Bachelorarbeit

Delaunay-Triangulierung und Ausdünnung

Maximilian Schmöller

betreut von
Prof. Dr. Tomas Sauer

31. März 2017

Kurzfassung

Der Fortschritt von Laserscan-Technologien hat dazu geführt, dass mit ihrer steigenden Qualität und Verbreitung die von ihnen produzierte Datenmenge immer weiter anwächst. Diese Arbeit stellt einen Ansatz vor, der über zweidimensionale Triangulationen der Daten dieses Problem zu lösen versucht. Es wird gezeigt, dass das beschriebene Vorgehen für die Ausdünnung von großen Punktwolken Potential birgt.

Inhaltsverzeichnis

Kapitel 1

Einführung

In diesem Kapitel soll zunächst die Motivation der Arbeit erläutert werden. Im Anschluss daran wird näher auf Triangulierungen eingegangen.

1.1 Motivation

Nachdem der erste Laser in den 60er-Jahren entwickelt wurde, war er als Hilfsmittel zur Abmessung von Umgebungen nicht mehr wegzudenken. Seit dem Ende des 20. Jahrhunderts gab es ein rapides Wachstum von Laser-Scan-Technologien, wodurch immer schnelleres und breitflächigeres Messen möglich wurde [Large and Heritage, 2009]. LiDAR-Systeme, die mittels Laserstrahlen Oberflächen abtasten und dadurch digitale 3D-Modelle erzeugen, sind heutzutage weit verbreitet und erfreuen sich anwachsender Beliebtheit. Die Anwendungsbereiche dieser Technologie reichen von Geologie, über Archäologie bis hin zu Robotik und selbstfahrenden Fahrzeugen. Auch in der Industrie werden solche Systeme mittlerweile vermehrt zur Prozesssteuerung oder für *Rapid-Prototyping-Verfahren* genutzt. So gut und wichtig diese Entwicklungen sind, mit wachsender Beliebtheit und wachsender Forschung, wachsen die Datenmengen, welche von den Scannern aufgenommen werden, auch aufgrund von ansteigender Auflösung. Dieser Zustand stellt hohe Anforderungen an die Rechenleistung der meisten Systeme. Außerdem erfordern die großen, hochaufgelösten Datenmengen eine größere Speicherkapazität und erschweren die Anwendbarkeit der Scan-Daten.

Scan-Daten werden häufig als Punktwolken gespeichert, deren Punkte allerdings oft nicht alle relevant für die Modellierung der dargestellten Objekte

sind. Diese Arbeit stellt einen Versuch dar, mithilfe eines auf Triangulierung basierenden Algorithmus, die von Laserscannern erzeugten Punktmengen, sinnvoll zu reduzieren. Ziel ist es, einen kleineren Datensatz zu erhalten, welcher trotz signifikanter Ausdünnung nicht an Aussagekraft verliert.

1.2 Triangulierungen

In diesem Abschnitt soll genauer auf Triangulierungen eingegangen werden, da diese eine wichtige Rolle für das in Kapitel 2 vorgestellte Verfahren spielen. Eine umfassende Arbeit zu diesem Thema stellt [Hjelle and Dæhlen, 2006] dar, das Buch dient zu großen Teilen als Leitfaden für diesen Abschnitt.

Bei Triangulierungen handelt es sich um Netze von Dreiecken. Wir befassen uns dabei spezifisch mit Triangulierungen, welche aus Mengen von Punkten konstruiert werden. Der Einfachheit halber konzentrieren wir uns hier auf Punkte im zweidimensionalen Raum, welche durch kartesische Koordinaten definiert sind. Solche Triangulierungen werden in einer Vielzahl von Anwendungsbereichen genutzt.

Triangulierungen werden zumeist als planare Graphen aufgefasst. Die Punkte der Menge, die zur Konstruktion der Triangulierung verwendet werden, stellen die Knoten V des Graphen dar. Die Menge der Kanten des Graphen sind die Menge der Kanten E der Dreiecke T, wobei jede Kante des Graphen Kante von maximal zwei Dreiecken ist. Hier sollte zudem kurz auf die Größe des Graphen eingegangen werden. Dafür muss jedoch eine weitere Unterteilung der Knoten vorgenommen werden. Es wird zusätzlich zwischen äußeren Knoten V_B und inneren Knoten V_I unterschieden. Es ergeben sich folgende Größen für die Menge der Dreiecke und Kanten im Graphen.

$$|T| = 2|V_I| + |V_B| - 2 \tag{1.1}$$

$$|E| = 3|V_I| + 2|V_B| - 3 \tag{1.2}$$

Ein Beweis dazu findet sich in [De Berg et al., 2000, S.193]. Ein Wissen um die Menge der Komponenten des Graphen ist relevant für die Analyse des Speicherplatzbedarfs von Implementierungen des Triangulierungs-Graphen. Zudem wird es benötigt, um Aussagen über die Laufzeit von Algorithmen,

welche den Graphen durchqueren, zu treffen. Aus 1.1 und 1.2 lassen sich des Weiteren die folgenden Aussagen über $|V|$ und $|E|$ ableiten.

$$|V| - 2 \leq |T| \leq 2|V| - 5 \tag{1.3}$$

$$2|V| - 3 \leq |E| \leq 3|V| - 6 \tag{1.4}$$

Diese Ungleichungen werden in [Hjelle and Dæhlen, 2006, S.9] knapp bewiesen. Diese Abschätzung ist von Vorteil, da für sie keine Unterteilung der Knoten vorgenommen werden muss und sie somit erfolgen kann, ohne die konvexe Hülle der Punktmenge zuvor zu bestimmen.

Für die Laufzeit des später beschrieben Algorithmus' zur Ausdünnung ist zudem die Anzahl der Kanten, welche an einem bestimmten Knoten $v_i \in V$ anliegen, relevant, sie notieren wir mit $deg(v_i)$. Über die Anzahl der Kanten pro Knoten lässt sich mit 1.2 und der Beobachtung, dass jede Kante genau zwei anliegende Knoten besitzt, zeigen, dass Folgendes gilt [Hjelle and Dæhlen, 2006, S.9].

$$\sum_{i=1}^{|V|} deg(v_i) = 2|E| = 6|V_I| + 4|V_B| - 6 = 6|V| - 2|V_B| - 6 \approx 6|V| \tag{1.5}$$

Jeder Knoten der Triangulierung ist somit durchschnittlich mit sechs weiteren Knoten über eine Kante verbunden. Die Abschätzung am Ende der Gleichung kann gemacht werden, da die Anzahl der Knoten des Randes einer Triangulierung in der Regel beträchtlich geringer ist als die der inneren Knoten.

