Fassadenzellen liegen. Wenn dies der Fall ist, wird getestet, ob die Richtungen der beiden Gruppen von Fassadenzellen ähnlich sind. Wenn die Richtungen ähnlich sind, werden die beiden Gruppen der Fassadenzellen zu einer Gruppe vereinigt. Die Gruppe von Fassadenzellen mit der kleineren Fassaden-ID, wird um die Zellen der Gruppe mit der größeren Fassaden-ID erweitert (Abbildung 51). Die Gruppe mit der größeren Fassaden-ID wird gelöscht. Gleichzeitig werden für die neu hinzugefügten Fassadenzellen die Fassaden-IDs aktualisiert.

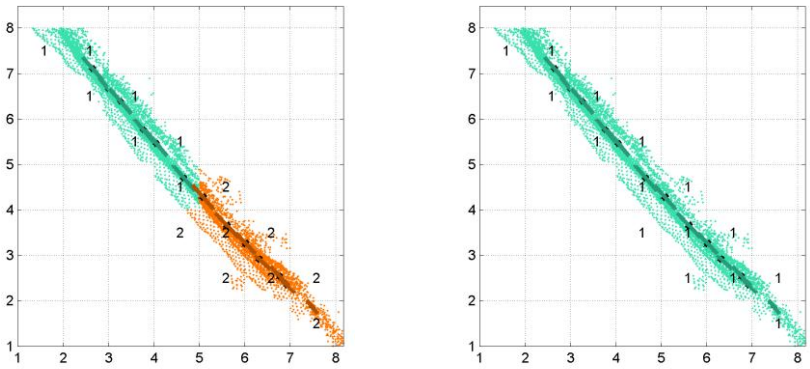

Abbildung 51: Vereinigung von Gruppe 1 und 2 von Fassadenzellen, links: vor der Vereinigung, rechts: nach der Vereinigung

Das Ergebnis der Funktion `merge_walls` wird in Abbildung 52 dargestellt (vgl. mit Abbildung 50).

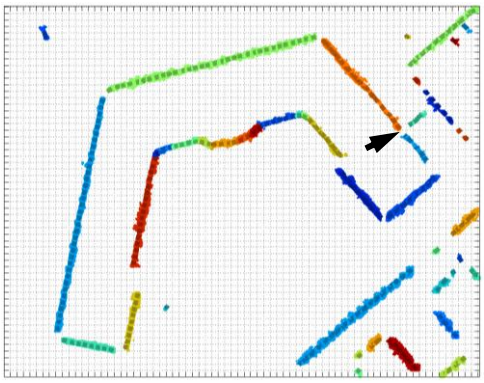

Abbildung 52: Ergebnis der Funktion merge_walls, Richtung der Fassade ist entsprechend Fassaden-ID eingefärbt (vgl. Abbildung 50).

Es ist zu sehen, dass nicht alle Fassaden durch eine einzige Gruppe von Fassadenzellen bedeckt sind. Insbesondere treten in dem mit einem Pfeil markierten Bereich Lücken auf. Diese werden durch einen der folgenden Algorithmen (siehe Kapitel 5.3.2) geschlossen.

5.2 Bestimmung von Fassadenstrecken durch Regression

Nun wird für jede Gruppe von Fassadenzellen die Regression nach Pearson durchgeführt (vgl. Kapitel 3.2). Dazu wird die Ausgleichsgerade durch die Punkte aller Fassadenzellen einer Gruppe berechnet. Diese ist gegeben durch den Schwerpunkt m und den normierten Richtungsvektor $v = v_{\max}$.

Anschließend werden der Anfangs- und Endpunkt für eine Strecke auf der Ausgleichsgerade berechnet. Diese Punkte werden in zwei Schritten bestimmt. Zunächst wird für jeden

Punkt p einer Fassadenzelle die Projektion p' auf die entsprechende Ausgleichsgerade berechnet [Gellert (1986), S. 588]:

$$p' = m + [(p - m) \circ v]v.$$

Für jeden projizierten Punkt p' wird die Entfernung λ zum Schwerpunkt der Fassadenpunktwolke bestimmt. Dabei wird das Skalarprodukt mit dem Richtungsvektor verwendet, um eine vorzeichenbehaftete Größe zu erhalten:

$$\lambda = (p - m) \circ v.$$

Die projizierten Punkte mit der minimalen und maximalen Entfernung λ zum Schwerpunkt definieren eine Strecke auf der Ausgleichsgerade. Diese Strecke wird im Folgenden Fassadenstrecke genannt. Die Koordinaten der beiden Endpunkte werden in dem Feld `segment` der Gruppe von Fassadenzellen abgespeichert. Die oben beschriebenen Aufgaben werden durch die Funktion `make_segments` (Anhang, S. [18]) realisiert. In der Abbildung 53 rechts werden die berechneten Strecken dargestellt.

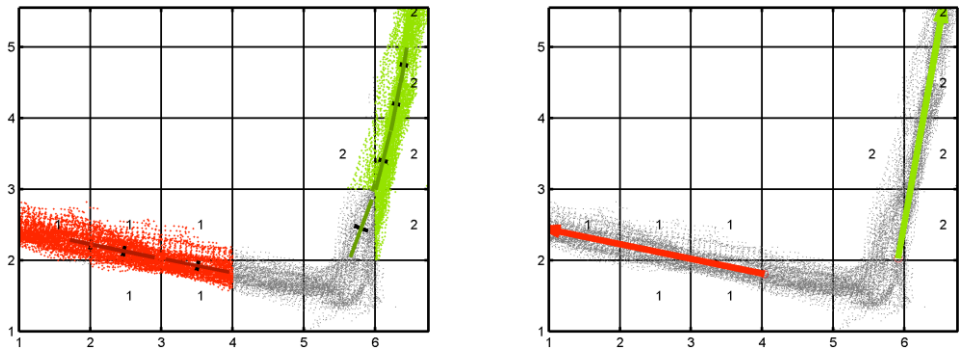

Abbildung 53: Regression durch Punkte aus Gruppe von Fassadenzellen, links: 2 Gruppen von Fassadenzellen, rechts: 2 Fassadenstrecken als Pfeil dargestellt.

Um Zellen zu entfernen, die fälschlicherweise als Fassadenzellen erkannt werden, werden zwei Parameter verwendet. Der Parameter $l_{\text{segment\_min\_length}}$ ist die Mindestlänge für eine Fassadenstrecke. Der Parameter $n_{\text{segment\_min\_cells}}$ ist die Mindestanzahl von Fassadenzellen einer Gruppe von Fassadenzellen. Wenn eine Gruppe von Fassadenzellen eine dieser Anforderungen nicht erfüllt, wird deren Parameter `iswall` auf `false` gesetzt, damit sie nicht mehr verwendet wird. Diese Entfernung von Gruppen von Fassadenzellen ist riskant, da auch korrekt detektierte kurze Fassaden entfernt werden können. Wünschenswert wäre es, wenn in der gefilterten Punktwolke keine Punkte vorhanden sind, die außerhalb von Fassaden liegen. In diesem Fall könnte auf die zusätzliche Filterung verzichtet werden.

5.3 Erstellung von geschlossenen Polygonzügen

Die oben berechneten Fassadenstrecken decken allerdings nicht die ganze Länge der projizierten Fassade ab. In den Endbereichen der Strecken liegen Punkte die zwar Teil der Fassade sind, aber die nicht zu Fassadenzellen gehören (vgl. Abbildung 53 rechts). Insbesondere haben die Fassadenstrecken von räumlich benachbarten Fassaden keinen gemeinsamen Endpunkt.

Um die gemeinsamen Eckpunkte zu bestimmen, soll der Eckpunkt als Schnittpunkt der entsprechenden 2 Ausgleichsgeraden bestimmt werden. Ist der Eckpunkt bekannt, können die Fassadenstrecken bis zu diesem Eckpunkt verlängert werden. Die so verlängerte Strecke deckt die ganze horizontale Fassadenläge ab.

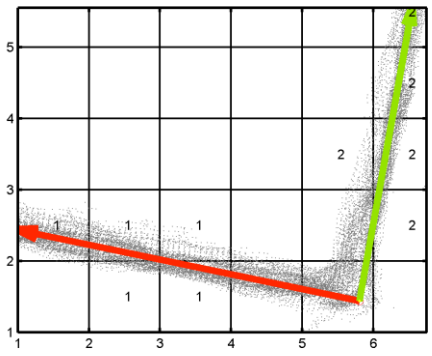

Abbildung 54: Verlängerung von zwei Fassadenstrecken bis zu ihrem gemeinsamen Eckpunkt.

Die Berechnung des Schnittpunkts ist aber nicht ohne weiteres möglich, da es keine Nachbarschaftsbeziehungen zwischen Gruppen von Fassadenzellen gibt. Somit ist für eine gegebene Gruppe von Fassadenzellen nicht bekannt, welches die benachbarten Gruppe von Fassadenzellen ist.

In den folgenden Kapiteln wird erläutert, wie die Bestimmung der Nachbarschaftsbeziehungen (Kapitel 5.3.1), die Vereinigung von benachbarten Fassadenstrecken (Kapitel 5.3.2) und die Bestimmung von Eckpunkten (Kapitel 5.3.3) durchgeführt wird. Diese Schritte sind die Grundlage für die Erstellung von geschlossenen Polygonen als Umriss eines Gebäudes.

5.3.1 Nachbarschaft von Fassadenstrecken

Die Bestimmung der Nachbarschaft von Fassadenstrecken ist in der Funktion `find_pairs` (Anhang, S. [21]) implementiert. Für jede Fassadenstrecke und jeden der zwei Endpunkte p_{base} wird eine Schleife über alle anderen Fassadenstrecken und deren Endpunkte $p_{\text{candidate}}$

durchgeführt. Von allen Endpunkten $p_{\text{candidate}}$ für die der Abstand zum Ausgangsendpunkt p_{base} kleiner ist ein voreingestellter maximaler Suchabstand $l_{\text{max\_pair\_distance}}$ von beispielsweise 10 m:

$$|p_{\text{base}} - p_{\text{candidate}}| < l_{\text{max\_pair\_distance}}$$

wird der mit dem minimalen Abstand gewählt. Wenn es keinen Endpunkt mit Abstand kleiner als $l_{\text{max\_pair\_distance}}$ gibt, dann wird keine Nachbarschaftsbeziehung definiert.

Die Nachbarschaftsrelation wird in einer speziellen Struktur für jeden der beiden Endpunkte einer Fassadenstrecke abgespeichert. Diese Struktur wird `next_corner` genannt. Sie enthält:

- die ID der benachbarten Fassade,
- einen Index, der den Wert 1 oder 2 annimmt und bestimmt, welche der beiden Endpunkte der benachbarten Fassadenstrecke der nächstliegende Endpunkt ist,
- einen boolschen Wert `is_corner` (mit `false` initialisiert) der bestimmt, ob für eine Ecke ein Schnittpunkt berechnet wird oder nicht (siehe weiter unten),
- einen boolschen Wert `use` (mit `false` initialisiert), der bestimmt, ob diese Struktur sinnvolle Werte enthält (siehe weiter unten).

In Abbildung 55 werden die topologischen Verbindungen zwischen den Fassadenstrecken dargestellt.

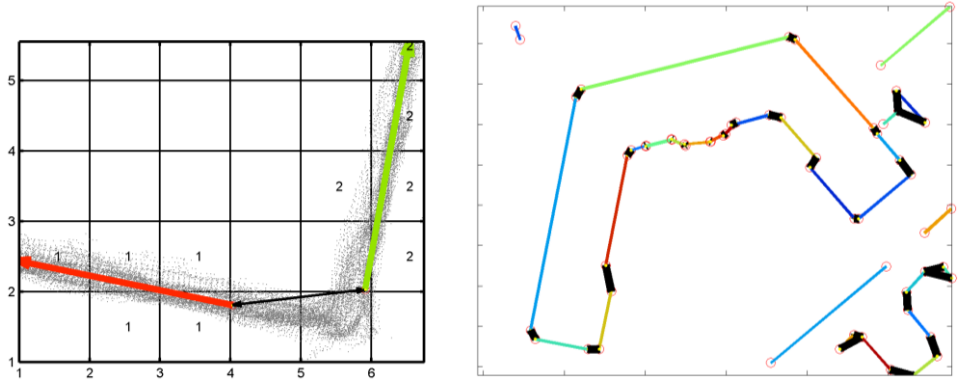

Abbildung 55: links: Darstellung der Nachbarschaftsbeziehungen zwischen Endpunkten der Fassadenstrecke (rot) und Fassadenstrecke (grün) durch schwarze Pfeile, rechts: Nachbarschaftbeziehungen für Gebäude B2 (dicke schwarze Linien).

Da für jede Fassadenstrecke eine Schleife über alle anderen Fassadenstrecken gemacht wird, ist der Aufwand dieses Algorithmus quadratisch. Dieser Algorithmus kann somit für eine große Anzahl von Gruppen von Fassadenzelle zu einer hohen Rechenzeit führen. Es soll nun erläutert werden, wie dieser Algorithmus durch einen Algorithmus mit linearem Aufwand ersetzt werden kann. Zunächst wird die Zelle C_{base} des Quadratgitters bestimmt, in der der

Endpunkt p_{base} liegt. Anschließend wird die quadratische Umgebung bestimmt, die alle Zellen $C_{\text{candidate}}$ mit Abstand C_{base} vom kleiner als $l_{\text{max\_pair\_distance}}$ enthält. Für jede Zelle $C_{\text{candidate}}$ innerhalb dieser Umgebung wird geprüft, ob diese zu einer Gruppen von Fassadenzellen gehört. Ist dies der Fall, dann wird der Abstand beider Endpunkte der zugehörigen Fassadenstrecke zum Endpunkt p_{base} berechnet. Da die Anzahl der Zellen in der quadratischen Umgebung nur vom Gitterabstand l_{interval} und maximalen Suchabstand $l_{\text{max\_pair\_distance}}$ abhängt, hat der Algorithmus einen linearen Aufwand.

Die hier vorgeschlagene Bestimmung der Nachbarschaftbeziehungen über den Abstand ist einfach zu implementieren, kann aber zu Fehlern führen. Beispielsweise kann es vorkommen, dass Fassaden von zwei benachbarten Gebäuden als benachbart erkannt werden, wenn der Abstand zwischen ihnen klein genug ist. Meist werden die Fehler in der Bestimmung von Nachbarschaften dadurch verursacht, dass eine der benachbarten Fassaden nur zu einem Teil detektiert wird oder dass bestimmte Bereiche fälschlicherweise als Fassade detektiert werden.

Um grobe Fehler zu eliminieren, wird für jede Nachbarschaftsbeziehung von einem Endpunkt p zu einem Endpunkt q geprüft, ob die Nachbarschaftsbeziehung auch umgekehrt von q auch zu p gilt. Ist dies nicht der Fall, dann ist die Nachbarschaftsbeziehung falsch und in der Struktur `next_corner` wird der Wert `use` auf `false` gesetzt. Im Folgenden werden Nachbarschaftsbeziehungen, für die der Parameter `use` den Wert `false` annimmt, nicht verwendet.

5.3.2 Vereinigung von Fassadenstrecken

In manchen Fällen werden Fassaden durch 2 oder mehrere Fassadenstrecken abgedeckt (Abbildung 56). Dies ist unerwünscht.

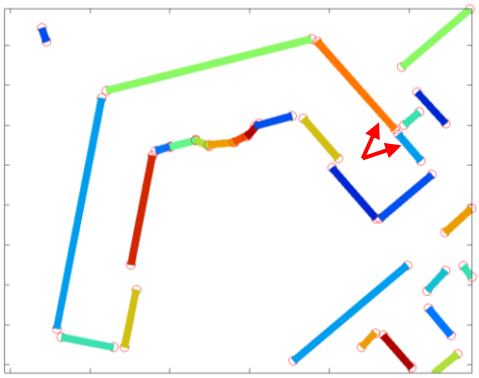

Abbildung 56: durch Pfeile markierte Fassadenstrecken liegen auf derselben Fassade und sollen vereinigt werden.

Deshalb wird ein Algorithmus zur Vereinigung von benachbarten Fassadenstrecken entwikkelt. Diese Aufgabe erfüllt die Funktion `merge_gaps` (Anhang, S. [25]).

Ähnlich wie in der Funktion `find_corner` (Anhang, S. [29]) wird die Nachbarschaftsbeziehung zwischen den Fassadenstrecken ausgenutzt. Für benachbarte Endpunkte p_2 und q_1 werden die zugehörige Fassadenstrecken $s = [p_1, p_2]$ und $t = [q_1, q_2]$ betrachtet. Wenn folgende zwei Kriterien erfüllt sind, werden die beiden Gruppen von Fassadenzellen vereinigt:

a) der Betrag des Winkel zwischen den Strecken s und t ist kleiner als ein festgelegter Parameter $\delta_{\text{merge\_segments}}$ (z. Bsp. 10°)
b) der Abstand zwischen dem Endpunkt p und dessen Projektion auf die Ausgleichsgerade durch die Nachbarstrecke t ist kleiner als ein festgelegter Parameter $l_{\text{merge\_segments}}$ (z. B. 1 m)

Die Fassadenstrecken s und t werden dann durch die eine neue Strecke $v = [p_1, q_2]$ ersetzt. Dadurch wird keine neue Ausgleichsgerade durch Regression bestimmt, sondern es werden die Endpunkte p_1 und q_2 der Fassadenstrecken s und t verwendet. Die Nachbarschaftsbeziehungen `next_corner,` der zu den Strecken s und t benachbarten Fassadenstrecken, werden aktualisiert.

Die Bedingungen a) und b) sollen sicherstellen, dass zwei Fassadenstrecken nur vereinigt werden, wenn sie fast auf einer gemeinsamen Geraden liegen. Die Bedingung a) stellt sicher, dass die Strecken fast parallel liegen. Eine Vereinigung von Strecken mit einem großen Winkel ist nicht erwünscht, da Details der Fassaden verlorengehen würden (siehe Abbildung 57, blaue Pfeile). Die Bedingung b) stellt sicher, dass sie nicht parallel zueinander verschoben liegen. Eine Vereinigung von parallelen, aber zu einander verschobenen Geraden ist aus demselben Grunde nicht erwünscht (siehe Abbildung 57, rote Pfeile).

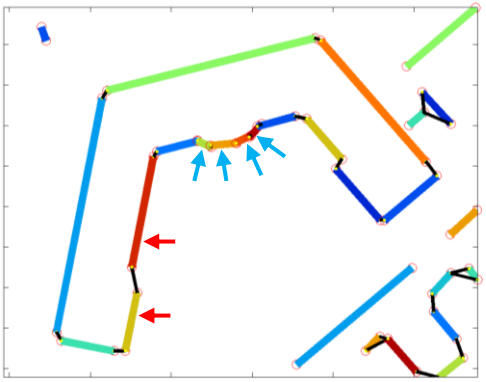

Abbildung 57: Vereinigung von Fassadenstrecken für Gebäude B2, Details bleiben nach Vereinigung erhalten.