Algorithmen, die eine einfach Triangulierung erzeugen, sind hinreichend bekannt und unter ihnen gibt eine Vielzahl von Abwandlungen. In der Regel unterscheidet man aber grob zwischen zwei Kategorien von Algorithmen.

Es gibt Algorithmen, die nach und nach Kanten in die Triangulierung einfügen, welche keine zuvor bereits eingefügte Kante schneiden. Ihr Abbruchkriterium ist erfüllt, wenn keine validen Kanten mehr eingefügt werden können.

Für die anderen, komplexeren Algorithmen muss zunächst die konvexe Hülle der Punktmenge bestimmt werden. Dieses geschlossene Polygon wird anschließend in Dreiecke aufgeteilt. Im nächsten Schritt werden die übrigen Punkte der Menge wieder eingeführt. Jeder Punkt wird bezüglich seiner Lage in der Triangulierung getestet: Liegt er in einem bereits vorhandenen Dreieck, so wird das Dreieck in drei neue Dreiecke aufgespalten, deren gemeinsamer Knoten der neu eingefügte Punkt ist. Liegt er auf einer Kante, so werden beide an der Kante liegenden Dreiecke gespalten.

Durch solche Algorithmen erzeugte Triangulierungs-Graphen sind nicht eindeutig. Ihre Struktur ist abhängig von der Reihenfolge, in der die Knoten eingefügt werden. Es gibt zudem keine Garantie bezüglich der Einheitlichkeit der generierten Dreiecke, ihre Innenwinkel und Flächen können teilweise stark variieren. Einfache Triangulierungen werden aus diesem Grund selten in der Praxis verwendet. Vielmehr können sie als Ausgangspunkt für Triangulierungen genutzt werden, die durch Edge-Flipping nach bestimmten Kriterien optimiert werden. Bei Edge-Flipping handelt es sich um einen Vorgang, bei dem jeweils zwei Dreiecke ausgewählt werden, die eine gemeinsame Kante besitzen. Diese Dreiecke werden mit den alternativen Dreiecken verglichen, die entstehen, wenn die gewählte Kante gelöscht und die von ihr verschiedene Kante in das so entstandene vierseitige Polygon eingefügt wird. Sollten die alternativen Dreiecke das Optimierungskriterium besser erfüllen als die alten Dreiecke, so wird die alte Kante gelöscht und die alternative Kante eingefügt. Dieser Vorgang wird solange wiederholt, bis keine der Kanten mehr optimiert werden kann.

Ziel einer solchen Optimierung ist es zum Beispiel den kleinsten Innenwinkel der Dreiecke zu maximieren. Alternativ dazu kann auch der größte Innenwinkel minimiert werden. Weitere Optimierungsbereiche sind unter anderem die minimalen bzw. maximalen Kantenlängen der Dreiecke, die minimale bzw. maximale Fläche der Dreiecke oder der maximale Grad der Triangulierung. Diese Optimierungsziele können durch Edge-Flipping in bestehenden Triangulierungen oder spezielle Triangulierungs-Verfahren erreicht werden. Nach einem dieser Kriterien optimierte Triangulierungen sind für fast alle Punktmengen eindeutig, obwohl es auch hier zu Spezialfällen kommen kann.

Die wohl meist genutzte Art der Triangulierung sind Delaunay-Triangulierungen, welche nach dem russischen Mathematiker Boris Nikolaevich Delaunay benannt sind. Delaunay-Triangulierungen sind Triangulierungen bei denen der kleinste Innenwinkel der Dreiecke maximal ist. Sie sind also *MaxMin* Triangulierungen bezüglich ihrer Innenwinkel. Diese Art der Trian-

gulierung zeichnet sich durch ihre besondere Gleichmäßigkeit aus.

Delaunay-Triangulierungen werden üblicherweise über die Umkreisbedingung definiert. Sie besagt, dass im Umkreis eines Dreiecks der Triangulierung kein Punkt liegen darf, der nicht Teil dieses Dreiecks ist. Auch so definierte Triangulierungen sind aber in seltenen Fällen nicht eindeutig, nämlich, wenn vier Punkte der Menge kozirkular sind, man also einen Kreis finden kann, der gleichzeitig diese vier Punkte schneidet. In diesem Fall ist die Umkreisbedingung für beide Varianten der lokalen Triangulierung erfüllt, man nennt dies den neutralen Fall.

Eine weitere Art eine Delaunay-Triangulierung einer Menge zu definieren ist es, ihre Definition direkt aus dem Voronoi-Diagramm der Punkte abzuleiten. Benachbarte Knoten im Voronoi-Diagramm, also Knoten deren Voronoi-Regionen sich eine Kante teilen, werden mit Kanten verbunden um eine Delaunay-Triangulierung zu erzeugen. Auch Triangulierungen, die aus Voronoi-Diagrammen erzeugt werden, sind für den neutralen Fall nicht eindeutig. Die Konstruktion aus Voronoi-Diagrammen ist in der Praxis dennoch eine weit verbreitete Methode, um Delaunay-Triangulierungen zu berechnen.

Es existiert allerdings auch eine Definition für die Delaunay-Triangulierung, die selbst im neutralen Fall eine eindeutige Triangulierung erzeugt. Eine Delaunay-Triangulierung wird dabei als optimale Triangulierung im Bezug auf das *MaxMin*-Innenenwinkel-Kriterium definiert. Eine Triangulierung ist hier dann optimal, wenn der Vektor der kleinsten Innenwinkel ihrer Dreiecke lexikographisch strikt kleiner ist als der Vektor der Innenwinkel aller anderen Triangulierungen, die auch das Kriterium erfüllen[Hjelle and Dæhlen, 2006, S.51].

Kapitel 2

Beschreibung des Verfahrens

Ziel der praktischen Arbeit war es, ein Programm zu implementieren, welches dreidimensionale Punktdaten einliest, diese trianguliert und sie anschließend mit einem eigens entwickelten Reduktionsverfahren ausdünnt. Die Implementierung umschließt dabei auch Funktionalität zur dreidimensionalen Darstellung der berechneten Triangulierungen. Der Programmcode wurde vollständig in C++ geschrieben.

Im folgenden Kapitel wird zunächst näher auf die verwendeten Programmbibliotheken eingegangen. Im Anschluss darauf wird die Idee hinter dem Ausdünnverfahren und seine algorithmische Umsetzung erläutert.

2.1 Erläuterungen zu den verwendeten Bibliotheken

2.1.1 Die *CGAL* Bibliothek

Die wichtigste Bibliothek für die Umsetzung des praktischen Teils der Arbeit ist *CGAL*. Die „Computional Geometry Algorithms Library“ stellt unserer Anwendung Funktionen und Datentypen zur Erstellung, Speicherung und Bearbeitung von Delaunay-Triangulierungen bereit. Sie wird durch ein duales Lizenzierungs-Schema vertrieben. Dieses umfasst sowohl Open-Source-Lizenzen nach GPL/LGPL-Vertragsbestimmungen als auch kommerzielle Lizenzen. Es gibt natürlich eine Reihe an anderen Bibliotheken, welche auch im Stande sind Delaunay-Triangulierungen effizient zu berechnen. *CGAL* überzeugt allerdings durch eine Vielzahl an Eigenschaften, welche in einer sol-

chen Kombination bei keiner der konkurrierenden Softwarepakete auftreten. Auf einige dieser Eigenschaften soll im Folgenden kurz eingegangen werden, um die Wahl der Bibliothek zu begründen.

Das erste Alleinstellungsmerkmal der Bibliothek, welches hier hervorgehoben werden muss, ist ihre umfassende Funktionalität. Neben einfachen Delaunay-Triangulierungen kann *CGAL* auch Delaunay-Triangulierungen mit Nebenbedingungen und einfache Triangulierungen mit oder ohne Nebenbedingungen erzeugen und verwalten. Außerdem beinhaltet die *CGAL*-Bibliothek Pakete, welche zum Erstellen von Oberflächen-Meshes und dreidimensionalen Triangulierungen dienen. Diese könnten als Basis für zukünftige Reduktions-Verfahren dienen. Die Bibliothek implementiert außerdem mathematische Operationen, welche wichtig für das Projekt sind und anderenfalls aus zusätzlichen Bibliotheken importiert, oder selbst implementiert werden müssten. Die Implementierung der Triangulierungs-Datenstruktur bietet alle Zugriffsoperatoren auf Dreiecke, Kanten und Knoten, die benötigt werden, um den Algorithmus effizient zu gestalten.

Die *CGAL*-Bibliothek ist des Weiteren hervorragend und lückenlos dokumentiert. Das *User Manual* ist dabei nach Paketen unterteilt. Hier werden für die Verwendung der Pakete wichtige Konzepte erläutert. Zusätzlich stehen auch offizielle Code-Beispiele bereit, welche das einarbeiten erleichtern. Die gesamte API der Bibliothek ist ihrem *Reference Manual* einsehbar, es bietet genaue Informationen zu allen Klassen und Methoden.

Es sollte außerdem die Cross-Plattform-Kompatibilität der Bibliothek angemerkt werden. Sie erlaubt es uns den Quellcode auf Windows, Unix-Systemen und MacOS X zu kompilieren und auszuführen. Auf ihre Plattformunabhängigkeit wurde auch bei der Auswahl der anderen verwendeten Bibliotheken geachtet. *CGAL* besitzt zudem lediglich eine Abhängigkeit von der *Boost* Bibliothek, was die Einrichtung von *CGAL* einfach gestaltet.

Als einzigen negativen Aspekt der Bibliothek kann man ihren enormen Umfang anmerken, welcher sie komplexer macht als Bibliotheken, welche Delaunay-Triangulierungen als Kern-Funktionalität haben. Dieser Punkt kann aber durch ihre Modularität und die genaue Dokumentation der Bibliothek kompensiert werden.

2.1.2 Triangulierung mit *CGAL*

CGAL bietet uns die Möglichkeit zweidimensionale sowie dreidimensionale Triangulierungen zu erstellen und zu verwalten. Im folgenden Abschnitt

befassen wir uns jedoch nur mit Triangulierungen von 2D-Punktmengen. Genauer befassen wir uns mit 2D-Delaunay-Triangulierungen. Diese leiten sich aber wie alle Spezialformen der Triangulierung in *CGAL* aus der einfachen Triangulierungs-Klasse `Triangulation_2<Traits, Tds>` ab. Eine Triangulierung wird hier über ihre zwei Template-Parameter konstruiert.

Der Parameter *Traits* steht dabei für Klassen, die das Konzept `TriangulationTraits_2` modellieren. Diese Klassen stellen die geometrischen Objekte, aus denen die Triangulierung aufgebaut ist, bereit. Des Weiteren definieren sie auch bestimmte Operationen auf diesen Objekten. Für unsere Arbeit nutzen wir als *Trait*-Klasse `Projection_traits_xy_3<K>`. Diese erlaubt es uns, 3D-Punkte in die Triangulierung einzufügen, welche intern als 2D-Punkte behandelt werden. Dies ermöglicht eine einfache Verwaltung der Projektion.

Der zweite Parameter wird durch Klassen, die das Konzept `Triangulation_data_structure_2<Vb,Fb>` modellieren, besetzt. Diese Klassen werden wiederum durch zwei Templateparameter genauer bestimmt und kapseln die Knoten und Dreiecke der Triangulierung. Die Templateparameter der Datenstruktur-Klasse werden durch *Vertex*- und *Face*-Klassen ausgefüllt, welche wieder die *Trait*-Klassen als Template besitzen. Die Bibliothek ermöglicht es zudem, die Knoten und Dreiecke der Triangulierung über angepasste Klassen mit zusätzlichen Informationen zu versehen.

Punkte können in einer Instanz der Triangulierungs-Klasse einzeln, über die `push_back` Methode oder gestaffelt über die `insert` Methode eingefügt werden. Das entfernen von Knoten erfolgt über `remove`, dieser Methode übergibt man als Argument einen *Handle*, der auf den zu entfernend Knoten zeigt. Durchlaufen kann man alle Knoten der Instanz mit einem von der Klasse bereitgestellten *Iterator*. Des weiteren bieten *CGAL*-Triangulierungen dem Nutzer die Möglichkeit, alle Dreiecke, die an einem bestimmten Knoten anliegen, über *Zirkulatoren* zu durchschreiten. Solche *Zirkulatoren* instanziiert man über einen *Handle* des jeweiligen Punktes. Man nutzt sie im Prinzip wie herkömmliche *Iteratoren*, allerdings muss der Endpunkt des Durchlaufs hier selbst verwaltet werden.