5.3.3 Eckpunkte von Fassadenstrecken

Die Berechnung der Fassadeneckpunkte wird im Algorithmus `find_corner` (Anhang, S. [29]) implementiert. Für jede Fassadenstrecke $s = [p_1, p_2]$ und jeden ihrer beiden Endpunkte wird die benachbarte Fassadenstrecke $t = [q_1, q_2]$ anhand der vorhanden Daten aus Nachbarschaftsrelation (siehe Abschnitt 5.3.1) bestimmt. Der Schnittpunkt zwischen der Ausgleichsgerade und der benachbarten Ausgleichsgerade wird bestimmt, wenn der Winkel zwischen beiden Geraden größer als ein Parameter $\delta_{\text{min\_corners}}$ ist.

Anfangs wurde für den Parameter $\delta_{\text{min\_corners}}$ der Wert 80° gewählt, damit für typische Gebäudeecken von 90° die Ecke als Schnittpunkt bestimmt wird. Damit der Schnittpunkt auch bei leicht gekrümmten Fassaden berechnet wird, wurde der Wert von $\delta_{\text{min\_corners}}$ auf 15° reduziert. Der Wert von $\delta_{\text{min\_corners}}$ sollte nicht zu klein gewählt werden, damit bei fast parallelen Strecken kein Schnittpunkt berechnet wird.

Zur Berechnung des Schnittpunktes wird die Ausgleichsgerade g in Parameterform [Gellert et al. (1986), S. 588 f]

$$p \epsilon g: \; p = p_0 + \lambda \cdot v$$

und die benachbarte Ausgleichsgerade h in Normalenform

$$q \epsilon h: \; (q - q_0) \circ n = 0$$

dargestellt. Der Schnittpunkt p_s auf der Ausgleichsgerade g ergibt sich durch:

$$(q - q_0) \circ n = \; (p_0 + \lambda \cdot v - q_0) \circ n = 0, \;\; \text{d.h.} \;\; \lambda = -\frac{(p_0 - q_0) \circ n}{v \circ n} \; .$$

Nun werden die Fassadenstrecken aktualisiert. Dies wird für den Fall illustriert, dass die Endpunkte p_2 und q_1 benachbart sind. Die Fassadenstrecken s und t werden nun neu definiert als: $s = [p_1, p_s]$ und $t = [p_s, q_2]$. Zusätzlich wird in der Struktur `next_corner` für die Endpunkte p_2 und q_1 der boolesche Wert `is_corner` auf `true` gesetzt. Damit wird markiert, dass der Schnittpunkt für eine Fassadenstrecke berechnet wurde.

In der Abbildung 58 wird das Gebäude B2 mit den Eckpunkten dargestellt.

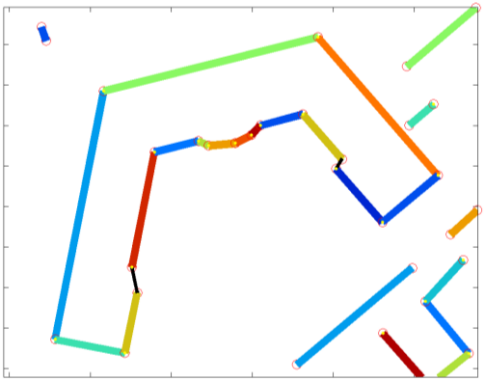

Abbildung 58: Berechnung der Eckpunkte für das Gebäude B2.

5.4 Höhe von den unteren und oberen Fassadenkanten

Die Berechnung der absoluten Z-Koordinaten sowohl von den oberen als auch von den unteren Fassadenkanten wird durch die Funktion `calculate_height` (Anhang, S. [31]) durchgeführt.

Um die Höheninformationen auszuwerten, ist es notwendig die Punkte der ungefilterten 3D-Ausgangspunktwolke zu verwenden, da in der gefilterten Punktwolke Boden und Dachpunkte entfernt wurden. Um den Algorithmus zu beschleunigen, werden für jede Gruppe von Fassadenzellen nur die Punkte verwendet, die in den Fassadenzellen liegen. Diese Punkte werden in einer temporären 3D-Punktwolke gespeichert.

Dazu werden die Punkte aus der ungefilterten 3D-Punktwolke in einem temporären Quadratgitter gespeichert. Dieses Gitter hat dieselbe Lage wie das Quadratgitter, das die Fassadenzellen enthält. Für jede Gruppe von Fassadenzellen wird eine Schleife über alle enthaltenen Fassadenzellen durchgeführt. Für jede Fassadenzelle werden deren Indizes im Quadratgitter ermittelt. Nun wird die Zelle aus dem temporären Quadratgitter mit denselben Indizes bestimmt. Deren Punkte werden in einer temporären 3D-Punktwolke gesammelt.

Aus den Punkten der temporären 3D-Punktwolke werden nur diejenigen ausgewählt, die in der Entfernung von maximal 0,50 m von der jeweiligen Fassadenstrecke liegen. Um die Ausreißer aus diesen Punkten zu eliminieren, werden für die Z-Werte die unteren und oberen Quantile entfernt. Üblicherweise werden dazu Werte bis 1% verwendet. Von den verbleibenden 3D-Punkten werden der kleinste und der größte Z-Wert genommen. Der kleinste Z-Wert definiert die minimale Höhe $z_{\min}$ der Gruppe von Fassadenzellen. Der größte Z-Wert bestimmt die maximale Höhe $z_{\max}$ der Gruppe von Fassadenzellen. Diese Werte werden als Feld `min_height` und `max_height` für jede Gruppe von Fassadenzelle gespeichert.

Ein Vergleich mit den tachymetrisch gemessenen Höhen zeigt, dass die obere Fassadenkante zu hohe Werte aufweist. Deshalb werden im Algorithmus die oberen Fassadenkanten durch Verwendung von unüblich hohen Werten von 5%-Quantile entfernt (Diskussion der Ergebnisse in Kapitel 6.1).

5.5 Erstellung von Gebäuden

Anhand der Nachbarschaftsbeziehungen soll nun der Gebäudegrundriss als geschlossenes Polygon erstellt werden. Dazu dient die Funktion `make_buildings` (Anhang, S. [33]). Diese Funktion soll auch für den Fall Ergebnisse liefern, dass nicht alle Fassaden eines Gebäudes detektiert wurden. In diesem Fall wird für eine Gruppe von zusammenhängenden Fassaden ein offenes Polygon erstellt. Das Polygon wird erstellt indem, ausgehend von einer Gruppe von Fassadenzellen, die jeweils benachbarte Gruppe von Fassadenzellen aufgesucht wird.

Für jede besuchte Gruppe von Fassadenzellen werden dem Polygon Endpunkte der zugehörigen Fassadenstrecke hinzugefügt. Wenn der Schnittpunkt zwischen zwei benachbarten Fassadenstrecken berechnet wurde, ist es nicht notwendig, den gemeinsamen Endpunkt der beiden Fassadenstrecken dem Polygon zweimal hinzuzufügen. Deshalb wird in diesem Fall dem Polygon nur ein Endpunkt der zugehörigen Fassadenstrecke hinzugefügt. Zur Prüfung ob ein Schnittpunkt berechnet wurde, wird in der Nachbarschaftsbeziehung `next_corner` das Feld `is_corner` auf den Wert `true` getestet.

Für ein geschlossenes Gebäude mit n Ecken enthält das Grundrisspolygon n+1 Punkte, da der letzte Punkt wiederholt wird. Dies ermöglicht die Kontrolle, ob das Polygon offen oder geschlossen ist.

Im Algorithmus wird eine Schleife über alle Gruppen von Fassadenzellen durchgeführt. Beim Ablauf des Algorithmus wird jeder Gruppe von Fassadenzellen die ID eines Gebäudes zugewiesen. Diese ID wird in einem speziellen Feld `building_id` als Feld der Gruppe von Fassadenelementen gespeichert. Das Feld `building_id` wird mit 0 initialisiert. Die Gebäude-ID beginnt mit 1 und wird für jedes neue Gebäude um 1 erhöht.

Die Höhen der unteren und oberen Fassadenkante eines jeden Gebäudes werden jeweils als Median der Höhen unteren und oberen Fassadenkanten des Gebäudes berechnet. Durch die Verwendung des Median sollen Ausreißer bei der Berechnung der Höhen der unteren und oberen Fassadenkante eliminiert werden.

6 Ergebnisse und Plausibilitätsprüfung

In diesem Kapitel werden die Ergebnisse der Extraktion von senkrechten Fassadenebenen für die Gebäude B1, B2, B3 und B4 vorgestellt.

In dem Algorithmus werden insgesamt 15 Parameter verwendet. Zunächst wird durch verschiedene Tests eine Kombination von Parametern bestimmt, bei denen für alle Gebäude gute Ergebnisse erzielt werden. Diese Wahl der Parameter wird Basiskombination genannt und ist im Formelverzeichnis beschrieben (Anhang, S. [1]). Unter guten Ergebnissen wird die genaue Extraktion der meisten Fassadenebenen für alle Gebäude verstanden. Die Genauigkeit der extrahierten Fassadenebenen wird relativ zum Nadir-TrueOrthobild und Nadir-DOM geprüft.

6.1 Ergebnisse für das Gebäude B1

Für das Gebäude B1 werden alle vier Fassaden als geschlossener Polygonzug extrahiert (Abbildung 59, links). Für jede Fassadenstrecke wurde die Standardabweichung berechnet und in der Abbildung dargestellt. Die Standardabweichungen für die extrahierten Fassadenstrecken beträgt von 0,24 m bis 0,33 m (Abbildung 59, links).

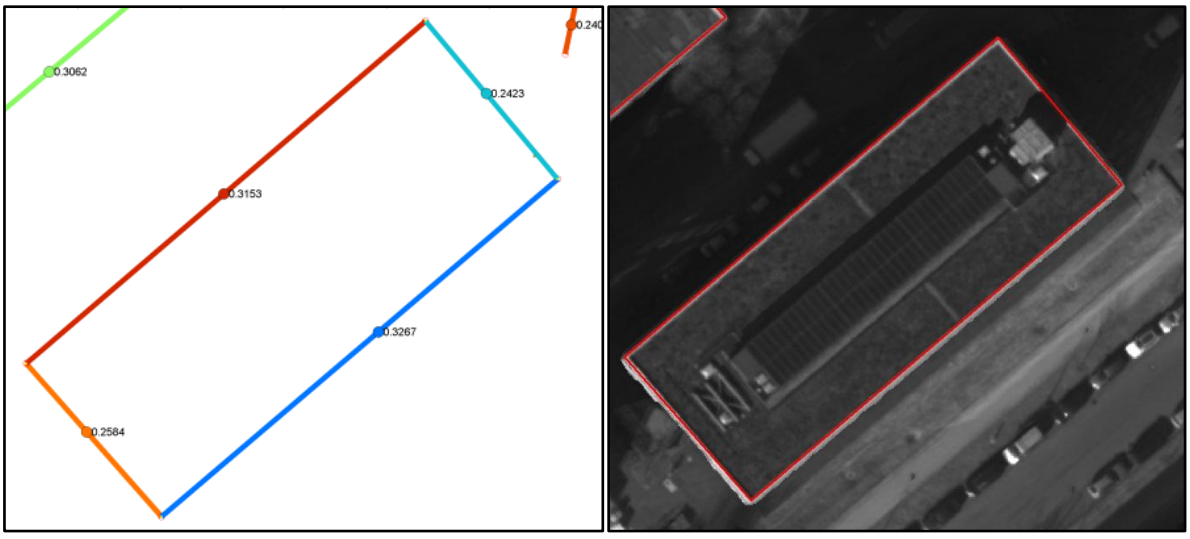

Abbildung 59: links: Extrahierte Fassadenebenen des Gebäudes B1, rechts: Ergebnisse des Gebäude B1 im Vergleich mit dem Nadir-TrueOrthobild.

Die extrahierten Fassadenstrecken wurden als Textdatei aus MatLab exportiert und in ArcMap 10 digitalisiert. Ein Vergleich des Nadir-TrueOrthobildes mit den extrahierten Fassa-

denstrecken zeigt eine gute Übereinstimmung (Abbildung 59, rechts). Auf der Vergrößerung (Abbildung 60) wird gezeigt, dass die extrahierte Fassade (rote Strecke) in der Mitte der Fassade des Nadir-TrueOrthobildes liegt. Das bedeutet, dass die extrahierten Fassadenebenen eine gute relative Genauigkeit aufweisen. Die Breite der Fassade auf dem Nadir-TrueOrthobild beträgt 5 Pixel. Bei einer Bodenauflösung 0,15 m und Berücksichtigung der diagonalen Lage ergibt sich eine absolute Breite von 0,85 m (0,15*$\sqrt{2}$*5). Die relative Genauigkeit der extrahierten Fassade für das Gebäude B1 kann also mit mindestens 0,425 m angegeben werden.

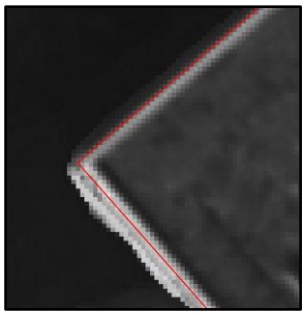

Abbildung 60: Vergrößerung einer Ecke.

Die berechnete Höhe der oberen Fassadenkante beträgt 103,50 m und weicht von der tachymetrisch gemessenen Höhe von 102,98 m um 0,52 m ab. Die berechnete Höhe der unteren Fassadenkante beträgt 74,64 m und ist im Vergleich zu der tachymetrisch gemessenen durchschnittlichen Höhe von 74,50 m um 0,13 m zu hoch. In der Abbildung 61 wird das Gebäude B1 als ein geschlossener Polygonzug mit dem Nadir-DOM im Programm ArcScene von ESRI verglichen.

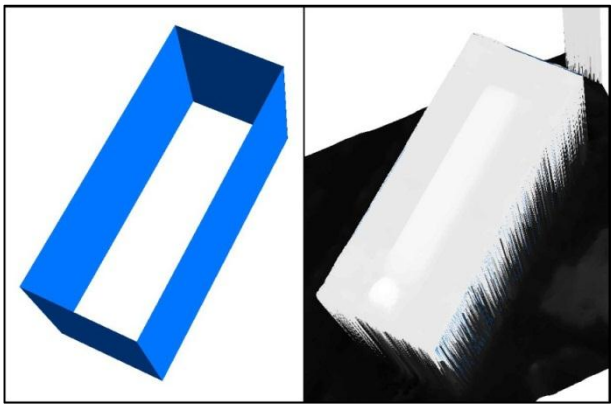

Abbildung 61: Extrahierte Fassadenebenen des Gebäudes B1 mit dem Nadir-DOM zusammengestellt.

Um die Ursache der Abweichungen in der Höhe zu analysieren, wird ein Histogramm der Z-Werte für die südöstliche Fassade von Gebäude B1 erstellt (Abbildung 62). Deutlich sichtbar ist die große Anzahl von Punkten in den Klassen mit dem Z-Wert größer als 103,0 m, also über der tachymetrisch gemessenen Höhe des Gebäudes B1. In den Klassen mit dem

Wert über 103,0 m befinden sich 2330 Punkte, was 14,8% der gesamten Menge von Punkten (15738) auf dieser Fassade darstellt. Das bedeutet, dass die Fassadenpunkte in der 3D-Punktwolke zu hoch liegen. Die Ursache dafür ist unklar. Bei einer Eliminierung des oberen 5 % Quantils wird die Höhe mit einer Abweichung +0,5 m bestimmt. Die minimale Höhe beträgt 74,64 m und ist im Vergleich mit der durchschnittlichen tachymetrisch gemessenen Höhe um 0,14 m zu hoch.

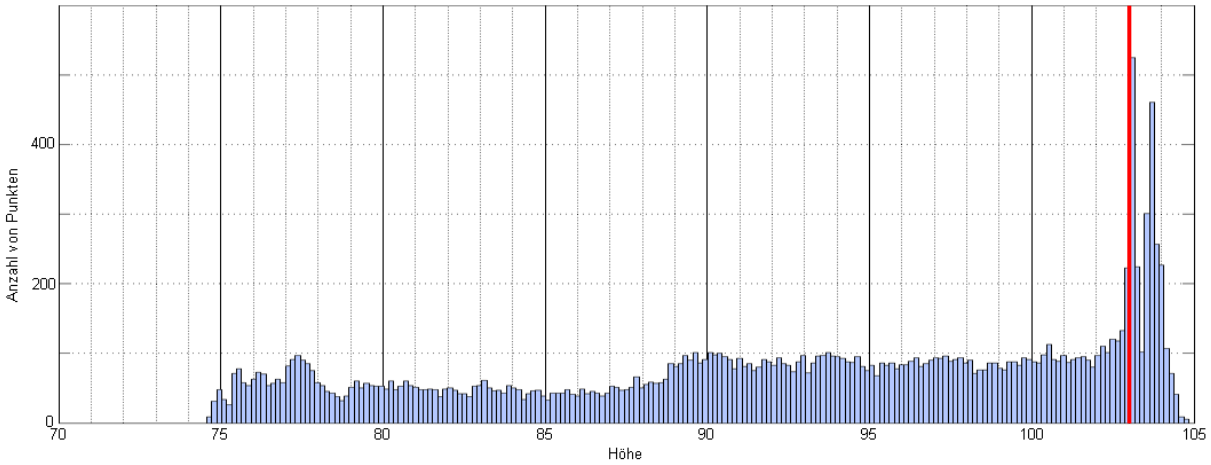

Abbildung 62: Histogramm der Höhen für die südöstliche Fassade von Gebäude B1, die tachymetrisch bestimmte Höhe beträgt 103,0 m (rot markiert).

Die Fassaden von Gebäude B1 werden mit derselben Qualität extrahiert, auch wenn die Parameter zur Fassadenextraktion variiert werden. Das heißt, dass die Ergebnisse für das Gebäudes B1 stabil sind. Ursache ist der einfache Grundriss, das Vorhandensein von langen Fassaden und eine große Gebäudehöhe.

6.2 Ergebnisse für das Gebäude B2

Im Gebäude B2 werden fast alle auf die XY-Ebene projizierten Fassadenebenen extrahiert. Eine Ausnahme stellen zwei kurze Fassaden dar, die auf der Abbildung 63 (links) als schwarze Strecken dargestellt werden. Die horizontale Länge dieser Fassaden beträgt 3 m. Sie werden nicht extrahiert, da das Regressionsfenster eine Breite von 4,8 m hat, was die Extraktion von Fassadenstrecken, die kürzer als 4,8 m sind, verhindert. Trotzdem ist es hier, dank der Herstellung von Nachbarschaftsbeziehungen zwischen den Fassadenstrecken (vgl. Kapitel 5.3.1), gelungen, einen geschlossenen Polygonzug von Fassadenstrecken zu erstellen.

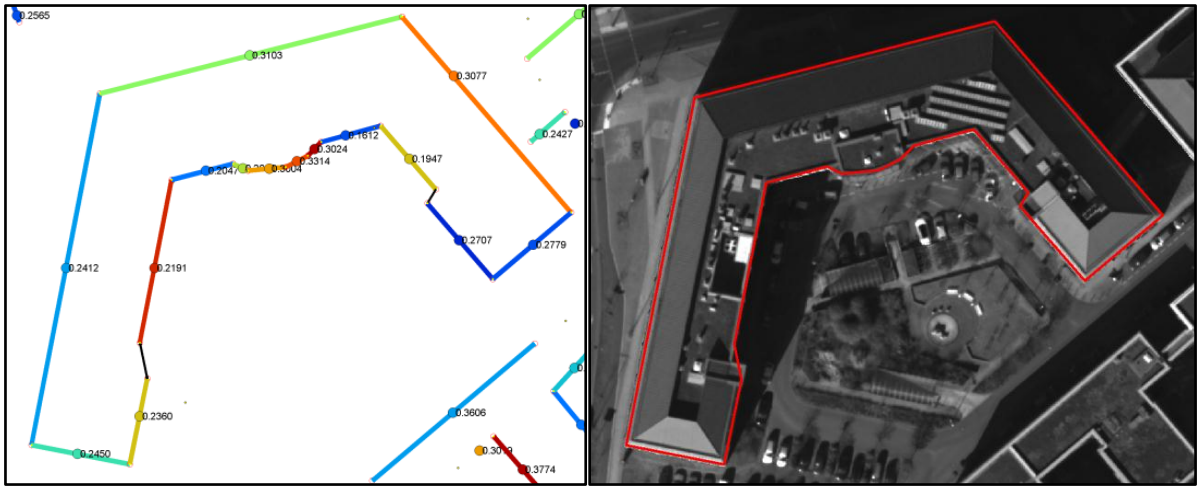

Abbildung 63: links: Extrahierte Fassadenebenen des Gebäudes B2, rechts: Ergebnisse des Gebäude B2 im Vergleich mit dem Nadir-TrueOrthobild.

Der gekrümmte Eingang wird durch vier Fassadenstrecken approximiert. Die Standardabweichung der berechneten Fassadenstrecken beträgt zwischen 0,21 m bis 0,31 m (Abbildung 63, links).

Ein Vergleich der extrahierten Fassadenstrecken mit dem Nadir-TrueOrthobild zeigt eine gute Übereinstimmung bei allen Fassaden außer bei den zwei kurzen (Abbildung 63, rechts).

Die absolute Höhe der berechneten oberen Fassadenkante beträgt 95,05 m und weicht von der tachymetrisch gemessenen Höhe 93,2 m um 1,85 m ab. Die Höhe der berechneten unteren Fassadenkante beträgt 74,89 m und ist im Vergleich zu der tachymetrisch gemessenen durchschnittlichen Höhe um 0,49 m zu hoch. Auf der Abbildung 64 wird der dreidimensionale Gebäudekörper des Gebäudes B2 mit dem Nadir-DOM verglichen.

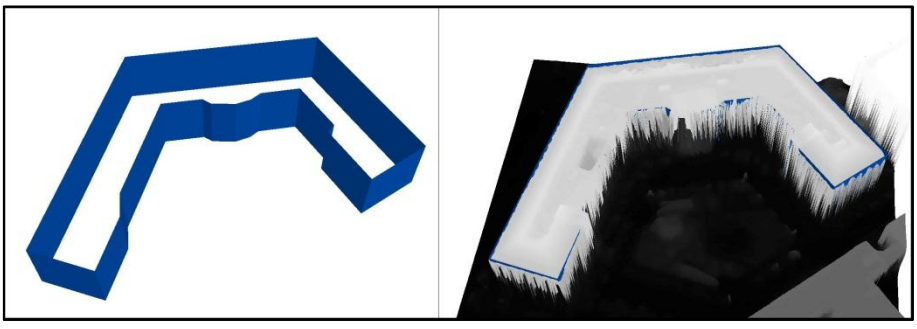

Abbildung 64: Ergebnisse des Gebäude B2 in 3D im Vergleich mit dem Nadir-DOM.

Die Ergebnisse des Gebäudes B2 sind, wie beim Gebäude B1, bei moderaten Parameteränderungen stabil.

6.3 Ergebnisse für das Gebäude B3

Für das Gebäude B3 werden alle Fassadenstrecken extrahiert und der Polygonzug, den diese Fassadenstrecken bilden, ist geschlossen (Abbildung 65, links). Die leicht gekrümmt vordere Fassade (auf der Abbildung 65: orange markiert) wird durch ein Fassadenstrecke approximiert. Die südwestliche Fassade, die aus zwei unter einem stumpfen Winkel zusammenlaufende Fassadenebenen besteht, wurde ebenfalls nur durch eine Fassadenstrecke abgebildet (in Abbildung 65: dunkelblau). Die Abweichungen zeigen sich in hohen Werten der Standardabweichung von 0,92 m für die leicht gekrümmte Fassade und in einer Standardabweichung von 0,73 m für die südwestliche Fassade.

Diese Abweichungen sind im Nadir-TrueOrthobild (Abbildung 60, rechts) deutlich zu sehen. Die Abweichung der leicht gekrümmten Fassade beträgt mit 17 Pixeln ca. 2,55 m.

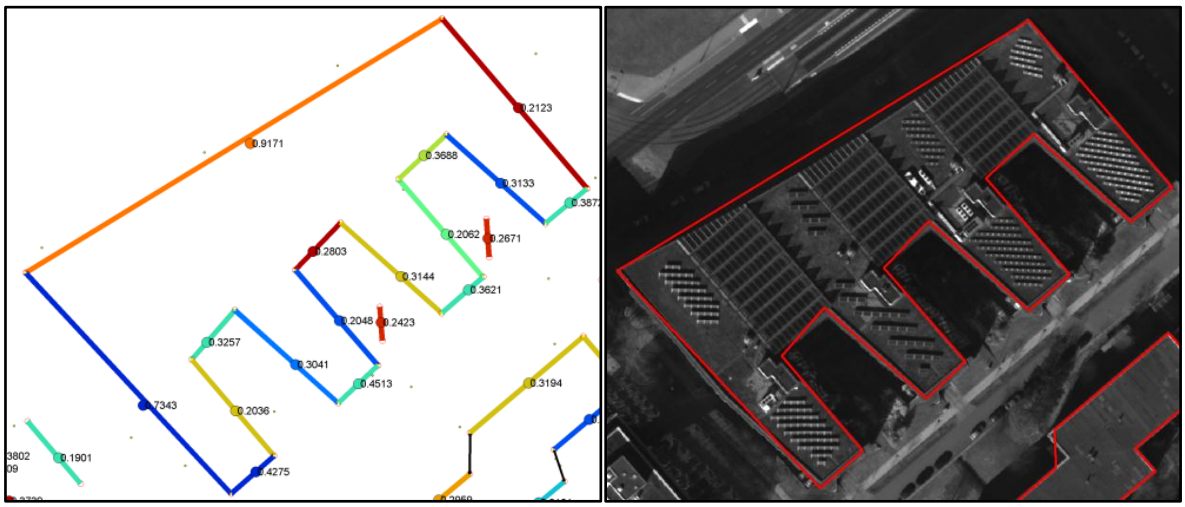

Abbildung 65: links: Extrahierte Fassadenebenen des Gebäude B3, rechts: Ergebnisse des Gebäude B3 im Vergleich mit dem Nadir-TrueOrthobild.

Generell stimmen die extrahierten Fassaden schlechter mit dem Nadir-TrueOrthobild überein, als für die Gebäude B1 und B2.

Die Ungenauigkeit der kurzen Fassaden ist durch die Datengrundlage bedingt. Die Streuung an diesen Fassaden ist wegen der Verglasung und der herausragenden Elemente groß. Die Abweichung bei der leicht gekrümmten und der südwestlichen Fassade lässt sich durch eine andere Parametereinstellung reduzieren.

Die berechnete Höhe der Dachkante beträgt 94,27 m. Die berechnete Bodenhöhe beträgt 74,24 m. Die beiden Höhen weichen von den tachymetrisch gemessenen Höhen (93,70 m und 74,5 m) um ca. 0,25 bzw. 0,60 m ab. In der Abbildung 66 werden die extrahierten Fassadenebenen des Gebäudes B3 mit dem Nadir-DOM verglichen.

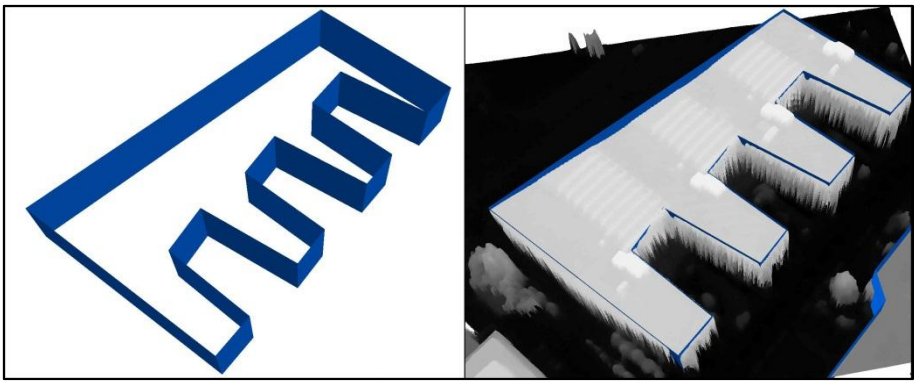

Abbildung 66: Ergebnisse des Gebäude B3 in 3D im Vergleich mit dem Nadir-DOM.

Die Ergebnisse des Gebäudes B3 sind nicht so stabil wie die von den Gebäuden B1 und B2. Das bedeutet, dass geringfügige Änderungen in der Kombination der Parameter ausreichen, und die Fassadenebenen werden möglicherweise nicht extrahiert.

Nun sollen Parametereinstellungen gefunden werden, um die Ergebnisse für das Gebäude B3 zu verbessern. Ziel ist, die leicht gekrümmte, nordwestliche Fassade und die südwestliche Fassade korrekt abzubilden.

Zuerst muss geprüft werden, welche Parameter für die ungenaue Extraktion dieser beiden Fassaden verantwortlich sind. Dazu werden die Zwischenergebnisse des Algorithmus Schritt für Schritt kontrolliert. Die Eigenvektoren und deren Richtungen (Abbildung 67, links), sehen plausibel aus: es wird ein leichter Bogen für die vordere Fassade und ein Knick bei der südwestlichen Fassade beobachtet.

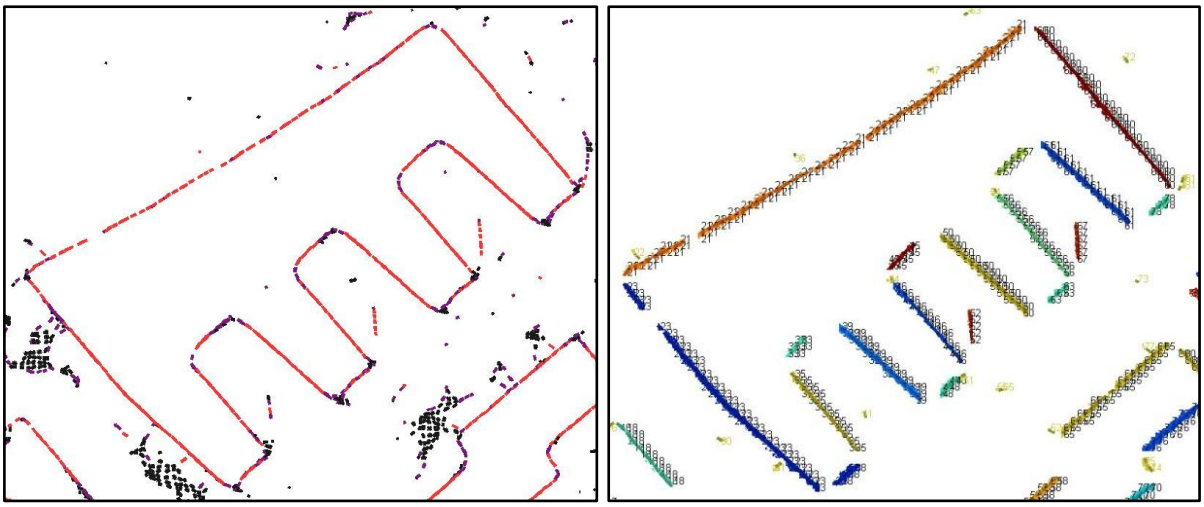

Abbildung 67: links: Eigenvektoren für das Gebäude B3 für Basiskombination von Parametern, rechts: Gruppen von Fassadenzellen farbkodiert nach Funktion `init_walls` für Basiskombination.

Der nächste Schritt ist die Initialisierung von Gruppen von Fassadenzellen. Der Schwellenwert δ_{incline} beträgt 15°. Nach der Durchführung der Funktion `init_walls` werden folgende Gruppen von Fassadenzellen erzeugt (Abbildung 67, rechts).

Es fällt auf, dass die leicht gekrümmte Fassade als eine einzige Fassade (Fassaden-ID 21, orangefarben) initialisiert ist. Die südwestliche Fassade wird auch als eine einzige Fassade (Fassaden-ID 23, dunkel blau) initialisiert. Als Konsequenz wird, bei der Erstellung von Fassadenstrecken die Regression durch alle Fassadezellen der Fassade 21 durchgeführt und der Bogen begradigt. Ähnlich wird die südwestliche Fassade begradigt. Um das zu vermeiden, muss der Schwellenwert δ_{incline} verringert werden. Dasselbe gilt für die Parameter $\delta_{\text{merge\_walls}}$, $\delta_{\text{merge\_segments}}$, die ebenfalls bestimmen, wie groß die maximale Abweichung der Richtungen zweier Fassaden sein darf, um diese zu vereinigen. Die Parameter δ_{incline} , $\delta_{\text{merge\_walls}}$, $\delta_{\text{merge\_segments}}$ werden auf 6° eingestellt.

Als Ergebnis dieser Änderungen wird die leicht gekrümmte Fassade durch vier Strecken approximiert. Die südwestliche Fassade wird korrekt durch zwei Fassadenstrecken abgebildet. Die optimierten Ergebnisse für das Gebäude B3 sind auf der Abbildung 68 (links) zu sehen.

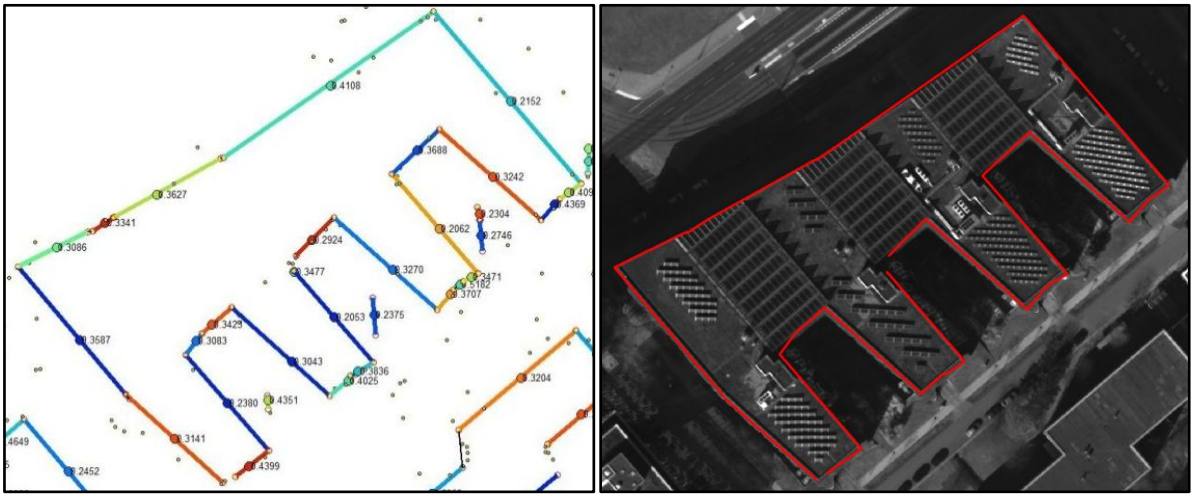

Abbildung 68: Optimierte Ergebnisse der Fassadenextraktion in 2D des Gebäude B3, rechts: optimierte Ergebnisse des Gebäude B3 im Vergleich mit dem Nadir-TrueOrthobild.

Die leicht gekrümmte nordwestliche und die südwestliche Fassade passen besser zum Nadir-TrueOrthobild als bei der Basiskombination der Parameter. Bei den optimierten Parametereinstellungen wird allerdings die Genauigkeit der kurzen Fassaden verschlechtert. Die kurzen Fassaden bestehen nun aus mehreren Fassadenstrecken (Abbildung 68, links). Die Ursache ist wie folgt. Punkte, die die kurzen Fassaden repräsentieren sind durch die externen Rettungstreppen an diesen Fassaden stark verrauscht. Die lokale Richtung von Fassadenzellen variiert demzufolge stark. In der Abbildung 69 wird eine kurze Fassade, die aus zwei Gruppen von Fassadenzellen mit der ID 68 und 71 besteht, dargestellt. Ihre Richtungen

betragen 63,5 und 49,0°. Die Differenz beträgt 14,5°. Um diese Fassaden zu vereinigen, muss der Wert des Parameters δ_{incline} mindestens 14,6° betragen. Das steht im Widerspruch zu den optimierten Parametereinstellungen.

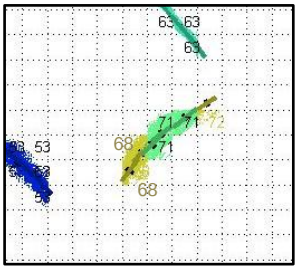

Abbildung 69: kurze Fassade, die aus den Gruppen von Fassadenzellen 68 und 71 besteht.

Das bedeutet, dass die zwei Ziele:

1. genaue Extraktion der leicht gekrümmten nordwestlichen und der mit einem leichten Knick versehenen südwestlichen Fassade des Gebäude B3
2. genaue Extraktion von den kurzen Fassaden des Gebäude B3

sich gegenseitig ausschließen. Auf Grund von der verrauschten Punktwolke, die die kurzen Fassaden repräsentiert, kann entweder die Genauigkeit der nordwestlichen und der südwestlichen Fassade oder der kurzen Fassaden gesichert werden.

6.4 Ergebnisse für das Gebäude B4

Bei der Basiskombination von Parametern für den Gebäudekomplex B4 werden nicht alle Fassaden extrahiert. Der Gebäudekomplex besteht aus einem niedrigen Gebäude B4a und einem hohen Gebäude B4c, sowie einem Verbindungsgebäude B4b (Abbildung 6). Bei dem Gebäude B4a wurde eine kurze Fassade nicht detektiert. Sie ist in der Abbildung 70 (links) als schwarze Strecke dargestellt. Diese Fassade ist aber dadurch, dass zwischen den Nachbarfassaden eine topologische Verbindung existiert, im Fassadenpolygonzug vorhanden. Vergleicht man diese topologisch erstellte Fassade mit dem Nadir-TrueOrthobild (Abbildung 70, rechts) fällt auf, dass sie geometrisch nicht zu dem Nadir-TrueOrthobild passt. Eine mögliche Ursache ist die Verdeckung durch Vegetation (siehe Abbildung 15).