Die in diesem Abschnitt aufgelisteten Informationen finden sich genauer im *CGAL*-Manual im Abschnitt *2D-Triangulation* [Yvinec, 2016].

2.1.3 Grafische Darstellung der Triangulierungen

Zur grafischen Darstellung der Triangulierung wird das *OpenGl*-Framework genutzt. Das Fenstermanagement wird durch die *SDL2*-Bibliothek geregelt. Das Anzeige-Fenster kann wie von anderen 3D-Viewer-Applikationen gewohnt gesteuert werden. Folgende Funktionen sind gegeben: Hinein- und Herauszoomen der Kamera in bzw. aus der dargestellten Szenerie, Rotieren der Kamera um die Szenerie und Verschieben der Kamera in der Szenerie. Die Daten werden im modernen *OpenGl*-Stil durch *Vertex-Array-Objekte* an die Grafikkarte übermittelt. Das Vertex-Shader-Programm erlaubt die Übergabe von Matrizen, welche zur Manipulation der Kamera und zur Konfiguration der perspektivischen Projektion dienen. Diese Matrizen werden dabei mit Hilfe der *OpenGL-Mathematics*-Bibliothek erstellt und manipuliert.

2.2 Das Ausdünnverfahren

2.2.1 Zugrunde liegende Idee

Ziel eines jeden Ausdünnverfahrens für Daten ist es, Ursprungs-Daten so zu reduzieren, dass es zu keinem Informationsverlust kommt. Es wird versucht, redundante beziehungsweise unwichtige Teile des Datensatzes zu identifizieren, um diese anschließend zu ersetzen oder zu entfernen. Im Folgenden wird ein Verfahren vorgestellt, welches, durch Zuhilfenahme von zweidimensionalen Triangulierungen, dieses Ziel erreicht.

Um ein solches Verfahren zu konzeptionieren stellt sich zunächst die Frage, welche Datenpunkte eines 3D-Laserscans keine wichtigen Informationen repräsentieren. Stellen wir uns dazu die rekonstruierte Oberfläche des durch einen Laserscanner eingefangen Objekts oder Terrains vor. Eine Technik, um die Oberfläche einer solchen Punktemenge zu approximieren, ist es, diese zunächst auf eine Ebene zu projizieren und die dadurch entstehenden zweidimensionalen Punkte zu triangulieren. Im Anschluss wird jeder Punkt des zweidimensionalen Triangulations-Graphen wieder um die durch die Projektion verlorene Komponente ergänzt, um den Graphen in die dritte Dimension anzuheben. Nach diesem Vorgang beschreiben die Flächen der Dreiecke der Triangulierung die Oberfläche des durch die Punkte modellierten Objekts.

Wenn aus einer solchen Triangulation ein Punkt entfernt wird, entstehen nach Neutriangulierung der nun vorhandenen Lücke weniger Dreiecke mit größerer Oberfläche. Um die Topografie des Objekts zu erhalten, müssen

die neuen Dreiecke möglichst gut der alten Modellierung der Oberfläche entsprechen. Diese Voraussetzung erreicht man, wenn die vor dem Löschen des Punktes bestehenden Dreiecke eine planare Fläche bilden. Die Topografie der Triangulation ändert sich in diesem Fall nicht, da die Kanten der alten und der neuen Dreiecke alle in der gleichen Ebene liegen.

Dieser Idealfall, dass alle Punkte auf einer Ebene liegen, tritt in der Realität allerdings so gut wie nie auf. Selbst kleine Messfehler sorgen dafür, dass nicht alle Punkte die selbe Ebenengleichung erfüllen. Zudem sind kleine Unebenheiten, welche von einem Laserscanner vielleicht erfasst werden, für ein 3D-Modell des eingescannten Terrains oder Objekts oft nicht relevant. Im hier vorgestellten Verfahren sind deshalb auch Cluster von Punkten, welche nur nahezu eine Fläche beschreiben, Kandidaten für die Reduktion. Nach dieser Feststellung stellt sich folgende Frage: Wie überprüft man ob alle Punkte des Clusters ungefähr auf einer Ebene liegen? Ob alle Punkte in einem Bereich diese Eigenschaft erfüllen, wird im vorgestellten Verfahren über die Normalenvektoren $\vec{n}_i$ der durch die Dreiecke $\triangle_i$ aufgespannten Ebenen getestet. Diese Vektoren erhält man wie folgt. Wir bezeichnen die Eckpunkte von $\triangle_i$ mit A_i, B_i, C_i, die Ortsvektoren $\vec{u}_i = \overrightarrow{A_iB_i}$ und $\vec{v}_i = \overrightarrow{A_iC_i}$ repräsentieren demnach zwei Kanten des Dreiecks. Einen Normalenvektor des Dreiecks berechnet man also, indem man das Kreuzprodukt zwei seiner Kanten bildet:

$$\vec{n}_i = \vec{u} \times \vec{v} = \begin{pmatrix} u_2 \cdot v_3 - u_3 \cdot v_2 \\ u_3 \cdot v_1 - u_1 \cdot v_3 \\ u_1 \cdot v_2 - u_2 \cdot v_1 \end{pmatrix} \tag{2.1}$$

Um zu quantifizieren, ob die Ebenen, in denen die Dreiecke liegen, sich ähneln, verwenden wir den Winkel zwischen ihren Normalenvektoren. Den Kosinus des Winkels θ zwischen den zwei Normalen $\vec{n}_i, \vec{n}_j$ erhält man über ihr Skalarprodukt:

$$\cos(\theta) = \frac{\vec{n}_i \cdot \vec{n}_j}{|\vec{n}_i| \ |\vec{n}_j|} \tag{2.2}$$

Mit $|\vec{n}_i|, |\vec{n}_j|$ bezeichnet man die Länge der beiden Vektoren. Man be-

rechnet also das Skalarprodukt der normalisierten Vektoren. Aus der Umkehrfunktion des Kosinus erhält man den gesuchten Winkel. Für uns gilt: Je kleiner θ, desto ähnlicher sind sich die Ebenen in denen die Dreiecke liegen. Beträgt der Winkel 0°, so liegen die Dreiecke auf einer Ebene. Wie oben bereits angeschnitten, müssen alle an einem Punkt anliegenden Dreiecke lediglich ungefähr auf einer Ebene liegen, damit der Punkt entfernt werden kann.