Ein zweiter Fehler im Gebäude B4a besteht darin, dass dessen Polygonzug in der nordöstlichen Ecke nicht geschlossen ist. Dies wird durch die Verbindung der nordöstlichen Wand (Abbildung 70 links, ockerfarben) mit dem Verbindungsgebäude verursacht (Abbildung 70, links, blau).

Die zweite fehlende Fassade ist die nordwestliche Fassade des Verbindungsgebäudes B4b. Diese Fassade ist fast vollständig von einem Baum bedeckt (vgl. Abbildung 10, links). Die südöstliche Fassade des Verbindungsgebäudes ist ebenfalls fehlerhaft.

Die dritte fehlende Fassade ist die südöstliche Fassade des Gebäudeteils B4c. Diese besteht aus fünf kurzen Fassadenstücken (vgl. Abbildung 10, drittes Bild von links).

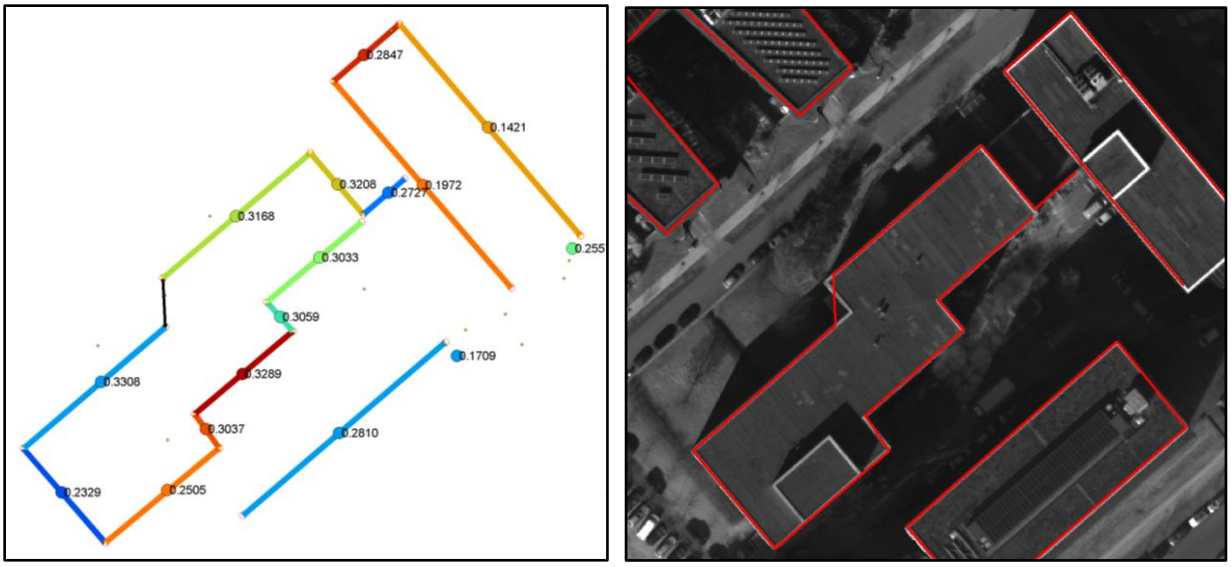

Abbildung 70: links: extrahierte Fassadenebenen des Gebäude B4, rechts: rechts: Ergebnisse des Gebäude B4 im Vergleich mit dem Nadir-TrueOrthobild.

Die berechnete Höhe der oberen Fassadenkante beträgt für den Gebäudeteil B4a 87,00 m und ist somit um 1,00 m höher, als die tachymetrisch gemessene Höhe. Die berechnete Höhe der unteren Fassadenkante beträgt 75,88 und ist somit fast 1,38 m höher als die tachymetrisch berechnete durchschnittliche Höhe. Auf der Abbildung 71 werden die 3D-Fassaden des Gebäudes B4a mit dem Nadir-DOM verglichen. Die Fassadenebene des Verbindungsgebäudes hat dieselbe Höhe wie das Gebäude B4a, da der Algorithmus diese Fassade als Teil des Gebäudes B4a identifiziert.

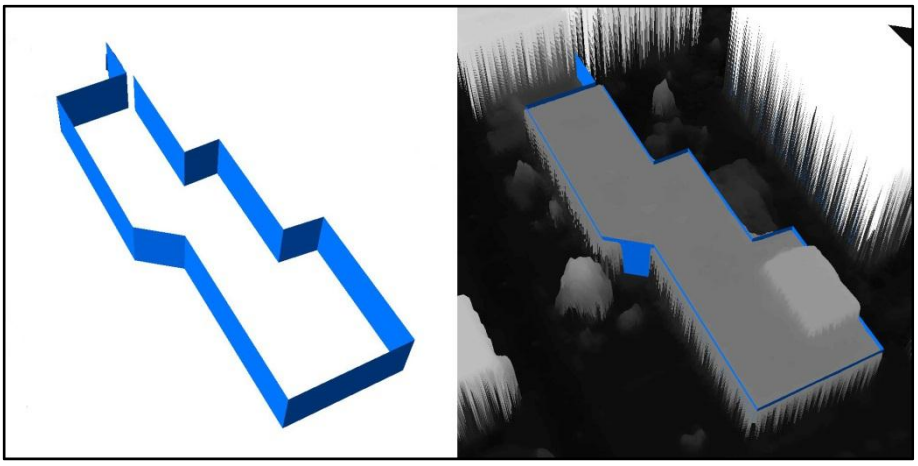

Abbildung 71: Ergebnisse des Gebäude B4a in 3D im Vergleich mit dem Nadir-DOM.

Die berechnete Höhe der oberen Fassadenkante für den Gebäudeteil B4c beträgt 96,25 m und ist um 0,95 m höher als die tachymetrisch gemessene Höhe. Die berechnete Höhe der unteren Fassadenkante beträgt 75,10 und ist 0,6 m höher als die tachymetrisch berechnete durchschnittliche Höhe. Auf der Abbildung 72 werden die 3D-Fassaden des Gebäudes B4c mit dem Nadir-DOM verglichen.

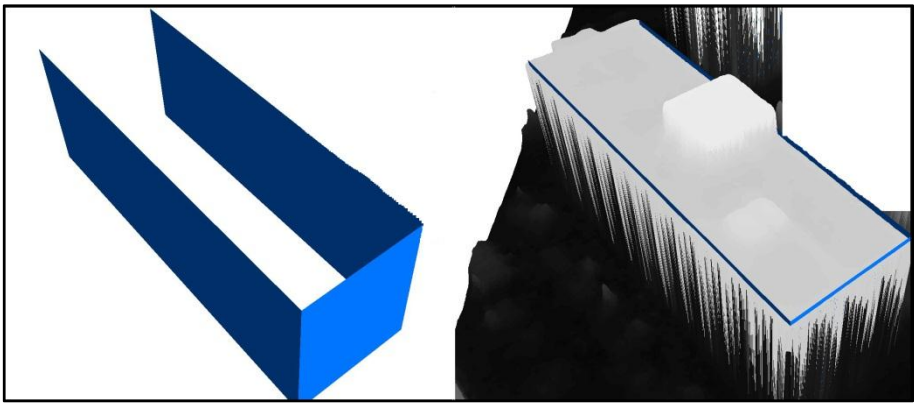

Abbildung 72: Ergebnisse des Gebäude B4c in 3D im Vergleich mit dem Nadir-DOM.

Nun sollen verbesserte Parametereinstellungen zur Extraktion der Fassaden des Gebäudekomplexes B4 gefunden werden. Besonders soll die fehlende, südöstliche Fassade des Gebäudesteil B4c extrahiert werden. Diese Fassade besteht aus fünf kleinen Fassadenstücken. Um kleine Fassaden zu extrahieren, sollte in erster Linie die Fenstergröße verkleinert werden. Parallel dazu sollte der Wert von $\varrho_{\max}$ weniger streng eingestellt werden. Er soll also vergrößert werden. Es werden folgende Einstellungen vorgenommen: Fenstergröße $l_{\text{interval}} = 1{,}3$ m und $\varrho_{\max} = 0{,}4$. Zusätzlich werden Parameter $n_{\text{segment\_min\_cells}}$ und $l_{\text{segment\_min\_length}}$ kleiner eingestellt um zu verhindern, dass kurze Fassadenstücke vom Algorithmus entfernt werden. In Abbildung 73 ist zu sehen, dass die fehlende Fassade extrahiert wurde.

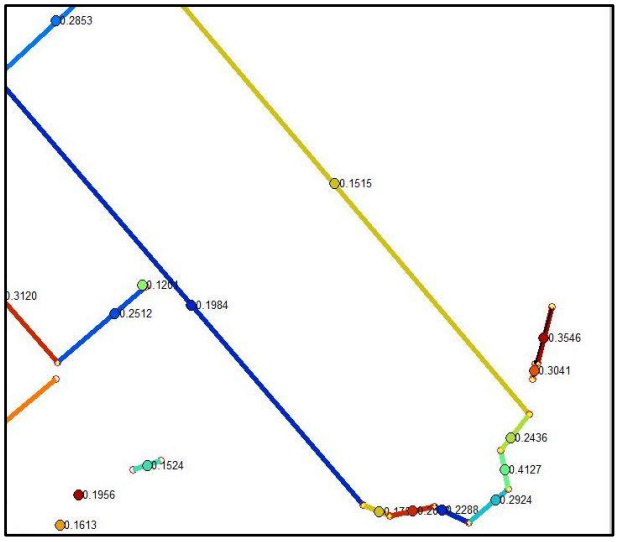

Abbildung 73: Extraktion von der südöstlichen Fassade des Gebäudes B4c.

Allerdings führen die neuen Parametereinstellungen dazu, dass eine kurze Fassade des Gebäudeteils B4a (Abbildung 74) nicht mehr extrahiert wird.

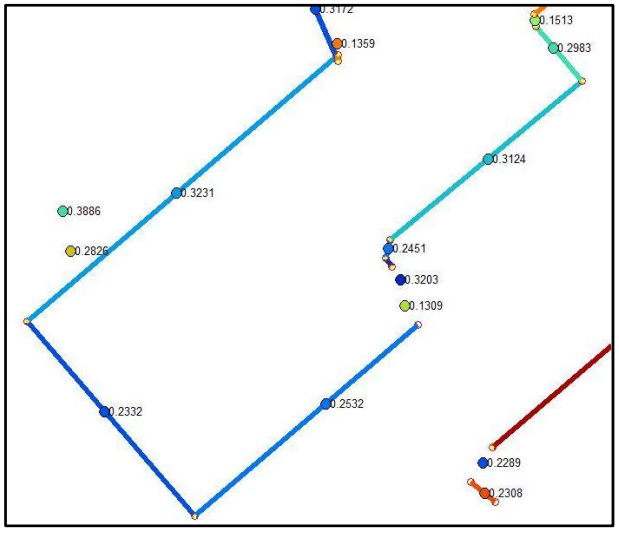

Abbildung 74: Fehlende Fassade im Gebäude B4a bei den neuen Eistellungen von Parametern.

Es wird nun geprüft, warum diese Fassade nicht extrahiert wird. Diese Fassade wird durch sechs Zellen repräsentiert (auf der Abbildung 75 rosa markiert). Die Richtungen $v_{\max}$ formen einen Bogen, der sich mit den Zellen der benachbarten nördlichen Wand verbindet. Durch den mit 15° relativ groß eingestellten Parameter $\delta_{\text{init\_walls}}$ werden die 6 Fassadenzellen mit den Fassadenzellen der nördlichen Wand zu einer Gruppe von Fassadenzellen verbunden. Um dies zu verhindern wird der Parameter $\delta_{\text{init\_walls}}$ auf 10° reduziert.

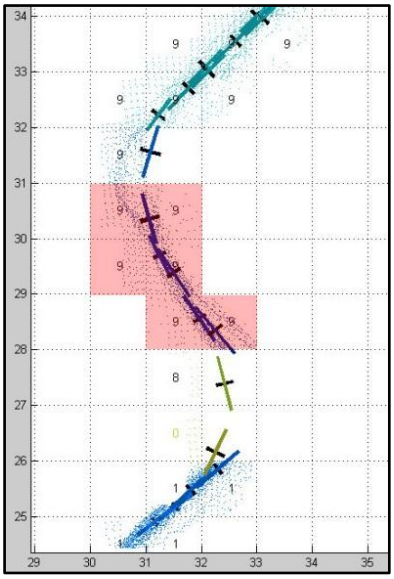

Abbildung 75: Fassadenzellen der fehlenden Fassade des Gebäudesteil B4a.

Dies führt zur erfolgreichen Extraktion diese Fassade. Die südöstliche Fassade des Gebäudeteils B4c wird weiterhin extrahiert. Die optimierten Ergebnisse für den Gebäudekomplex B4 werden in der Abbildung 76 (links) dargestellt.

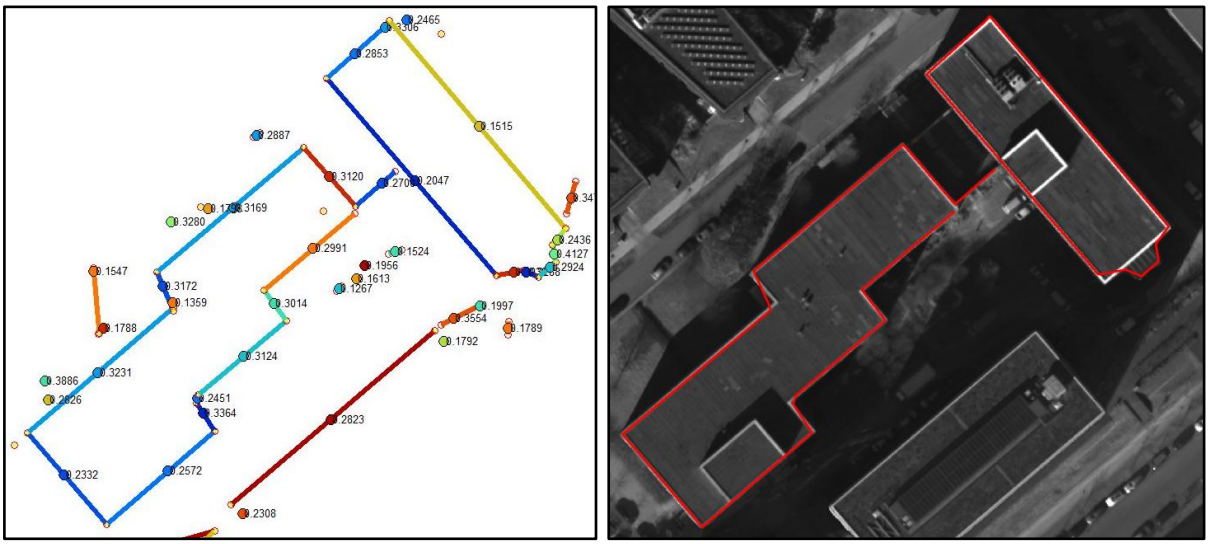

Abbildung 76: links: optimierte Ergebnisse der Fassadenextraktion in 2D des Gebäude B4, rechts: Vergleich mit Nadir-TrueOrthobild.

In der Abbildung 76 (rechts) werden die optimierten Ergebnisse des Gebäudes B4 mit dem Nadir-TrueOrthobild zusammengestellt. Die südöstliche Fassade des Gebäudeteils B4c weist Abweichungen auf.

6.5 Rechenzeit

Zur Einschätzung der Anwendbarkeit der entwickelten Algorithmen für praktische Berechnungen soll die Rechenzeit für ein Beispiel analysiert werden. Für das Gebäude B2 aus ca. 2.200.000 Punkten beträgt die Berechnungszeit 78 Sekunden. Die gefilterte Punktwolke enthält dabei ca. 330.000 Fassadenpunkte. Die einzelnen Funktionen der Hauptfunktion `main` werden nach der benötigten Rechenzeit sortiert (Abbildung 77).

Function Name	Function Type	Calls	Total Time	% Time	Time Plot
calculate_height	function	1	31.884 s	40.7%	
z_filter	function	1	25.990 s	33.2%	
make_cloud_in_cells	function	2	14.343 s	18.3%	
make_segments	function	2	5.001 s	6.4%	
eigenvectors_full_cells	function	1	0.327 s	0.4%	
plots3d	function	1	0.249 s	0.3%	
init_walls	function	1	0.234 s	0.3%	
find_pairs	function	1	0.047 s	0.1%	
make_buildings	function	1	0.032 s	0.0%	
merge_walls	function	1	0.032 s	0.0%	
merge_gaps	function	1	0.016 s	0.0%	
find_corner	function	1	0.015 s	0.0%	
Self time (built-ins, overhead, etc.)			0.126 s	0.2%	
Totals			78.295 s	100%	

Abbildung 77: Übersicht über die Rechenzeit der einzelnen Algorithmen.

Die meisten Zeit wird von der Funktion `calculate_height` benötigt (40 % der Rechenzeit). Die Filterung nach Z-Werten und die Initialisierung der Arrays von Zellen benötigen zusammen ca. 51% der Rechenzeit. Eine genauere Analyse ergibt, dass diese Rechenzeiten hauptsächlich durch das Hinzufügen von Elementen zu Arrays verursacht werden. Die vergleichsweise hohe Rechenzeit beim Erweitern von Arrays ist eine Eigenschaft von MatLab.

Die Berechnung der lokalen Regression und die Bestimmung der Fassadenstrecken als geschlossenes Polygon benötigen mit 5,3 Sekunden ca. 6 % der Rechenzeit. Davon benötigt die Bestimmung der Fassadenstrecken mit 5,0 Sekunden den größten Anteil und hat damit das größte Optimierungspotenzial. Der größte Anteil der Rechenzeit wird hierbei durch die Projektion der Punkte auf die Ausgleichsgerade verursacht.

Die Berechnungen wurden auf Windows 7 mit Intel i7-Prozessor, 2,6 GHz auf einer CPU durchgeführt. Da alle vorgestellten Algorithmen lokale Informationen verwenden, ist eine Aufteilung der Berechnungen auf mehrere Gebiete oder CPUs möglich.

7 Fazit und Ausblick

Die Ergebnisse dieser Arbeit zeigen, dass eine effiziente, vollautomatische Extraktion von Fassadenebenen durch lokale Regression für hochaufgelöste 3D-Punktwolken möglich ist. Allerdings hängt deren Qualität von der Komplexität der Gebäudegrundrisse und der Qualität der Punktwolke ab. Eine fehlerfreie Extraktion ist mit der in dieser Arbeit entwickelten Methodik nicht zu erwarten. Die entwickelte Methode wurde auf frei stehende Gebäude mit Flachdächern angewändet. Eine Anwendung in eng bebauten städtischen Gebieten steht aus.