2.2.2 Beschreibung des Verfahrens

Der hier gewählte Ansatz, um zu Testen, ob Dreiecke auf einer Ebene liegen, ist es jedes Dreieck, das an einem Punkt anliegt, mit seinen direkten Nachbarn zu vergleichen. Zwei Dreiecke werden dabei als direkte Nachbarn bezeichnet, wenn sie sich eine gemeinsame Kante teilen. Liegt der Winkel zwischen den Normalenvektoren für alle direkten Nachbarn eines Punktes unter einer zuvor festgelegten Schranke α, so kann der Punkt entfernt werden. Der Algorithmus muss somit aus zwei Schleifen bestehen. Die äußere Schleife iteriert über die Knoten des Triangulations-Graphen und die innere Schleife über die Dreiecke, die am jeweiligen Knoten anliegen. Die Iteration über die Dreiecke erfolgt mithilfe des von *CGAL* bereitgestellten Zirkulators für am Punkt anliegende Dreiecke.

Der nachfolgende Algorithmus 1 beschreibt die Vorgehensweise. Der Funktionsparameter α, welcher die Schranke für die Winkel zwischen den Normalen angibt, ist frei wählbar. Er sollte so gesetzt werden, dass der Algorithmus ein gewünschtes Ausmaß an Ausdünnung erzielt. Abbildung 2.1 zeigt wie eine solche Ausdünnung für ein kleines α aussehen kann. Signifikante Kanten bleiben im rechten Bild erhalten, wohingegen Bereiche mit kleinen Unebenheiten und ebene Flächen, durch weniger und größere Dreiecke zusammengefasst werden.

Den Winkel α manuell zu bestimmen ist allerdings in der Regel nicht praktikabel, da das Ergebnis für ein bestimmtes α von Datensatz zu Datensatz stark schwanken kann. Deshalb wird das Verfahren zusätzlich in einen iterativen Prozess, wie in Algorithmus 2 beschrieben, eingebettet. Dieser wendet Algorithmus 1 mit in jedem Schritt größer werdender Schranke α auf die Triangulation an, welche in jedem Schritt weniger Punkte enthält, bis eine Reduzierung um den gewünschten Prozentsatz erfolgt ist. Das Verfahren garantiert damit eine Ausdünnung des Ursprungs-Datensatzes um mindestens den gewünschten Prozentsatz. Abbildung 2.2 zeigt das Resultat einer solchen

Ausdünnung. Das Inkrement, um das die Schranke pro Durchlauf erhöht werden soll, ist frei wählbar und abhängig davon, wie präzise das Reduktions-Ziel erreicht werden soll. Je kleiner man die Inkrement setzt, desto genauer wird die angeforderte Reduktion erreicht. Allerdings bedeutet ein kleineres Inkrement auch, dass die Schleife öfters durchlaufen werden muss.

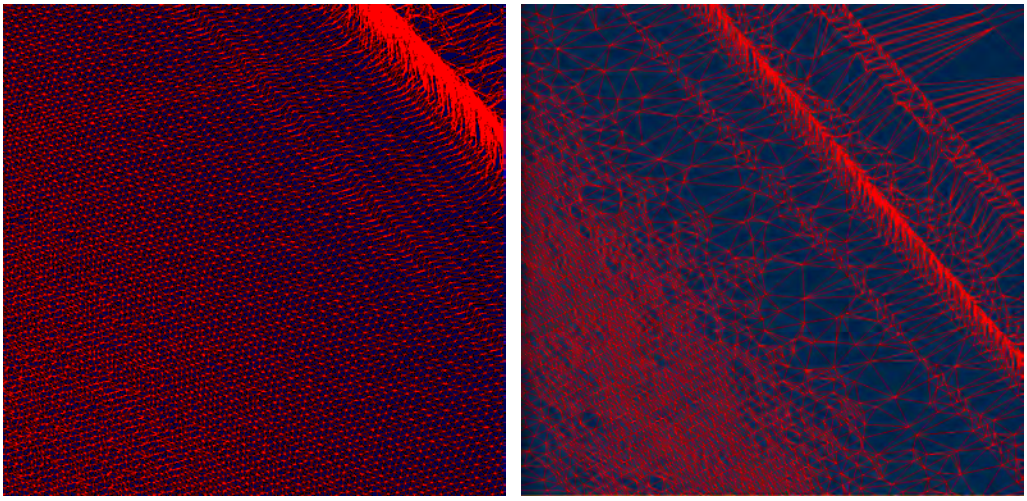
Im nachfolgenden Kapitel wird nach der Vorstellung der Struktur der Testdaten und der Ergebnisse des hier beschrieben einfachen Verfahrens noch auf eine Erweiterung des Verfahrens eingegangen werden. Dieses Verfahren wurde aufgrund von Besonderheiten der hier verwendeten Daten erstellt und soll deshalb erst nach Vorstellung dieser gezeigt werden.

Algorithmus 1 Ausdünn-Algorithmus

```
function ReduceTriangulation(triangulaton, α)
    v ← FirstVertex(triangulation)
    while v ≠ VerticesEnd(triangulation) do
        △cur ← FirstTriangle(v)
        △end ← FirstTriangle(v)
        remove ← true
        do
            △cur ← NextTriangle(△cur)
            △next ← NextTriangle(△cur)
            if ShareCommonEdge(△cur, △next) then
                n⃗ ← NormalVector(△cur)
                n⃗next ← NormalVector(△next)
                remove ← Angle(n⃗, n⃗next) ≤ α
            end if
        while △cur ≠ △end and remove
        if remove then
            RemoveFromTriangulation(v)
        end if
        v ← NextVertex(v)
    end while
end function
```

Algorithmus 2 Iterative Ausdünnung

```
function IterativeReduction(triangulaton, redRate)
    initSize ← NumberOfVertices(triangulation)
    curSize ← initSize
    α ← 0
    while redRate < curSize ÷ initSize do
        α ← α + increment
        ReduceTriangulation(triangulation, α)
        curSize ← NumberOfVertices(triangulation)
    end while
end function
```



(a) Originaldaten (b) Ausdünnung mit $\alpha = 5$

Abbildung 2.1: Veranschaulichung der Ausdünnung

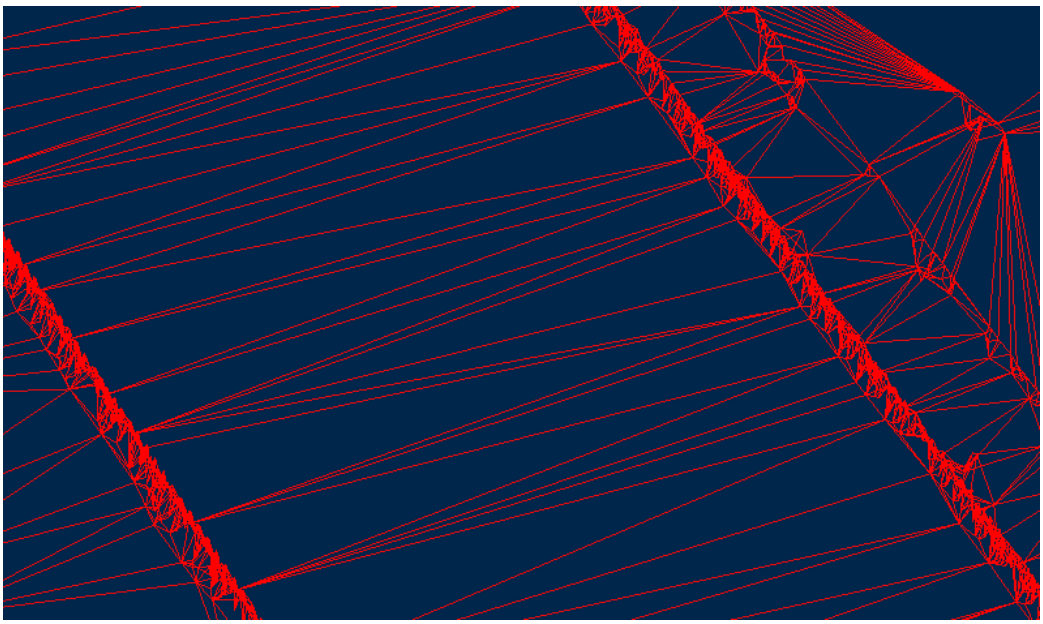


Abbildung 2.2: Ausdünnung um 85% durch Iteration

Kapitel 3

Besprechung der Ergebnisse

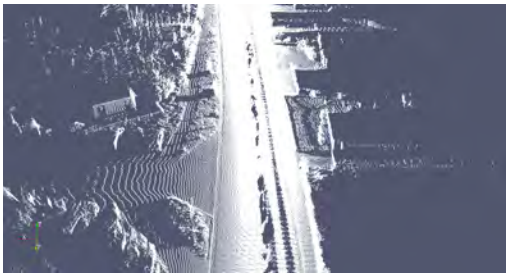
In diesen Kapitel werden zunächst kurz die zum Testen des Verfahrens verwendeten Daten beschrieben. Im Anschluss daran werden die Ergebnisse des Verfahrens anhand von Bildern erläutert.

3.1 Informationen zu den Verwendeten Testdaten

Bei den Daten, anhand derer das Verfahren zu Ausdünnung der Punktmengen getestet wird, handelt es sich um Punktwolken, welche wie üblich über ein LiDAR-Verfahren erfasst wurden. Sie bilden verschiedene Straßenverläufe ab. Die Scanvorrichtung wurde am Heck des Fahrzeugdaches nach unten gerichtet angebracht, um Straßen während der Fahrt zu erfassen.

Zu jedem einzelnen Punkt des Datensatzes stehen die folgenden Informationen bereit: Die x-, y-, z- Koordinaten des jeweiligen Punktes und eine Scanlinien-Nummer, welche beschreibt, zu welchem Zeitpunkt der Punkt erfasst wurde. Außerdem besitzt jeder Punkt einen Wert, welcher die an dieser Stelle gemessene Intensität angibt. Er gibt an, in welchem Ausmaß die bestrahlte Oberfläche den Laserstrahl zurückwirft. Dazu ist noch bekannt, ob es sich bei einem Punkt um den Mittelpunkt der Scanlinie handelt. Gespeichert sind die Daten in einer einfachen *csv*-Datei ohne Header-Informationen.

(a) Mehrspurige Straße



(b) Urbane Straße

Abbildung 3.1: Nicht reduzierte Test-Daten

3.2 Visuelle Auswertung des Verfahrens

Die visuelle Auswertung des Verfahren erfolgt anhand zweier unterschiedlicher Test-Datensätze. Diese wurden gewählt, da sie die Stärken und Schwächen des Verfahrens gut verdeutlichen und im Bezug auf viele Straßenscans repräsentativ sind. Der erste Datensatz 3.1a bildet eine mehrspurige Straße ab. Die beiden Richtungsfahrbahnen sind durch Leitplanken voneinander getrennt. Die linke Fahrbahn wurde dabei nicht vollständig eingefangen. Der zweite Datensatz 3.1b zeigt eine urbane Szenerie. Die Straße wird von einem niedrigen Bordstein gesäumt, auf beiden Straßenseiten befinden sich ein Bürgersteig und Häuser, welche auf Grund der Position und Neigung des Scanners allerdings abgeschnitten sind. Zudem zeichnen sich auf der rechten Fahrbahnseite Artefakte ab, welche auf Fahrzeuge im Gegenverkehr zurückzuführen sind.

Gehen wir jetzt auf die Resultate ein, die das in Algorithmus 2 beschriebene iterative Verfahren liefert, wenn es auf die eben beschriebenen Daten angewendet wird.. Betrachten wir dazu zunächst eine Ausdünnung um 90% der in Abbildung 3.1a abgebildeten mehrspurigen Straße. Die Abbildung 3.2

zeigt gut, dass selbst bei einer erheblichen Reduktion der Ursprungsdaten die Begrenzung am Rand der Fahrbahn durchgängig erhalten bleibt. Die Punkte, welche die Fahrbahn beschreiben, wurden hingegen vollständig eliminiert, da die Fahrbahn natürlich eine Fläche bildet. Auch viele Punkte, welche die Gegenfahrbahn darstellen, konnten entfernt werden.