Mit der entwickelten Methode werden sowohl planare senkrechte Fassaden, als auch gekrümmte senkrechte Fassaden, automatisch extrahiert. Letzte werden durch mehrere Fassadenebenen approximiert. Alle extrahierten Fassadenstrecken werden sortiert und als geschlossener Polygonzug exportiert. Als Ergebnis wird der Grundriss der Gebäude im zweidimensionalen Koordinatensystem erzeugt. Außerdem werden die absoluten Höhen der unteren und der oberen Fassadenkante eines jeden Gebäudes im dreidimensionalen Koordinatensystem bestimmt.

Diese Methode wurde an vier Gebäuden unterschiedlicher Komplexität getestet. Dabei bestand das Ziel darin, die Fassadenextraktion mit globalen Parametereinstellungen für alle Gebäude durchzuführen. Bei einer festen Kombination von voreingestellten Parametern werden bis auf fünf Fassaden alle Fassaden der vier getesteten Gebäude erkannt. Besonders gute Ergebnisse werden für Gebäude mit hohen Fassaden erzielt, die nicht durch Vegetation oder Anbauten verschattet werden. Problematisch sind kurzen Fassaden oder Fassaden, bei denen die Punktwolke durch Anbauten eine starke Abweichung von der Fassadenebene hat. Mit individuell angepassten Einstellungen konnten die fehlenden Fassaden (bis auf eine) erkannt werden.

Bei den berechneten Höhen der unteren und oberen Fassadenkante lassen sich Abweichungen von den tachymetrisch gemessenen Höhen von 0,13 m bis zu 1,85 m beobachten. Eine Analyse der Z-Werte der 3D-Punkte in Fassadennähe zeigt, dass die eine erhebliche Anzahl von Punkten oberhalb der Fassadenoberkante liegt. Dies deutet auf Fehler in der Datengrundlage hin.

Die Methode ist empfindlich gegenüber der Streuung der 3D-Punkte in Richtung normal zur Fassadenebene. Ist diese Streuung groß, wird die Fassadenextraktion erschwert. Ein Lösungsansatz ist die Vergrößerung der Breite des Regressionsfensters. Dadurch werden aber

Fassaden, deren horizontale Länge kleiner ist als die Breite des Regressionsfensters, nicht mehr extrahiert.

Eine Idee für die weitere Entwicklung des Algorithmus ist die automatische, adaptive Einstellung von Parametern. So soll für jeden Gebäudeteil die jeweils eine optimale Einstellung gewählt werden. Eine weitere Entwicklungsmöglichkeit ist die Entwicklung eines Kriteriums, welches prüft, ob die Fassaden richtig extrahiert wurden. Beispielsweise könnte geprüft werden, ob das Gebäudepolygon geschlossen ist oder ob die Standardabweichungen der berechneten Fassaden kleine Werte annehmen. Falls das Kriterium nicht erfüllt ist, sollte die Nutzer darüber informiert werden. Der Nutzer könnte die fehlenden oder fehlerhaft extrahierten Fassaden selbst digitalisieren oder eine Einstellung von Parametern finden, bei denen diese Fassaden extrahiert werden.

Entscheidend für die Extraktion von Fassaden ist die vorherige Entfernung von Punkten, welche außerhalb von Fassaden liegen. Um diese zu eliminieren wird ein Z-Filter entwickelt, der die Boden- und Dachpunkte entfernt (vgl. Kapitel 4.5). Der Filter benötigt keine globalen Parameter und erkennt Fassaden von Gebäuden mit unterschiedlicher Höhe. Der Filter sollte jedoch weiterentwickelt werden, da geringe Mengen von Objekten, die zu keiner Fassade gehören, verbleiben. Die entsprechenden Punkte erschweren die Extraktion von Fassaden. Eine mögliche Erweiterung des Filters könnte darin bestehen, dass die Identifikation der Maxima verfeinert wird, beispielsweise dadurch, dass die Maxima deutlich die anderen Klassen des Histogramms dominieren.

Die Grundlage für die Berechnung der Fassadenecken (Kapitel 5.3.3) und für die Vereinigung von Fassadenstrecken (Kapitel 5.3.2) ist die Bestimmung der Nachbarschaft zwischen den Fassadenstreckenenden (Kapitel 5.3.1). In der aktuellen Version des Algorithmus wird für jedes Ende einer Fassadenstrecke im Umkreis von 10 m das nächstlegende Ende einer benachbarten Fassadenstrecke gesucht. Diese Suche ist fehleranfällig und führt besonders bei falsch positiv detektierten Fassadenzellen zu Fehlern, die alle weiteren Berechnungen verfälschen. Eine Lösung für das Problem kann darin bestehen, die Berechnung der Fassadenstrecken zu verbessern. Bisher werden die Fassadenstrecken nur in dem Bereich der Fassade bestimmt, der linear ist. Die Fassadenstrecken haben dadurch einen Abstand von bis zu 4 m zur Fassadenecke. Die Verbesserung besteht darin, die Fassadenstrecken mit einem zu entwickelnden Algorithmus zu verlängern. Dies verursacht, dass die benachbarten Enden von Fassadenstrecken sich nähern. Damit kann zur Bestimmung der Nachbarschaft ein kleinerer Suchradius verwendet werden. Eine weitere Verbesserung könnte darin bestehen, zu fordern, dass die benachbarten Fassadenstrecken dieselbe Höhe haben.

Zur Bestimmung der absoluten Höhe der unteren und oberen Fassadenkante wird ein Algorithmus entwickelt, dessen Voraussetzung es ist, dass jede Fassade eine konstante Höhe aufzuweisen. Für zwei Gebäude verschiedener Höhen mit einer gemeinsamen Front, wird der Algorithmus die Höhe der höchsten Fassade berechnen. Dies ist ebenfalls ein Punkt für die Weiterentwicklung.

8 Quellenverzeichnis

Bahrenberg, G., Giese, E., Nipper, J. (1990): Statistische Methoden in der Geographie. Band 1. Univariate und bivariate Statistik.- Teubner Studienbücher der Geographie, Stuttgart.

Brenner C., H. N. (2001): Towards fully automated 3D city model generation.- In: Automatic Extraction of Man-Made Objects from Aerial and Space Images III.

Brenner, C. (1999): Interactive Modeling Tools for 3D Building Reconstruction.- Photogrammetric Week, Band 99, Heidelberg, Wichmann, S. 22-23.

Derpanis, K. (2010): Overview of the RANSAC Algorithm.- Online: http://www.cse.yorku.ca/~kosta/CompVis_Notes/ransac.pdf [Stand 26.09.2012].

DLR (2012): Modulare Luftbildkamerasysteme.- Online: http://www.dlr.de/os/desktopdefault.aspx/tabid-8078/12173_read-28835/ [Stand 01.10.2012].

Dorninger, P., Nothegger, C. (2007): 3D segmentation of unstructured point clouds for building modelling. – In: Stilla, U. et al. (Hrsg.) PIA07. International Archives of Photogrammetry, Remote Sensing and Spatial Information Sciences, Band 36, S. 191-196, Online: http://www.pf.bv.tum.de/isprs/pia07/puba/PIA07_Dorninger_Nothegger.pdf [Stand: 15.05.2012].

ESRI, (2011): ArcGIS Resource Center.- Online: http://help.arcgis.com/de/arcgisdesktop/10.0/help/ [Stand: 24.09.2012].

Fischler, M., Bolles, R. (1981): Random Sample Consensus: A Paradigm for Model Fitting with Apphcatlons to Image Analysis and Automated Cartography.- In: Communications of the ACM, Band 6,S. 381-395, Online: http://www.ai.sri.com/pubs/files/836.pdf [Stand: 24.9.2012].

Gellert, W., Küstner, H., Hellwich, M., Kästner, H. (Hrsg.) (1986): Kleine Enzyklopädie Mathematik.- VEB Bibliographisches Institut Leipzig, Leipzig.

Gröger, G., Benner, J., Dörschlag, D., Drees, R., Gruber, U., Leinemann, K., Löwner, M.-O. (2005): Das interoperable 3D-Stadtmodell der SIG 3D - In: Zeitschrift für Geodäsie, Geoinformation und Landmanagement (zfv), Nummer 06/2005, Wißner Verlag, Augsburg, S. 343-354.

Hirschmüller, H. (2008): Stereo Processing by Semi-Global Matching and Mutual Information. – In: IEEE Transactions on Pattern Analysis and Machine Intelligence, Band 30, Nr. 2, 2008, S. 328-341.

Hoppe, H., DeRose, T., Duchamp, T., McDonald, J., Stuetzle, W. (1992): Surface Reconstruction from Unorganized Points.- In: ACM SIGGRAPH 1992 Proceedings, S. 71-78, Online: http://research.microsoft.com/en-us/um/ people/hoppe/recon.pdf [Stand: 29.08.2012].

IGI mbH (2006): AeroControl.- Online. http://www.igi.eu/aerocontrol.html [Stand: 26.09.2012].

INPHO GmbH (2006): MATCH-AT 5.3 – Reference Manual.- Stuttgart.

Konecny, K. (2011): Evaluierung der Nutzbarkeit von Punktwolken aus Schrägluftbildern zur Kartierung von Grundrisselementen.- Masterarbeit, Beuth Hochschule für Technik Berlin, Berlin.

Kraus, K. (2004): Photogrammetrie, Band 1.- Walter de Gruyter, Berlin.

Lehmann, F., Berger, R., Brauchle, J., Hein, D., Meissner, H., Pless, S., Strackenbrock, B., Wieden, A. (2011): MACS – Modular Airborne Camera System for generating photogrammetrichigh-resolution products. – In: Photogrammetrie, Fernerkundung, Geoinformation. Nummer 6/2011, E. Schweizerbartsche Verlagsbuchhandlug, Stuttgart, S. 423-434.

Laible, S. (2008): Klassifikation von Pflanzen anhand 3D-Laserscanner-Daten mittels maschineller Lernverfahren.- Online: http://www.ra.cs.uni-tuebingen.de/mitarb/laible/plant_ classification.pdf [Stand 21.08.2012].

MeshLab (2011): MeshLab.- Online. http://meshlab.sourceforge.net/ [Stand: 25.09.2012].

Mitra, N., Nguyen, A. (2004): Estimating Surface Normals in Noisy Point Cloud Data.- Special issue of Int. J. Computational Geometry and its Applications, Band 14 (4-5), S. 261-276, Online: http://graphics.stanford.edu/projects/lgl/papers/mn-esnnpcd-03/mn-esnnpcd-03.pdf [Stand: 08.09.2012].

Nagel, C., Häfele, K.-H. (2007): Generierung von 3D-Stadtmodellen auf Basis des IFC-Gebäudemodells.- In: Entwicklerforum Geoinformationstechnik 2007 - Junge Wissenschaftler forschen, S. 151-165, Online: http://www.redaktion.tu-berlin.de/fileadmin/fg227/ Publications/entwicklerforum_nagel.pdf [Stand: 28.08.2012].

Pearson, K. (1901): On lines and planes of closest fit to systems of points in space.- Philosophical Magazine, S. 559-572, Online: http://stat.smmu.edu.cn/history/pearson1901.pdf [Stand 21.08.2012].

Schweizer, W. (2009): MATLAB kompakt.– Oldenburg Wissenschaftsverlag, München.

Wieden, A. (2012): Ableitung von Geoinformationen aus Schrägluftbildern. Ansätze zur Gebäuderekonstruktion.- Masterarbeit, UNIGIS am Zentrum für GeoInformatik der Paris Lodron-Universität Salzburg, Salzburg.

Wikipedia, (2012): RANSAC-Algorithmus.- In: Wikipedia, Die freie Enzyklopädie, Online: http://de.wikipedia.org/w/index.php?title=RANSAC-Algorithmus&oldid=109058534 [Stand: 28.10.2012].

Anhang

A 1. Formelverzeichnis

Parameter der Basiskombination für alle Gebäude

Symbol	Funktion	Kap.	Wert	Name im Programm
$l_{\text{z\_filter}}$	Breite des Quadratgitter für den z-Filter	4.5	0,5 m	interval_filter
l_{interval}	Breite des Quadratgitters für lokale Fassadenrichtungen	4.4	1,6 m	interval_width
ϱ_{max}	max. Schwellenwert für lokale Regression	4.3	0,3	ratio_threshold
$n_{\text{nbhd\_regression}}$	Breite der Nachbarschaft für lokale Regression (z.B. 1 für 3x3-Umgebung)	4.4	1	delta
$n_{\text{min\_points}}$	min. Anzahl der Punkte in der Zelle für lokale Regression	4.4	10	minimum_number_of_points_in_cell
$h_{\text{z\_filter}}$	min. Abstand zwischen Maxima im Histogramm	4.5	3 m	z_filter_min_height
$n_{\text{z\_filter}}$	max. Anzahl von leeren Histogrammklassen	4.5	0	z_filter_max_count_null_classes
$\delta_{\text{init\_walls}}$	max. Abweichung zw. Fassadenrichtungen für Vereinigung von Gruppen	5.1.1	15°	init_walls_max_incline_difference
$\delta_{\text{merge\_walls}}$	max. Abweichung zw. Fassadenrichtungen für Vereinigung von Gruppen	5.1.2	10°	merge_walls_max_incline_difference
$n_{\text{segment\_min\_cells}}$	min Anzahl von Zellen einer Gruppe von Fassadenzellen	5.2	3	segment_min_cell_number
$l_{\text{segment\_min\_length}}$	min. Länge eines Fassadensegments	5.2	2 m	segment_min_length
$l_{\text{max\_pair\_distance}}$	max. Suchumgebung für benachbarte Fassadensegmente	5.3.1	10 m	pairs_max_distance
$\delta_{\text{merge\_segments}}$	max. Abweichung zwischen Fassadensegmenten für Vereinigung von Gruppen	5.3.2	10°	merge_gaps_max_incline_difference
$l_{\text{merge\_segments}}$	max. paralleler Abstand zw. Fassadensegmenten für Vereinigung von Gruppen	5.3.2	1 m	merge_gaps_max_parallel_offset

$\delta_{\text{min\_corners}}$	min. Abweichung zwischen Fassadensegementen für Berechnung von Eckpunkten	5.3.2	10°	find_corners_min_incline_difference

Verwendete Formelsymbole

Bezeichnung	**Symbol**
Standardabweichung	σ
minimale Standardabweichung einer Punktwolke	$\sigma_{\min}$
maximale Standardabweichung einer Punktwolke	$\sigma_{\max}$
Richtung der kleinsten Streuung einer Punktwolke	$v_{\min}$
Richtung der größten Streuung einer Punktwolke	$v_{\max}$
kleinster Eigenwert der Kovarianzmatrix	$\lambda_{\min} := \sigma_{\min}^2$
größter Eigenwert der Kovarianzmatrix	$\lambda_{\max} := \sigma_{\max}^2$
Koeffizient als Maß für die Linearität einer Punktwolke	$\varrho := \frac{\sigma_{\min}}{\sigma_{\max}}$

B 1. main

```
function  [cloud_filter,  cells, walls, buildings] = ...
    main(  cloud, interval_filter, interval_width, ratio_threshold, delta)

% author:   Magdalena Linkiewicz
%
% Masterarbeit 2012, Studiengang Geoinformationsysteme,
% Extraktion von senkrechten Fassadenebenen aus 3D-Punktwolken
% von Schrägluftbildern
%
% version:  1.0
%
% name:     main
%
% description:
% Main function, calling subsequent functions.
% Defines main parameters for controlling sub functions.
% The main idea is to subdivide the 3D point cloud by a grid
% into an two dimensional array of cells.
% For each cell it is calculated, if the cell belongs to a wall
% by local regression. Cells not belonging to a wall will
% be ignored for the remainder of the program.
% For each cell the local direction of the wall is defined as angle.
% Neighbouring cells with similar angle will be grouped into a
% array "walls". Regression analysis is done for each wall, defining
% a segment (a line restricted to the wall). Segments will be
% joined further and intersection
% between neighbouring segments is calculated. This results in
% a closed polygon which will define a "building".
% Additionally the calculation of the lower and upper border of
% the wall is done.
%
%
% input
% cloud:             3D point cloud to be processed
% interval_filter:   grid size in m for z-filtering
% interval_width:    grid size in m for local regression
% ratio_threshold:   upper threshold for local cloud to be accepted
%                    as linear
% delta:             width of cell neighboorhood taken as local regression
%                    window
%
% output
% cloud_filter:      3D point cloud after z-filtering
% cells:             two dimensional array of cells of grid used for
%                    regression analysis
% walls:             array of array of cells belonging to the same wall
% buildings:         array of array of points defining a building
%
% Note that notation differs between documention and code.
% In the documentation the term "facade" used for "wall" in the code.
%
% example
% [cloud_filter,  cells, walls, buildings] = ...
%                           main(  b2_gesamt, 0.5, 1.6, 0.3, 1 )


% parameters to control extraction of facades
```

```
% minimum distance between maxima in histogram of z-values
z_filter_min_height = 3.0;

% maximum number of empty histogram classes between maxima of histogram
z_filter_max_count_null_classes = 0;

% mininum number of points in cell to calculate local regression
minimum_number_of_points_in_cell = 10

% maximum angle deviation in degree for cells to be joined into one group
init_walls_max_incline_difference = 15;

% maximum angle deviation in degree for group of cells to be joined
merge_walls_max_incline_difference = 10;

% minimum required number of cells in wall,
% wall with less cells will be ignored
segment_min_cell_number = 3;

% minimum length of wall segment in m,
% wall with shorter segment will be ignored
segment_min_length = 2.0;

% maximum search radius in m for finding neighbouring segments
pairs_max_distance = 10.0;

% maximum angle deviation in degree between segments
% for segments to be joined
merge_gaps_max_incline_difference = 10;

% minimum parallel offset of segments in m
% for segments to be joined
merge_gaps_max_parallel_offset = 1.0;

% minimum angle deviation for segments
% in order to perform calculation of intersection
find_corners_min_incline_difference = 10.0;

% define cells for z-filter
[cells] = make_cloud_in_cells( cloud, cloud, interval_filter);

% apply z-filter to remove points outside facades
[cloud_filter] = ...
    z_filter( cells, z_filter_max_count_null_classes, z_filter_min_height);

% define cells for facade extraction
[cells] = make_cloud_in_cells( cloud_filter, cloud_filter, interval_width);

%  local regression analysis on cells, determine local wall direction
[cells] = eigenvectors_full_cells( cells, ratio_threshold, delta, ...
                        minimum_number_of_points_in_cell);

% init walls structure, each wall contains cells with similar wall
% direction
[cells, walls] = init_walls( cells, init_walls_max_incline_difference);

% merge neighbouring walls
[cells, walls] = ...
    merge_walls( cells, walls, merge_walls_max_incline_difference);
```

```
% calculate wall segment by local regression
[walls] = make_segments( cells, walls, segment_min_cell_number, ...
                         segment_min_length, false);

% define neighbourhood relation for walls
[walls] = find_pairs( walls, pairs_max_distance);

% merge walls
[walls] = merge_gaps( walls, merge_gaps_max_incline_difference, ...
                         merge_gaps_max_parallel_offset);

% calculate facade corners by intersecting walls
[walls] = find_corner( walls, find_corners_min_incline_difference);

% calculate height of each wall
[walls] = calculate_height( cloud, cloud_filter, walls, interval_width);

% combine walls to building and calculate height of building
[walls, buildings] = make_buildings(walls);

end
```

B 2. make_cloud_in_cells

```
function [cells] = make_cloud_in_cells(cloud, base_cloud, interval_width )

% author:   Magdalena Linkiewicz
%
% Masterarbeit 2012, Studiengang Geoinformationsysteme,
% Extraktion von senkrechten Fassadenebenen aus 3D-Punktwolken
% von Schrägluftbildern
%
% version:  1.0
%
% name:     make_cloud_in_cells
%
% purpose:  Defines a regular grid of cells
%
% Given by a rectangular domain the bounding box of the grid is defined.
% Cells are defined by a two dimensional array.
% Points with X,Y-coordinates in the rectangular domain are added to cells.
%
% input
% cloud:            3D point cloud to be processed
% base_cloud:       3D point cloud defining a rectangular domain
% interval_width:   grid size in m
%
% output
% cells:            two dimensional array of cells
%                   each cell is a structure with fields
%
%
% example:  [cells] = make_cloud_in_cells(cloud_filter, cloud_filter, 1.6);

% calculate coordinates of rectangular domain as bounding box of base_cloud
min_x = min(base_cloud(:, 1));
```

```
max_x = max(base_cloud(:, 1));

min_y = min(base_cloud(:, 2));
max_y = max(base_cloud(:, 2));

offset_x = min_x;
offset_y = min_y;

% number of cells in x and y direction
count_interval_x = ceil((max_x-min_x)/interval_width);
count_interval_y = ceil((max_y-min_y)/interval_width);

% define cell structure
cells(count_interval_x, count_interval_y)= struct( 'points', [],  ...
'num_points', [],'mean', [], 'center', [], 'ratio',[], 'v_long_scal',[],...
'v_short_scal', [], 'incline', [], 'is_wall', [], 'wall_id', []);

% init cells
for k = 1:count_interval_x
    for l = 1:count_interval_y
        cells(k,l).is_wall = false;
        cells(k,l).wall_id = 0;
        cells(k,l).v_long_scal = 0;
        cells(k,l).v_short_scal = 0;
        cells(k,l).incline = 0;
        cells(k,l).mean = 0;
        cells(k,l).ratio = 0;
        cells(k,l).center=[];
        % preallocated memory for speed
        % adding NaN is save, because NaNs is not allowed
        % in further computation
        cells(k,l).points = nan(500,3);
        cells(k,l).num_points = 0;
    end
end

% assign center coords. for cells
for k = 1:count_interval_x
    for l = 1:count_interval_y
        cells(k,l).center = ...
            [min_x+(interval_width/2)+(k-1)*interval_width; ...
             min_y+(interval_width/2)+(l-1)*interval_width];
    end
end

% assembly of cells
[length_cloud] = size(cloud,1);
for i = 1:length_cloud

    x = cloud(i,1); y = cloud(i,2); z = cloud(i,3);

    index_cells_x = ceil((x-offset_x)/interval_width);
    index_cells_y = ceil((y-offset_y)/interval_width);

    % error case: prevent indices with value 0
    % occurs for the 2 points defining the minimum value for x and y
    if (index_cells_x <= 0) || (index_cells_y <= 0)
        continue;
    end
```

```
    % error case: check that index is inside bounds
    if (index_cells_x>count_interval_x) || (index_cells_y>count_interval_y)
        continue;
    end

    % number of points in cell
    cells(index_cells_x, index_cells_y).num_points = ...
                    cells(index_cells_x, index_cells_y).num_points+1;
    np = cells(index_cells_x, index_cells_y).num_points;

    % add point to cell
    cells(index_cells_x, index_cells_y).points(np,:) = [x y z];

end

% remove unnecessary preallocated memory
for k = 1:count_interval_x
    for l = 1:count_interval_y
        np = cells(k,l).num_points;
        capacity = size(cells(k,l).points,1);
        cells(k,l).points(np+1:capacity,:) = [];
    end
end

 end
```

B 3. z_filter

```
function [cloud_filter] =...
    z_filter(cells, max_count_null_classes, min_height)

% author:   Magdalena Linkiewicz
%
% Masterarbeit 2012, Beuth Hochschule fÃ¼r Technik,
% Studiengang Geoinformationsysteme
%
% Extraktion von senkrechten Fassadenebenen aus 3D-Punktwolken
% von Schrägluftbildern
%
% version: 1.0
%
% name:    z_filter
%
% purpose: Filters out points from 3D point cloud not belonging to facades.
%
% Uses histogram of z-values for points in the neighbourhood of a cell.
% Condition: histogram has to have two maxima with a minimum distance.
% If a histogram of a cell fails to fulfill the condition, the cell is
% ignored.
% If a histogram fulfills the conditions a lower and upper threshold for
% z-values is calculated based on the 2 maxima. Points of the cell between
% these thresholds are added to output argument cloud filter.
%
% input
% cloud:        3D point cloud to be processed
% max_count_null_classes:
%               maximum number of empty histogram classes between
%               maxima of histogram
```

```
% min_height:   minimum distance between maxima in histogram of z-values
%
% output
% cloud_filter: 3D point cloud containing points of cloud belonging to
% facades
%
% example:  [cloud_filter] = z_filter(cells, 0, 3.0);

% number of histogram classes
class_count = 10;


% width of neighbourhood for calculation of histogram
delta = 1;

cloud_filter=[];

[length_cells, width_cells] = size(cells);

% loop over all cells of the cells
for k = 1:length_cells
    for l = 1:width_cells

        % get points from cell neighbourhood
        points = get_neighbourhood_points( cells, k, l, delta );

        if (numel(points) == 0)
            continue;
        end

        % analyze z coordinate by histogram
        points = points(:,3);

        [class_points, class_value] = hist(points, class_count);

        hist_matrix=[class_points; class_value]';

        % center class number of classes in histogram
        center = ceil(class_count/2);

        % search max on the left side (ground)
        [max_left, index_left] = max(hist_matrix(1:center,1));
        z_left = hist_matrix(index_left,2);

        % search max on the right side (roof)
        [max_right,index_right] = max(hist_matrix(center+1:class_count,1));
        index_right = index_right + center;
        z_right = hist_matrix(index_right,2);


        % skip if there are too much classes without
        % points between max left and max right
        wall_classes = hist_matrix(index_left:index_right,1);
        [wall_classes_length]=size(wall_classes, 1);

        count_null_class = 0;
        for i= 1:wall_classes_length
            if wall_classes(i) == 0
                count_null_class = count_null_class + 1;
```

```
            end
        end

        if count_null_class > max_count_null_classes
            continue;
        end

        % calculate lower and upper threshold for filter
        class_width= hist_matrix(2,2) - hist_matrix(1,2);

        z_min = z_left + class_width/2;
        z_max = z_right - class_width/2;

        % skip when z_min smaller then z_min_real
        if z_max-z_min < min_height
            continue;
        end

       % save wall points in cell
       wall_points = cells(k,l).points((cells(k,l).points(:,3) >= z_min)...
            & (cells(k,l).points(:,3) <= z_max),:);
         cloud_filter = [cloud_filter; wall_points];

    end
end

 end
```

B 4. eigenvectors_full_cells

```
function [cells] = eigenvectors_full_cells(cells, ratio_threshold,...
    delta, minimum_number_of_points_in_cell)

% author:   Magdalena Linkiewicz
%
% Masterarbeit 2012, Beuth Hochschule für Technikm
% Studiengang Geoinformationsysteme
%
% Extraktion von senkrechten Fassadenebenen aus 3D-Punktwolken
% von Schrägluftbildern
%
% version:  1.0
%
% name:     eigenvectors_full_cells
%
% purpose:  Performs linear regression analysis for neighbourhood of cells.
%
% A local point cloud based on points from neighbouring cells of a cell is
% calculated. Calculation of eigenvectors of 2D covariance matrix for the
% local point cloud is performed.
% The eigenvector for the larger eigenvalue defines the local direction
% of the wall. The local direction is saved in the field "incline" of the
% cell as angle in degrees. A cell is marked as belonging to a wall, if the
% ratio of the lower and greater eigenvalues is smaller then a threshold.
%
% input
% cells:            array of cells containing points
```

```
% ratio_threshold:  upper threshold for local cloud to be accepted as
%                   linear
% delta:            width of neighboorhood taken as window for local
%                   regression
%
% output
% cells:            the cells used as input with angle of local regression
%                   line
%
%
% example:  [cells] = eigenvectors_full_cells(cells, 0.3, 1)

[length_cells, width_cells] = size(cells);

ratio=[];

% loop over all cells of the 'cells'
for k = 1:length_cells
    for l = 1:width_cells

        % skip cells that have allready wall properties
        if  cells(k, l).is_wall == true
            continue
        end

        % skip cells with too little points
        if (size(cells(k,l).points, 1) < minimum_number_of_points_in_cell)
            continue;
        end

        % calculate indices defining a neighbourhood of the current cell
        cloud_cells = get_neighbourhood_points( cells, k, l, delta );

        [count_cloud] = size(cloud_cells,1);

        % mean of point cloud
        cells(k, l).mean = ...
            [mean(cloud_cells(:, 1)), mean(cloud_cells(:, 2))];

        % covariance matrix of projected points in cell
        cov_matrix = cov(cloud_cells(:,1:2));

        % ordered eigenvectors as columns and eigenvectors on the diagonal
        [v, n_square] = eig(cov_matrix);

        % root of eigenvalue is standard deviation
        % standard deviation is proportional to width of point cloud
        % in opposite to variance which is not
        n = sqrt(n_square);

        % scale eigenvectors with eigenvalue
        % the first eigenvector has the lowest eigenvalue
        v_short_scal = n(1,1) * v(:,1);
        v_long_scal = n(2,2) * v(:,2);

        % ratio;
        if n(2,2) > 0.0
            ratio = n(1,1) / n(2,2);
        end
```

```
        % assign data to cell
        cells(k, l).ratio = ratio;
        cells(k, l).v_long_scal = v_long_scal;
        cells(k, l).v_short_scal = v_short_scal;

        % calculate angle of eigenvector with largest eigenvalue
        % with respect to x-axis
        % in MatLab: atan(1/0) = 1.570796326794897 => 90.000000000000014
        % degree
        incline = ...
            atan(cells(k, l).v_long_scal(2)/cells(k, l).v_long_scal(1));
        incline = 180.0*incline/pi;
        cells(k, l).incline = incline;

        % if ratio is to large, there is no line detected
        if ratio < ratio_threshold
            cells(k, l).is_wall = true;
        end

    end % width
end % length
end
```

B 5. init_walls

```
function [cells,walls] = init_walls(cells, max_incline_difference)

% author:   Magdalena Linkiewicz
%
% Masterarbeit 2012, Beuth Hochschule für Technik,
% Studiengang Geoinformationsysteme
%
% Extraktion von senkrechten Fassadenebenen aus 3D-Punktwolken von
% Schrägluftbildern
%
% version:  1.0
%
% name:     init_walls
%
% purpose:  Initializes groups of neighbouring cells with similar angle.
%
% A group of cells is organized in a structure "wall".
% Each wall contains the indices of the cells belonging to it.
% The walls are organized in the array "walls".
% Each cell contained in a wall, has an index "wall_id" as
% index of the wall in the array "walls".
% Each wall has a field "incline" defining the angle between the wall and
% the x-axis.
% This field is calculated as mean of the first three cells added to the
% wall.
% A cell is added to the wall, if the deviation of the cells angle from
% the walls angle is smaller than a threshold. Otherwise a new wall is
% initialized with the cell.
%
% input
% cells:   array of cells containing points with information angle of local
%          regression line
% max_incline_difference:
```

```
%          maximum angle deviation in degree for cells to be
%          joined into one group of cells
%
% output
% cells:    the cells used as input with additional data to which wall they
%           belong
% walls:    an array of the structure wall, containing cells and containing
%           the field "incline", as angle of local regression line of the
%           wall
%
%
% example:  [cells,walls] = init_walls( cells, 10.0 )


% control admissible difference between inclines
cell_num_used_for_average = 3;

% global wall index
new_wall_index = 0;

[length_cells, width_cells] = size(cells);

% maximum number of walls
max_walls_id = length_cells*width_cells;

walls(max_walls_id) = struct('id', [], 'cells_ids', [], 'incline', [], ...
    'incline_tmp_buffer', [], 'segment', [], 'next_wall', [], 'is_wall',...
    [],'min_height', [],'max_height', [],'std_dev_normal', [], 'mean', []);

% init walls (overhead will be removed at the end of this function)
for k = 1:max_walls_id

    % global wall id
    walls(k).id = [];

    % ids of cells belonging to wall
    walls(k).cells_ids = [];

    % wall direction
    walls(k).incline = [];

    % buffer used to calculate wall direction
    walls(k).incline_tmp_buffer = [];

    % segment, given by the 2D coordinates of the two endpoints of the
    % wall, each endpoint has a local index which is 1 or 2
    walls(k).segment = [];

    % id of building the wall belongs to
    walls(k).building_id = 0;

    % Structure for neighbourhood relation
    % for first end point of the wall.
    % This structure contains data of the neighbouring wall.
    % It is defined for the first (1) and second (2) end point
    % of the wall.
    %
    % neighbourhood relation for first end point
    %
    % local index of end point (1 or 2)
```

```
    walls(k).next_wall(1).point_index = [];

    % global id
    walls(k).next_wall(1).wall_id = [];

    % if false do not use neighbourhood relation
    walls(k).next_wall(1).use = false;

    % true if corner between walls has been calculated
    walls(k).next_wall(1).is_corner = false;

    % neighbourhood relation for second end point
    walls(k).next_wall(2).point_index = [];
    walls(k).next_wall(2).wall_id = [];
    walls(k).next_wall(2).use = false;
    walls(k).next_wall(2).is_corner = false;

    % index to indicate if wall is to be used
    walls(k).is_wall = false;

    % height of lower and upper end of the wall
    walls(k).min_height =[];
    walls(k).max_height = [];

    % statistical data only for documentation
    walls(k).std_dev_normal = -1.0;
    walls(k).mean = [];
end

% loop over all cells of the cells
for k = 1:length_cells
    for l = 1:width_cells

        % skip cells which are no walls
        if cells(k, l).is_wall == false
            continue
        end

        % determine indices of neighbouring cells
        min_k = max(1,k-1);
        max_k = min(length_cells,k+1);
        min_l = max(1,l-1);
        max_l = min(width_cells,l+1);

        % loop over neighbour cells
        % and search for neighbour with valid wall_id and same incline
        for k_neighbour= min_k:max_k
            for l_neighbour = min_l:max_l

                % skip neighbouring cell coinceeding with original cell
                if (k_neighbour == k) && (l_neighbour == l)
                    continue;
                end

                % skip neighbours cells which are no walls
                if cells(k_neighbour, l_neighbour).is_wall == false
                    continue
                end
```

```
            % error case
            if cells(k_neighbour, l_neighbour).wall_id == 0
                continue;
            end

            % error case: no id set
            if cells(k, l).wall_id ~= 0
                continue;
            end

            % get wall id from neighbour
            wall_id  = cells(k_neighbour, l_neighbour).wall_id;

            % reverse sign of cell incline if necessary
            % to point in direction of wall incline
            wall_incline = walls(wall_id).incline;
            cell_incline = cells(k, l).incline;
            if (abs(wall_incline - cells(k, l).incline) > 90.0)
                cell_incline = cell_incline + sign(wall_incline)*180.0;
            end

            % main idea: assure that directions are nearly the same
            %
            % continue if cell incline deviates too much
            % from wall incline
            if (abs(wall_incline - cell_incline) > ...
                    max_incline_difference)
                continue;
            end

            % assign wall to cell
            cells(k, l).wall_id = wall_id;

            % assign cell to wall
            walls(wall_id).cells_ids = [walls(wall_id).cells_ids; k,l];

            % calculate incline as average from first 3 cells
            % in order to improve results
            num_cells = size(walls(wall_id).cells_ids, 1);
            if num_cells <= cell_num_used_for_average

                % append incline to buffer
                %
                % note that cell incline has beed oriented according
                % to wall incline
                walls(wall_id).incline_tmp_buffer(end+1) =...
                    cell_incline;

                % wall incline is a average of incline of buffer
                walls(wall_id).incline = ...
                        mean(walls(wall_id).incline_tmp_buffer(:));
            end

        end % l
    end % k, end loop over neighbour cells

    % skip cell with wall id
    if cells(k, l).wall_id ~= 0
        continue;
    end
```

```
        % programm flow:
        %
        % Cell has no wall assigned.
        % Neighbours of the current cells have been checked.
        % No neighbour cell found with wall that has similar incline as
        % cell. Hence create new wall containing this cell.

        % generate new wall index
        new_wall_index = new_wall_index+1;
        wall_id = new_wall_index;

        % assign wall to cell
        cells(k, l).wall_id= wall_id;

        % assign cell to wall
        walls(wall_id).is_wall = true;
        walls(wall_id).id = cells(k, l).wall_id;
        walls(wall_id).cells_ids = [walls(wall_id).cells_ids; k,l];

        % init wall inclination
        walls(wall_id).incline = cells(k, l).incline;

        % buffer to calculate incline as mean
        walls(wall_id).incline_tmp_buffer(1) =  walls(wall_id).incline;

    end % end looping l
end % end looping k

% remove overhead of initialized wall data
walls(new_wall_index+1:end) = [];

end
```