Kommen wir nun zur Auswertung der Ausdünnung der in Abbildung 3.1b dargestellten urbanen Straßenszenerie. Grafik 3.3 zeigt eine iterative Ausdünnung mit verschiedenen Ausdünnraten. Die blauen Punkte bleiben bei einer Rate von 70% bestehen, die grünen Punkte bei einer Rate von 80% und die roten Punkte bei einer Rate von 90%. Hier zeigt sich, dass vertikale Flächen in den hier behandelten Testdaten mit dem vorgestellten Verfahren nicht reduzierbar sind. Dies liegt an der Art der Projektion, welche für Punkte, die sich vertikal überlagern, nicht geeignet ist, da das aus den projizierten Punkten resultierende Dreiecks-Netz die Oberfläche der vertikalen Flächen nicht richtig approximiert. Bei Scandaten mit einem großen Anteil an vertikalen Flächen muss deswegen darauf geachtet werden, dass die Ausdünnrate nicht zu hoch angesetzt wird. Sonst könnten wichtige Informationen, wie die Lage des Bordsteins, verloren gehen, weil anstelle der vertikalen Flächen solche Punkte entfernt werden. Mit einer Reduktionsrate von 70% erhält man für die hier gezeigte urbane Straße ein gutes Ergebnis. Der Bordstein zeichnet sich vollständig ab und Einfahrten werden korrekt ausgespart. Punkte der Straßenfläche sowie des Bürgersteigs werden fast vollständig entfernt. Allerdings können die Artefakte auf der rechten Seite der Fahrbahnhälfte nicht vollständig entfernt werden.

3.2.1 Modifizierte Version des Verfahrens

Die Inklusion des zusätzlichen Parameters *Intensitätswert* i, in dem in Abschnitt 3.1 beschriebenen Dateiformat, erlaubt uns, eine erweiterte Form von Algorithmus 1 anzuwenden. In ihm wird zusätzlich der Winkel zwischen den benachbarten Dreiecken, welche entstehen, wenn die zweidimensionalen Knoten durch i in die dritte Dimension angehoben werden, berechnet. Wenn sowohl dieser Winkel als auch der auf einfache Art, über die z-Koordinate berechnete, Winkel unter die gleiche Schranke α fallen, darf der Knoten entfernt werden. Diese erweiterte Version der Ausdünnung wird in den bereits beschriebenen iterativen Prozess aus Algorithmus 2 eingebettet. Da die hier verwendeten Daten für die *Intensität* ein hohes Maß an Rauschen aufzeigen, wurden sie vor der Ausdünnung zusätzlich gefiltert. Für die Filterung wurde

Abbildung 3.2: Mehrspurige Straße um 90% reduziert.

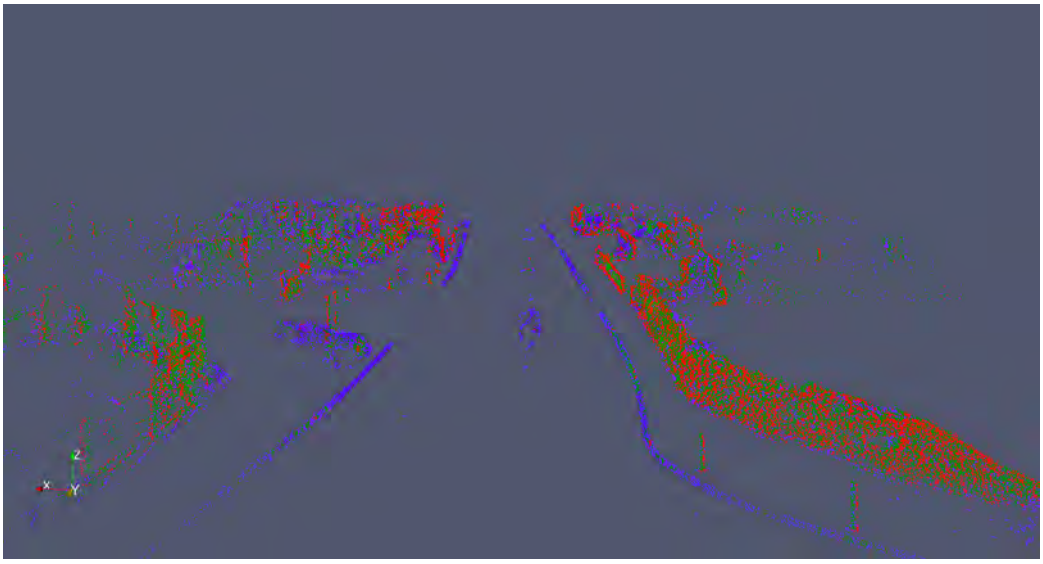


Abbildung 3.3: Urbane Straße, verschiedene Ausdünnraten.

Abbildung 3.4: Mehrspurige Straße mit modifizierter Ausdünnung.

ein Gauß-Filter gewählt, dieser wurde Scanlinien-weise auf die Daten angewandt. Das eindimensionale Fenster umfasste dabei sieben Punkte und für Sigma galt $\sigma = 1$. Im Folgenden Abschnitt betrachten wir die Ergebnisse des modifizierten Verfahrens für die bereits eingeführten Straßenabschnitte.

Abbildung 3.4 zeigt diese Version des Verfahrens angewandt auf die Daten der mehrspurigen Straße. Es wurde eine Ausdünnrate von 70% angesetzt. Die Leitlinie der rechten Fahrbahn, welche die Fahrstreifen kennzeichnet, hebt sich bei dieser Rate deutlich ab. Auch die Fahrbahnbegrenzung zu beiden Rändern der Fahrbahn wird klar hervorgehoben. Des weiteren bleiben alle Kanten, die durch die Anwendung der einfachen Version des Verfahrens erhalten bleiben würden, auch hier bestehen. Auf der Fahrbahn bleiben einige Punkte, welche keine wichtigen Informationen tragen, bestehen, da diese nicht vollständig entfernt werden konnten. Um bei einer Ausdünnung mit zusätzlichen Parametern ein gutes Ergebnis zu erzielen, muss die Reduktionsrate gesondert evaluiert werden. Eine Reduktion um 90%, die für das einfache Verfahren ein gutes Ergebnis liefert, würde hier Punkte, die für die korrekte Repräsentation notwendig sind, eliminieren.