B 6. merge_walls

```
function [cells,walls] = merge_walls(cells, walls, max_incline_difference)

% author:   Magdalena Linkiewicz
%
% Masterarbeit 2012, Beuth Hochschule für Technik,
% Studiengang Geoinformationsysteme,
% Extraktion von senkrechten Fassadenebenen aus 3D-Punktwolken
% von Schrägluftbildern
%
% version:  1.0
%
% name:     merge_walls
%
% purpose:  Merges neighbouring walls with similar angle.
%
% A loop is performed for a cells. If a cell (base cell)
% has a neighbouring cell and if the deviation of angles
% of the wall containing the base cell and the wall containing the
% neighbouring cell is smaller than a threshold, the walls are merged.
% The cells of the wall with the higher ID are added to the wall
% with the lower ID. The wall with the higher ID is marked as not to use.
```

```
% The  field "wall_id" of the cells added are updated to the wall with the
% lower ID.
%
% input
% cells:        array of cells containing points
% walls:        walls to be merged
% max_incline_difference:
%               maximum angle deviation in degree for cells
%               to be joined into one group of cells
%
% output
% cells:        the cells used as input with additional data to which
%               wall they belong
% walls:        the walls used as input either with added cells or
%               marked as not to use
%
%
% example:  [cells,walls] = init_walls( cells, 10.0 )

% control admissible difference between inclines
delta = 1; % width of cell neighbourhood

% loop over all cells
[length_cells, width_cells] = size(cells);
for k = 1:length_cells
    for l = 1:width_cells

        % skip cell which is not lying on a walls
        if cells(k, l).is_wall == false
            continue;
        end

        % bounding indices for neighbouring cells
        min_k = max(1,k-delta);
        max_k = min(length_cells,k+delta);
        min_l = max(1,l-delta);
        max_l = min(width_cells,l+delta);

        % loop over neighbour cells
        for k_neighbour = min_k:max_k
            for l_neighbour = min_l:max_l

                % exclude neighbouring cell which coinceeds
                % with original cell
                if (k_neighbour == k) && (l_neighbour == l)
                    continue;
                end

                % skip cell which are no walls
                if cells(k_neighbour, l_neighbour).is_wall == false
                    continue;
                end

                wall_id_base = cells(k, l).wall_id;
                wall_id_neighbour = cells(k_neighbour,l_neighbour).wall_id;

                % error case: no id set
                if wall_id_neighbour == 0
                    continue;
                end
```

```
% error case: no id set
if numel(wall_id_base) == 0
    continue;
end

% continue if neighbouring cell and base cell
% belong to same wall
if wall_id_neighbour == wall_id_base
    continue;
end

% adjust sign if incline of neighbour wall
% to point in direction of incline of base wall
incline_base =  walls(wall_id_base).incline;
incline_neighbour = walls(wall_id_neighbour).incline;

if (abs(incline_base - incline_neighbour) > 90.0)
    incline_neighbour = ...
        incline_neighbour + sign(incline_base) * 180.0;
end

% main idea: assure that directions are nearly the same
%
% continue if incline of neighbour wall deviates too much
% from incline of base wall
if (abs(incline_neighbour - incline_base) > ...
                                max_incline_difference)
    continue;
end

% definition of master und slave cell, the lower index
% survives
k_master = 0; l_master = 0; k_slave = 0; l_slave = 0;

% case 1
if wall_id_neighbour > wall_id_base
    k_master = k;
    l_master = l;
    k_slave = k_neighbour;
    l_slave = l_neighbour;
end

%case 2
if wall_id_neighbour < wall_id_base
    k_slave = k;
    l_slave = l;
    k_master = k_neighbour;
    l_master = l_neighbour;
end

% determine slave and master wall id
wall_id_slave = cells(k_slave, l_slave).wall_id;
wall_id_master = cells(k_master, l_master).wall_id;

% error case: continue if wall with no data
if numel(walls(wall_id_neighbour)) == 0
    continue;
end

% count of cells in slave wall
```

```
                [count_cells, m] = size(walls(wall_id_slave).cells_ids);

                % loop over cells in slave wall
                for cell_index = 1:count_cells

                    k_orig = walls(wall_id_slave).cells_ids(cell_index,1);
                    l_orig = walls(wall_id_slave).cells_ids(cell_index,2);

                    % update wall index in slave cells
                    cells(k_orig, l_orig).wall_id = wall_id_master;

                    % append slave cell to master wall
                    walls(wall_id_master).cells_ids = ...
                        [walls(wall_id_master).cells_ids; k_orig,l_orig];

                end


                % mark slave wall to not be used
                % for the remainder of the programm
                walls(wall_id_slave).is_wall = false;

                % just for sure: reset properties of slave wall
                walls(wall_id_slave).id = 0;
                walls(wall_id_slave).cells_ids = [];


            end % end of loop over neighbour cells
        end

    end % end of loop over all cells
end

end
```

B 7. make_segments

```
function [walls]  = make_segments( cells, walls, min_count_cells, ...
    min_segments_length, only_statistics)

% author:   Magdalena Linkiewicz
%
% Masterarbeit 2012, Beuth Hochschule für Technik,
% Studiengang Geoinformationsysteme
%
% Extraktion von senkrechten Fassadenebenen aus 3D-Punktwolken
% von Schrägluftbildern
%
% version:  1.0
%
% name:     make_segments
%
% purpose:  Perform linear regression on cells belonging to wall.
%           The result is a best fitting line to the wall.
%           The end points of this line are saved as field
%           "segment" of wall.
%
```

```
% A "wall" not meeting restriction on minimum number of cells
% or minimum segment length is marked as not to use.
% Standard deviation normal to wall is calculated and
% saved to field "std_dev_normal" of wall.
% The end points of the line are calculated by projecting
% point of cells of the wall on the line.
% The projected points with the minimum and maximum signed distance
% to the mean of the wall define the segment.
%
%
% input
% cells:            array of cells containing points
% walls:            walls for which segments are calculated
% min_count_cells:  minimum number of cells for segment calculation
% only_statistics:  false/true, if true only standard deviation is
%                   calculated
%
% output
% walls:            the walls used as input with
%                   value in field "std_dev_normal" and
%                   value in field "segment"
%
%
% example:  [walls]  = make_segments( cells, walls, 3, 3.0, false)
%


[length_walls] = size(walls,2);


for i=1:length_walls



    % skip walls not to use
    if (walls(i).is_wall == false)
        continue;
    end

    % error case: skip empty wall
    if numel(walls(i).cells_ids) == 0

        % mark wall for removal
        walls(i).is_wall = false;
        walls(i).id = 0;
        continue;
    end

    % gather points from cells belonging to wall
    wall_points =[];

    cells_indicies = walls(i).cells_ids;
    count_cells = size(cells_indicies, 1);

    % skip wall with less than 2 cells, because they may
    % represent non facade objects like trees or other noise
    if count_cells < min_count_cells

        % mark wall to not be used
```

```
    walls(i).is_wall = false;
    walls(i).id = 0;
    continue;
end

% gather points in all cells belonging to wall
for j=1:count_cells

    points = cells(cells_indicies(j,1),cells_indicies(j,2)).points;

    wall_points = [wall_points; points];
end

% unlikely case
% skip walls with less then 2 points,
% because linear regression can not be calculated
if size(wall_points,1) < 2

    % mark wall to not be used
    walls(i).is_wall = false;
    walls(i).id = 0;

    continue;
end

[count_wall_points]=size(wall_points,1);

% mean of projected points
mean_wall = mean(wall_points, 1);
mean_wall = mean_wall(1:2);

% matrix of covariance
cov_wall_points = cov(wall_points(:,1:2));

% ordered eigenvectors as columns and eigenvectors on the diagonal
[v,n] = eig(cov_wall_points);

% calculation v, with length equal to 1
%
% this is the direction of the regression line
tangent_dir = [v(1,2),v(2,2)];

% minimum standard deviation is root of minimum eigenvalue
walls(i).std_dev_normal = sqrt(n(1,1));

walls(i).mean = mean_wall;

% if parameter is set, calculation is stopped here
if (only_statistics)
    continue
end;

%
% proceed with calculation of segments end points
%

% loop over all points in the wall
% calculate projection of points on regression line
lambdas = zeros(count_wall_points,1);
```

```
    for j=1:count_wall_points

        point_to_mean = wall_points(j,1:2)- mean_wall;

        lambdas(j) = dot(point_to_mean, tangent_dir);
    end

    % coefficients for end points of segment
    lambda_lower =  min(lambdas);
    lambda_upper =  max(lambdas);

    % end points of segment on regression line
    projec_lower = mean_wall + lambda_lower * tangent_dir;
    projec_upper = mean_wall + lambda_upper * tangent_dir;

    % skip segments smaller then min_segments_length
    if ( norm( projec_upper - projec_lower ) <= min_segments_length)

        % mark wall to not be used
        walls(i).is_wall = false;
        walls(i).id = 0;

        continue;
    end

    % save coordinates to wall
    walls(i).segment = [projec_lower; projec_upper];

end % end loop over walls
end
```

B 8. find_pairs

```
function [walls] = find_pairs( walls, max_pair_distance)

% author:   Magdalena Linkiewicz
%
% Masterarbeit 2012, Beuth Hochschule für Technik,
% Studiengang Geoinformationsysteme
%
% Extraktion von senkrechten Fassadenebenen aus 3D-Punktwolken
% von Schrägluftbildern
%
% version:  1.0
%
% name:     find_pairs
%
% purpose:  Create neighbourhood relationship between walls.
%
% For each segment of each wall the nearest segment of all other walls
% is searched inside a given distance. The id of the next segment and the
% local index of the end point of the next segment are saved in
% the structure "next_wall" of each wall.
%
% input
% walls:               array containing a field segment with two end points
% max_pair_distance:   maximum search distance
```

```
%
% output
% walls:               the walls used as input either with structure
%                      "next_wall" containing data of next wall
%
%
% example:  [walls] = find_pairs( walls, 10.0)

[walls_length] = size(walls,2);

% loop over all walls
for i = 1:walls_length

    % skip wrong wall
    if (walls(i).is_wall == false)
        continue;
    end

    % error case: wall with no segments
    if numel(walls(i).segment) == 0
        continue;
    end

    % for each end point of a segment
    for point_index = 1:2

        % the point for which neighbouring segment is searched
        base_point = walls(i).segment(point_index, :);

        % loop over all other walls
        min_distance = 1.0e30;
        for j = 1:walls_length

            % skip same wall
            if i == j
                continue
            end

            % skip wrong wall
            if (walls(j).is_wall == false)
                continue;
            end

            % error case: wall with no segments
            if numel(walls(j).segment) == 0
                continue;
            end

            % check distance to start and end point of walls(j)
            for k = 1:2

                point = walls(j).segment(k, :);

                distance = norm(base_point - point);

                if distance < min_distance
                    min_distance = distance;

                    if min_distance < max_pair_distance
```

```
                        % save id of next wall
                        walls(i).next_wall(point_index).wall_id...
                                    = walls(j).id;

                        % save index of neighbouring end point
                        walls(i).next_wall(point_index).point_index = k;

                        % mark structure next_wall as valid
                        walls(i).next_wall(point_index).use = true;

                    end
                end

            end % end loop over two end points of segment

        end % end of loop over other walls

    end % loop over point_index 1:2 of base wall

end

%
% Neighbourhood relation may create wrong links between walls.
% Obvious error will be detected and neighbourhood relation will be
% marked as not to use.
%

% loop over all walls
% Check if each link from one wall to another points back to the same
% wall. If this is not the case, mark relation as not to use.
number_wrong_links = 0;
for b = 1:walls_length

    if (walls(b).is_wall == false)
        continue;
    end

    for point_index = 1:2

        % if there is no link, skip checking
        if ( walls(b).next_wall(point_index).use == false)
            continue
        end

        % link to pair wall
        pair_wall_id = walls(b).next_wall(point_index).wall_id;

        pair_point_index = walls(b).next_wall(point_index).point_index;

        % link from pair wall
        pair_pair_wall_id = ...
                walls(pair_wall_id).next_wall(pair_point_index).wall_id;


        % pair wall should point to base wall
        % if not then remove link from base wall to pair wall
        if ( b ~= pair_pair_wall_id )
```

```
            % output number of wrong links
            number_wrong_links = number_wrong_links + 1

            % mark links as not to use
            walls(b).next_wall(point_index).use = false;

            % just for sure reset parameters of link
            walls(b).next_wall(point_index).point_index = [];
            walls(b).next_wall(point_index).wall_id = [];
        end

    end

end

% loop over all walls
% Check if both links from one wall point to different walls
% wall. If this is not the case, mark relation as not to use.
for b = 1:walls_length

    if (walls(b).is_wall == false)
        continue;
    end

    % if there is no link, skip checking
    if ( walls(b).next_wall(1).use == false) || ...
                        ( walls(b).next_wall(2).use == false)
        continue
    end

    % link to pair wall
    pair_wall_id_1 = walls(b).next_wall(1).wall_id;
    pair_wall_id_2 = walls(b).next_wall(2).wall_id;

    % error case: both end points point to same wall
    if (pair_wall_id_1 == pair_wall_id_2)

        number_wrong_links = number_wrong_links + 1;

        % mark links as not to use
        walls(b).next_wall(1).use = false;

        % just for sure reset parameters of link
        walls(b).next_wall(1).point_index = [];
        walls(b).next_wall(1).wall_id = [];

        % mark links as not to use
        walls(b).next_wall(2).use = false;

        % just for sure reset parameters of link
        walls(b).next_wall(2).point_index = [];
        walls(b).next_wall(2).wall_id = [];
    end

end


end %end of function
```

B 9. merge_gaps

```
function [ walls] =...
    merge_gaps( walls, max_incline_difference, max_parallel_offset )

% author:   Magdalena Linkiewicz
%
% Masterarbeit 2012, Beuth Hochschule für Technik,
% Studiengang Geoinformationsysteme
%
% Extraktion von senkrechten Fassadenebenen aus 3D-Punktwolken
% von Schrägluftbildern
%
% version:  1.0
%
% name:     merge_gaps
%
% purpose:  Merges neighbouring walls with similar angle with normal
%           distance below a given threshold. The segment for the merged
%           wall is calculated.
%
% A loop over all walls is performed. Starting from a so called base wall
% all neighbouring walls are visited using the neighbourhood relation
% "next_wall". Two conditions have to be fulfilled
% in order to merge a wall with its neighbour.
% - the deviation of angles of the walls segments is lower than
%   max_incline_difference
% - the distance between the walls segments meassured in normal direction
%   to the walls is smaller than max_parallel_offset
% If wall B is merged with wall A, than cells of B are added to wall A.
% The segment of wall A is redefined and will end with an end point
% of wall B. Wall B is marked as not to use.  Neighbourhood relations are
% updated.
%
% input
% walls:                   walls to be merged
% max_incline_difference: maximum angle deviation in degree for walls to
%                          be merged
% max_parallel_offset:    maximim normal offset between wall to be merged
%
% output
% walls:                 the walls used as input possibly merged
%
%
% example:  [ walls] = merge_gaps( walls, 10.0 , 1.0 )

% difference of angles should be less than max_incline_difference
% that means that dot product of the direction should be larger than
% the following threshold:
corner_min_dot = cos(max_incline_difference*pi/180.0);

% loop over all walls
[length_walls] = size(walls,2);
for base_wall_id = 1:length_walls

    base_wall = walls(base_wall_id);

    % skip wrong walls
    if (base_wall.is_wall == false)
        continue;
```

```
end

% error case: wall with no segment
if numel(base_wall.segment) == 0
    continue;
end


% direction of base wall r := p_2 - p_1
direct_vector = base_wall.segment(2, :) - base_wall.segment(1, :);
norm_direct_vector = direct_vector/norm(direct_vector);


% for each end point of base wall
for active_point_index = 1:2
    active_point = base_wall.next_wall(active_point_index);


    % chain of neighbouring walls starting from base wall in one
    % direction

    % an end point is initialized as "active_point" and is updated for
    % each visited neighbouring wall

    % stop condition for loop
    while ( active_point.use == true )

        % candidate for connected wall
        pair_id = active_point.wall_id;

        pair_wall = walls(pair_id);

        % error case: pair wall is wrong wall
        if  (pair_wall.is_wall == false)
            break;
        end

        % error case: pair wall and base wall have same id
        if pair_wall.id == base_wall.id
            break;
        end


        % calculate normal distance between segments: "parallel_offset"
        p = base_wall.segment(active_point_index, :);
        q1 = pair_wall.segment(1,:);
        q2 = pair_wall.segment(2,:);

        u = p - q1;
        v = q1 - q2;
        v = v / norm(v);

        lambda = dot(u, v);

        p_prim = q1 + lambda*v;
        distance = p - p_prim;
        parallel_offset = norm(distance);
```

```
% adjust sign if incline of pair wall
% to point in direction of incline of base wall
if (abs(base_wall.incline - pair_wall.incline) > 90.0)
    pair_wall.incline = ...
           pair_wall.incline + sign(base_wall.incline) * 180.0;
end

corner_dot = dot(norm_direct_vector, v);

% main idea:
%
% merge wall with similar direction and parallel distance lower
% than threshold
if ( abs(corner_dot) > corner_min_dot ) ...
        && (parallel_offset < max_parallel_offset)

        %  convert 1->2 and 2->1
    opposite_index = mod(active_point.point_index,2) + 1;



    %
    %  step 1: enlarge segment
    %
    %  enlarge segment of base wall to end of segment of pair
    %  wall

    base_wall.segment(active_point_index, :) = ...
                        pair_wall.segment(opposite_index, :);

    %
    %  step 2: update cells belonging to walls
    %
    % base wall takes cells from pair wall
    base_wall.cells_ids = ...
                    [base_wall.cells_ids; pair_wall.cells_ids];
    walls(pair_wall.id).cells_ids = [];

    % eliminate pair wall from walls
    walls(pair_wall.id).is_wall = false;

    % save changes to base wall
    walls(base_wall_id) = base_wall;

    %  step 3: update topology
    %
    % This is a little bit difficult to understand.
    % There are three walls involved:
    % 1.) the base_wall,
    % 2.) its pair_wall as neighbouring wall,
    % 3.) the neighbour of the pair wall called "new_next_wall"
    %
    % After the update the walls "new_next_wall" and
    % "base_wall" will be neighbours.

    % wall id of new next wall
    new_next_wall_id =...
        pair_wall.next_wall(opposite_index).wall_id;
    % point index of new next wall
    new_point_index = ...
        pair_wall.next_wall(opposite_index).point_index;
```

```
                % wrong values of new next wall
                %
                % in case that a wall has no neighbour defined
                % topology will not be updated
                if ( numel(new_point_index) == 0 || ...
                               numel(new_next_wall_id) == 0 )

                    % mark neighbourhood relation as not to use
                    walls(base_wall_id).next_wall(...
                               active_point_index).use = false;
                    walls(base_wall_id).next_wall(...
                               active_point_index).point_index =[];
                    walls(base_wall_id).next_wall(...
                               active_point_index).wall_id =[];

                    % stop searching for pair wall
                    break;
                end;

                % update link from base wall to new next wall
                % update id of next wall to id of pair wall
               walls(base_wall_id).next_wall(active_point_index).wall_id...
                                    = new_next_wall_id;

                % update index of end point of new next wall according
                % to index of end point of pair wall
                walls(base_wall_id).next_wall(...
                          active_point_index).point_index =new_point_index;

                % update link from new next wall to base wall
                walls(new_next_wall_id).next_wall(...
                          new_point_index).point_index =active_point_index;
                walls(new_next_wall_id).next_wall(...
                          new_point_index).wall_id = base_wall_id;
             else

                % stop searching for pair wall
                break;
            end % merge wall

            % if closest point of pair wall, is end point (number == 2) of
            % pair wall, then proceed with start point of pair wall
            if active_point.point_index == 2
                active_point = pair_wall.next_wall(1);
            end

            if active_point.point_index == 1
                active_point = pair_wall.next_wall(2);
            end

        end % end chain of neighbouring walls

    end % end for active_point_index = 1:2

end % end walls

end
```

B 10. find_corner

```
function [ walls  ] = find_corner( walls, min_incline_difference )

% author:   Magdalena Linkiewicz
%
% Masterarbeit 2012, Studiengang Geoinformationsysteme,
%
% Extraktion von senkrechten Fassadenebenen aus 3D-Punktwolken
% von Schrägluftbildern
%
% version:  1.0
%
% name:     find_corner
%
% purpose:  Calculates intersection of neighbouring wall segments,
%           if their angle deviation is greater than a certain threshold
%
% Intersection of wall segments is calculated resulting in a corner.
% The end points of the two segments are updated by the coordinates of the
% corner.
%
% input
% walls:                    walls to be processed
% min_incline_difference:   minimum deviation of angle between walls in
%                           degree
%
% output
% walls:                    the walls used as input either with more cells
%                           or marked as not to use
%
%
%
% example:  [ walls  ] = find_corner( walls, 10.0 )

% if the angle deviation is greater than a "min_incline_difference"
% than the dot product of the direction should be lower than
% "corner_max_dot"
corner_max_dot = cos(min_incline_difference*pi/180.0);

% loop over all walls
[walls_length] = size(walls,2);
for w = 1:walls_length

    % skip wrong wall
    if walls(w).is_wall == false
        continue;
    end

    base_wall = walls(w);

    % direction of base wall r := p_2 - p_1
    direct_vector = base_wall.segment(2, :) - base_wall.segment(1, :);
    base_wall_dir = direct_vector/norm(direct_vector);

    % calculate corner intersection for both ends of segment of base wall
    for active_point_index = 1:2;

        % skip when no topology
```

```
if ( base_wall.next_wall(active_point_index).use == false )
    continue;
end

pair_wall_id = base_wall.next_wall(active_point_index).wall_id;

% error case
if walls(pair_wall_id).is_wall == false
    continue;
end

% skip if corner intersection has already been done
if ( base_wall.next_wall(active_point_index).is_corner == true )
    continue;
end

next_wall_point_index = ...
        base_wall.next_wall(active_point_index).point_index;
pair_wall = walls(pair_wall_id);
pair_wall_dir = pair_wall.segment(1,:) - pair_wall.segment(2,:);
pair_wall_dir = pair_wall_dir/norm(pair_wall_dir);

corner_dot = dot(base_wall_dir, pair_wall_dir);

% swip corner intersection calculation if angle between
% base wall and pair wall is to small
if abs(corner_dot) > corner_max_dot
    continue;
end

% calculate intersection of  line_q with normal n and point q_0
% with line_p given by point p_0 and direction v
% resulting in corner
%
% here:
% - n is orthogonal to direction of pair wall
% - r is in direction of base wall
% - q_0 is point on pair wall
% - p_0 is point on base wall
%
% n
pair_wall_normal = [-pair_wall_dir(2), pair_wall_dir(1)];

% q_0 - p_0
sub_points =  pair_wall.segment(1,:) - base_wall.segment(1, :);

% lambda = dot(n , q_0 - p_0 ) / dot ( n, v )
lambda = ...
dot(pair_wall_normal, sub_points)...
        /dot(pair_wall_normal, base_wall_dir);

% p_1 + lambda * r
corner = base_wall.segment(1, :) + (lambda*base_wall_dir);

% assign corner as new end point for segment of base wall and pair
% wall
pair_wall.segment(next_wall_point_index, :) = corner;
base_wall.segment(active_point_index,:) = corner;

% mark that corner calculation was done
```

```
        base_wall.next_wall(active_point_index).is_corner = true;
        pair_wall.next_wall(next_wall_point_index).is_corner = true;

        % write results back to walls
        walls(w) = base_wall;
        walls(pair_wall_id) = pair_wall;

    end % end for both end points of segments

end % loop over walls

end
```

B 11. calculate_height

```
function [ walls ] = ...
    calculate_height( unfiltered_cloud, base_cloud, walls, interval_width)

% author:   Magdalena Linkiewicz
%
% Masterarbeit 2012, Studiengang Geoinformationsysteme
%
% Extraktion von senkrechten Fassadenebenen aus 3D-Punktwolken
% von Schrägluftbildern
%
% version:  1.0
%
% name:     calculate_height
%
% purpose:  Calculates height of lower and upper edge of wall
%           based on unfiltered point cloud.
%
% For each wall points of an unfiltered cloud in a neighourhood of wall
% segment are collected in temporary cloud. In order to speed up collection
% the indices of cells belonging the wall are used.
% To create these cells the cloud of these point will be subdivided in
% in the same way, the filtered cloud was subdivided. For this the point
% cloud "base_cloud" is used.
% For points in the temporary cloud with in a certain distance from wall
% segment the z coordinate is used for calculation of wall height.
%
%
% input
% unfiltered_cloud:     cloud with all points including points from
%                       ground, roof and other non facade objects
% base_cloud:           point cloud used for subdividing the filtered point
%                       cloud in cells
% walls:                walls for which height has to be calculated
%                       minimum deviation of angle between walls
% interval_width:       cell width used to define cells for filtered point
%                       cloud
%
% output
% walls:                the walls with additional data in fields
%                       min_height and max_height
%
%
%
```

```
% example:  [ walls ] = ...
%           calculate_height( unfiltered_cloud, base_cloud, walls, 1.6)

% value to remove outliers for points on ground and roof
ground_outlier = 0.1; % quantile 0.1%
roof_outlier = 5; % quantile 5%

% maximum distance of projected point from segment for height calculation
delta = 0.5;

% subdivide unfiltered cloud in cells with the same grid used to subdivide
% filtered cloud
[cells_unfiltered] = ...
        make_cloud_in_cells(unfiltered_cloud, base_cloud, interval_width);

% loop over all walls
[length_walls] = size(walls,2);
for i=1:length_walls

    % temporary array to hold points near wall segment
    wall_points =[];

    % skip wrong walls
    if (walls(i).is_wall == false)
        continue;
    end

    cells_indicies = walls(i).cells_ids;

    count_cells = size(cells_indicies, 1);

    % gather points in all cells (cloud_orig) belonging to wall
    for j= 1:count_cells

        points = ...
          cells_unfiltered(cells_indicies(j,1),cells_indicies(j,2)).points;

        % skip cells with no points
        if numel(points)== 0
            continue;
        end

        tangent_dir = walls(i).segment(1,:) - walls(i).segment(2,:);
        tangent_dir = tangent_dir / norm(tangent_dir);

        % loop over points in cell, collect near points
        num_points = size(points,1);
        for p = 1:num_points

            % calculate distance of projected point to wall segment
            point = points(p,1:2);

            point_to_mean = point - walls(i).mean;

            lambda = dot(point_to_mean, tangent_dir);

            projection = walls(i).mean + lambda*tangent_dir;

            distance = norm(point - projection);
```

```
            % if point is away from segment, then skip point
            if distance > delta
                continue
            end

            % append point to wall
            wall_points = [wall_points; points(p,:)];
        end

    end

    if ( numel(wall_points) == 0 )
        continue;
    end

    % sort points in cell (height)
    sort_points = sortrows(wall_points, 3);
    n=size(sort_points,1);

    % calculate quantile of ground outlier
    k_ground = ceil(n*ground_outlier/100);
    min_height = sort_points(k_ground,3);

    % calculate quantile of roof outlier
    point_k_roof = ceil(n*(100-roof_outlier)/100);
    max_height = sort_points(point_k_roof,3);

%     % make histogramm of wall, class_width = 1.0 m
%     class_width = 1.0;
%     min_z = min(wall_points(:,3));
%     max_z = max(wall_points(:,3));
%     count_interval = (max_z - min_z)/class_width;
%
%     [class_points, class_value] = hist(wall_points(:,3), count_interval);
%     hist_matrix=[class_points; class_value]';

    % z-value of lower edge of wall
    walls(i).min_height = min_height;

    % z-value of upper edge of wall
    walls(i).max_height = max_height;

end

end
```

B 12. make_buildings

```
function [ walls, buildings ] = make_buildings( walls )

% author:   Magdalena Linkiewicz
%
% Masterarbeit 2012,
% Beuth Hochschule für Technik, Studiengang Geoinformationsysteme,
% Extraktion von senkrechten Fassadenebenen aus 3D-Punktwolken
```

```
% von Schrägluftbildern
%
% version:  1.0
%
% name:     make_buildings
%
% purpose:  join walls to buildings
%
% Chain of connected walls are written as polygon to structure "buildings"
% Each chain of connected walls is a building, consisting of a polygon
% containing the end points of the wall segments.
%
% For each wall two loops will be performed.
% In the first loop the end of the chain of connected wall is searched for.
% For a closed chain of walls there is no end and the loop finishes
% when the start wall is reached.
% In the second loop starting from the end of the chain all connected
% walls are traversed. For each wall the end point of a segment is
% added to buildings polygon. If corner calculation was not done, than
% both end points are added to the polygon.
%
% A minimum  and maximum height is assigned to building
% based on median values of walls.
%
%
% input
% walls:     walls to be processed
%
% output
% walls:     walls used as input with additional field "building_id"
% buildings: an array of structure building
%
%
% example:  [ walls, buildings ] = make_buildings( walls )

global_building_id = 0;
buildings(1) = struct('points', []);

% loop over all walls
[length_walls] = size(walls,2);
for base_id = 1:length_walls

    if (  walls(base_id).is_wall == false )
        continue;
    end

    % skip walls with building id
    if (walls(base_id).building_id ~= 0)
        continue;
    end

    pair_id =  base_id;
    opposite_index = 1;

    % these values are used only for initialisation
    % and will be overwritten later
    min_height = 5;
    max_height = 40;

    % assign the same building id to all connected pair walls
    active_point_index = 1;
```

```
active_point = walls(base_id).next_wall(active_point_index);

% first loop connected walls starting from base wall in one direction

% for open chain of walls the stop condition will be that the
% end of the chain is reached: there will be no neighbour, that means
% the value "use" of structure "next_wall" is set to "false"
test_endless = 0;
while ( active_point.use == true )

    % error case: to prevent an endless loop due to bad topology
    % use a counter as backdoor
    test_endless = test_endless + 1
    if test_endless > 1000
        stop = 1;
    end

    % id of pair  wall
    pair_id = active_point.wall_id;

    % error case: pair wall is wrong wall
    if ( walls(pair_id).is_wall == false )
        break;
    end

    % error case:pair wall has no building id assigned
    if ( walls(pair_id).building_id ~= 0 )
        break;
    end

    % stop condition: for closed buildings loop will
    % lead to starting wall
    if (  pair_id == base_id )
        break;
    end


    %  convert 1->2 and 2->1
    opposite_index = mod(active_point.point_index,2) + 1;

    %  next wall
    active_point = walls(pair_id).next_wall(opposite_index);

end

% start with pair wall from end of chain of walls
start_id = pair_id;

% start new building
% increment and assign global building id to wall
global_building_id = global_building_id + 1;

walls(start_id).building_id = global_building_id;

buildings(global_building_id).points = ...
    [walls(start_id).segment(opposite_index,:),min_height, max_height];
```

```
%  convert 1->2 and 2->1
opposite_index = mod(opposite_index,2) + 1;

buildings(global_building_id).points(end+1,:) = ...
    [walls(start_id).segment(opposite_index,:),min_height, max_height];


active_point = walls(start_id).next_wall(opposite_index);

test_endless = 0;

% second loop over connected walls starting end of wall chain
while ( active_point.use == true )

    % backdoor to prevent endless loop
    test_endless = test_endless + 1
    if test_endless > 1000
        stop = 1;
    end

    pair_id = active_point.wall_id;

    % error case
    if ( walls(pair_id).is_wall == false )
        break;
    end

    % stop, if pair wall has building id assigned
    if ( walls(pair_id).building_id ~= 0 )
        break;
    end

    % stop condition for closed chain of walls
    if ( pair_id == start_id )
        break;
    end

    %  convert 1->2 and 2->1
    opposite_index = mod(active_point.point_index,2) + 1;


    % if not corner, add additional point
    % as point opposite to active point
    if (active_point.is_corner == false)
        buildings(global_building_id).points(end+1,:) =  ...
            [walls(pair_id).segment(active_point.point_index,:),  ...
            min_height, max_height];
    end

    % add point to buildings polygon
    buildings(global_building_id).points(end+1,:) =  ...
     [walls(pair_id).segment(opposite_index,:),min_height, max_height];


    walls(pair_id).building_id = global_building_id;


    %  next wall
    active_point = walls(pair_id).next_wall(opposite_index);
```

```
    end

end % loop over all walls

% assign height to building
for b = 1:size(buildings,2)

    wall_ids = find(horzcat(walls(:).building_id) == b);

    b_min_height = median([walls(wall_ids).min_height]);
    b_max_height = median([walls(wall_ids).max_height]);

    buildings(b).points(:,3) = b_min_height;
    buildings(b).points(:,4) = b_max_height;

end;

end
```

C 1. Berechnete Koordinaten der Fassadenebenen

UTM, WGS84, Zone 33 N

# X	Y	Z- untere Kante	Z- obere Kante
# B1			
400134.938629	5809714.920479	74.627826	103.495233
400148.116469	5809699.991444	74.627826	103.495233
400186.668668	5809732.883642	74.627826	103.495233
400173.796657	5809748.273201	74.627826	103.495233
400134.938629	5809714.920479	74.627826	103.495233
# B2			
399863.262387	5809748.518116	74.891271	95.047315
399851.259738	5809687.210486	74.891271	95.047315
399868.506228	5809683.852029	74.891271	95.047315
399871.543043	5809698.732991	74.891271	95.047315
399870.237473	5809705.028693	74.891271	95.047315
399875.817649	5809733.453724	74.891271	95.047315
399886.623232	5809736.253824	74.891271	95.047315
399889.203152	5809735.053355	74.891271	95.047315
399895.817874	5809735.678510	74.891271	95.047315
399899.719900	5809737.660884	74.891271	95.047315
399902.038426	5809740.033405	74.891271	95.047315
399912.469273	5809742.864493	74.891271	95.047315
399922.129173	5809731.778881	74.891271	95.047315
399920.647189	5809729.345612	74.891271	95.047315
399932.107103	5809716.161058	74.891271	95.047315
399945.944948	5809727.840678	74.891271	95.047315
399916.273500	5809761.892608	74.891271	95.047315
399863.262387	5809748.518116	74.891271	95.047315
# B3			
400016.578593	5809798.554684	74.232980	94.268036
400115.239487	5809858.683392	74.232980	94.268036
400149.569065	5809818.531448	74.232980	94.268036
400139.655693	5809810.276196	74.232980	94.268036
400116.276961	5809831.261839	74.232980	94.268036
400104.887348	5809820.720547	74.232980	94.268036
400124.926711	5809797.561510	74.232980	94.268036
400115.150460	5809789.020154	74.232980	94.268036
400091.179896	5809810.426530	74.232980	94.268036
400080.427640	5809799.236345	74.232980	94.268036

400100.016193	5809776.690601	74.232980	94.268036
400090.619353	5809767.770746	74.232980	94.268036
400066.156061	5809789.709649	74.232980	94.268036
400055.919592	5809778.045143	74.232980	94.268036
400075.414068	5809755.227338	74.232980	94.268036
400065.206374	5809746.303744	74.232980	94.268036
400016.578593	5809798.554684	74.232980	94.268036

B4a

400158.212891	5809770.665847	75.883800	86.997550
400140.652597	5809755.836931	75.883800	86.997550
400145.449650	5809750.375144	75.883800	86.997550
400127.240697	5809734.961744	75.883800	86.997550
400131.772483	5809728.674953	75.883800	86.997550
400110.599777	5809711.030577	75.883800	86.997550
400095.394675	5809728.572004	75.883800	86.997550
400121.748429	5809751.111659	75.883800	86.997550
400121.341602	5809760.295773	75.883800	86.997550
400148.715020	5809783.550019	75.883800	86.997550
400158.554770	5809772.151124	75.883800	86.997550
400166.211768	5809778.748465	75.883800	86.997550

B4b

400186.063565	5809758.450641	75.107600	96.250350
400153.022428	5809796.931708	75.107600	96.250350
400165.272569	5809807.593982	75.107600	96.250350
400198.965242	5809768.115326	75.107600	96.250350
400196.325390	5809764.181519	75.107600	96.250350

B3 – optiemiert

400064.861489455220000	5809747.568451961500000
400041.861974997970000	5809768.492461268800000
400042.715773300380000	5809767.331402614700000
400016.274321901840000	5809798.594346266200000
400032.805712985980000	5809806.910164530400000
400034.054738214650000	5809807.097067131700000
400039.167551172200000	5809810.455494697200000
400038.900845639230000	5809810.227903091300000
400064.383773060860000	5809824.392704316400000
400064.982471077640000	5809824.487929050800000
400114.805863041720000	5809859.175553429900000
400149.790469486560000	5809818.326109778100000
400144.789422738950000	5809814.750795695000000

400140.340116203820000 5809809.691818292300000
400116.237228462590000 5809831.228062874600000
400104.893319652590000 5809820.715872995600000
400125.383563604090000 5809797.038126389500000
400121.795123522640000 5809795.085660875800000
400119.559571597960000 5809793.234105804000000
400120.406550512360000 5809793.926167663200000
400115.711633377710000 5809788.523717571000000
400091.250152707100000 5809810.330488015900000
400082.336755992090000 5809801.369520183700000
400075.707844360960000 5809755.006739340700000
400056.019413886940000 5809777.734955032400000
400059.899234468520000 5809782.843862192700000
400066.952485677670000 5809788.979104214300000
400090.212757341450000 5809768.156167746500000
400096.434412356290000 5809773.170661238000000
400095.164779870360000 5809772.834159286700000
400100.630314788320000 5809775.976271756900000
400081.816388336070000 5809797.652515528700000
400081.816388336070000 5809797.652515528700000

B4 – optimiert
400158.383462198540000 5809770.817085921800000
400140.893101110710000 5809756.013892569600000
400145.289760137970000 5809750.126477717400000
400128.241360311860000 5809735.897681574300000
400127.870200466130000 5809734.319564461700000
400131.405190705900000 5809728.841998685200000
400110.595080561410000 5809710.996255927700000
400095.413155428080000 5809728.580688550100000
400123.494669853540000 5809752.628725331300000
400123.536863267540000 5809752.083881441500000
400120.354203688970000 5809759.459576945800000
400148.650047472150000 5809783.521133310200000
400158.577475253900000 5809772.185555736500000
400166.185391483070000 5809778.823007063000000
400153.094577533250000 5809796.837789331600000
400185.770429185940000 5809758.803555017300000
400190.888006939440000 5809759.827922891800000
400193.839959287610000 5809758.479314575000000
400197.214467822870000 5809761.366087926600000
400196.587893887660000 5809764.699951580700000
400199.098461517370000 5809767.916796560400000
400165.058338957200000 5809807.866948647400000

400164.547884644300000	5809807.071391183900000
400153.094577533250000	5809796.837789331600000