Die letzte Abbildung, die hier besprochen werden soll, zeigt wieder den bereits bekannten urbanen Straßenabschnitt. Die roten Punkte in Abbildung 3.5 stellen das Ergebnis einer Reduktion um 70% dar, die blauen Punkte

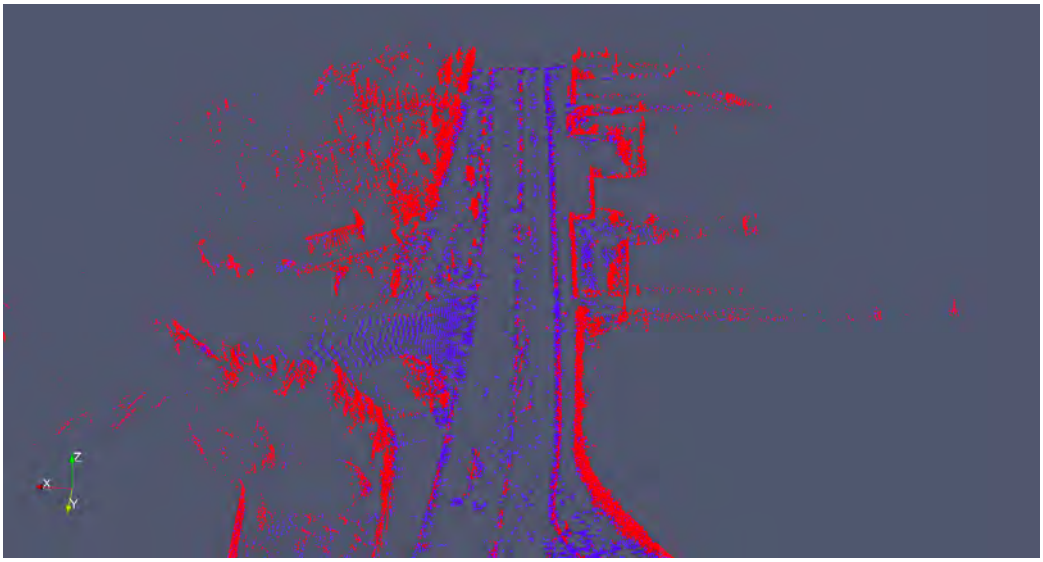


Abbildung 3.5: Urbane Straße mit modifizierter Ausdünnung

das einer Reduktion um 60% mit dem modifizierten Verfahren. Man sieht, dass für ein gutes Ergebnis, bei dem alle relevanten Punkte beibehalten werden, eine sehr niedrige Ausdünnrate angesetzt werden muss. Selbst dann ist das Ergebnis allerdings nicht wirklich zufriedenstellend, da viele Punkte, die augenscheinlich irrelevant sind, nicht verworfen werden.

Kapitel 4

Schluss

4.1 Zusammenfassung

In dieser Arbeit wurde ein Verfahren vorgestellt, welches dazu genutzt werden kann, den Umfang von dreidimensionalen Punktwolken zu verringern. Getestet wurde das, im Rahmen der Arbeit entwickelte, Programm anhand von 3D-Straßendaten. Es liefert dabei gute Ergebnisse für Straßenverläufe, in denen wenige vertikale Flächen präsent sind. Bei solchen Daten kann eine Reduktion von 90% oder mehr erreicht werden, ohne dass wichtige Details verloren gehen. Für Punktwolken, in denen viele vertikale Flächen mit eingefangen wurden, liefert das Verfahren, trotz einiger Schwierigkeiten, weiterhin gute Resultate. Die Ausdünnrate muss hier allerdings niedriger angesetzt werden. Es zeigt sich generell, dass Bereiche, in denen sich vertikal viele Punkte überlagern, mit dem vorgestellten Verfahren nicht reduzierbar sind. Dies ist der einfachen Art der Projektion zu schulden, welche vertikale Flächen verzerrt. Des Weiteren wurde gezeigt, dass das Verfahren mit kleinen Modifikationen auch auf 3D-Daten mit zusätzlichen Informationen angewandt werden kann. Für diese sind die Resultate allerdings nur dann verlustfrei, eine niedrigere Ausdünnrate angesetzt wird.

4.2 Ausblick

Wie bereits erwähnt, liefert das Verfahren für Daten, die keine oder nur wenige sich vertikal überlagernde Punkte enthalten, ein solides Ergebnis, auch bei großen Reduktionsraten. Die Schwächen des Verfahrens liegen primär in sei-

ner Fähigkeit, vertikale Flächen zu reduzieren. Somit wäre eine Möglichkeit, um das Verfahren effizienter zu gestalten, vor der Triangulierung der Punktewolke eine komplexere Projektion anzuwenden und so eine weniger verzerrte Approximation der Oberfläche zu erhalten. Auch komplexere Verfahren zur Oberflächenkonstruktion könnten angewandt werden, um diesen Aspekt des Vorgehens zu verbessern. Es ist sicher wichtig, noch weiterhin im Bereich der Ausdünnung von dreidimensionalen Datenmengen zu forschen, da die Entwicklung im Feld der 3D-Scan-Technologien stetig voranschreitet. Das sinnvolle Verkleinern der Datenmengen wird für viele Systeme das Arbeiten mit solchen Daten erleichtern.

Algorithmenverzeichnis

Abbildungsverzeichnis

Literaturverzeichnis

[De Berg et al., 2000] De Berg, M., Van Kreveld, M., Overmars, M., and Schwarzkopf, O. C. (2000). Delaunay triangulations. In *Computational geometry*, chapter 9, pages 191–218. Springer.

[Hjelle and Dæhlen, 2006] Hjelle, Ø. and Dæhlen, M. (2006). *Triangulations and applications*. Springer Science & Business Media.

[Large and Heritage, 2009] Large, A. R. and Heritage, G. L. (2009). Laser scanning–evolution of the discipline. *Laser Scanning for the Environmental Sciences. Chichester: Wiley*, 120.

[Yvinec, 2016] Yvinec, M. (2016). 2D triangulation. In *CGAL User and Reference Manual*. CGAL Editorial Board, 4.9 edition.

Erklärung

Hiermit erkläre ich, dass ich die vorliegende Arbeit selbstständig und nur unter Zuhilfenahme der ausgewiesenen Hilfsmittel angefertigt habe. Sämtliche Stellen der Arbeit, die im Wortlaut oder dem Sinn nach anderen gedruckten oder im Internet verfügbaren Werken entnommen sind, habe ich durch genaue Quellenangaben kenntlich gemacht.

Unterschrift	Datum