

Wörter: 9267

To-dos

1: Update mit Stand der Technik	2
2: Stand der Technik	4
3: Historisch?	4
4: mehr	4

Masterarbeit

Berechnung von charakteristischen Eigenschaften von botanischen Bäumen mithilfe von 3D-Punktwolken.

Name: Anton Wetzel

E-Mail: anton.wetzel@tu-ilmenau.de

Matrikelnummer: 60451

Studiengang: Informatik Master

Betreuer: Tristan Nauber

E-Mail: tristan.nauber@tu-ilmenau.de

Professor: Prof. Dr.-Ing. Patrick Mäder

E-Mail: patrick.maeder@tu-ilmenau.de

Fachgebiet: Data-intensive Systems and Visualization Group

Datum: 20.03.2024



**TECHNISCHE UNIVERSITÄT
ILMENAU**

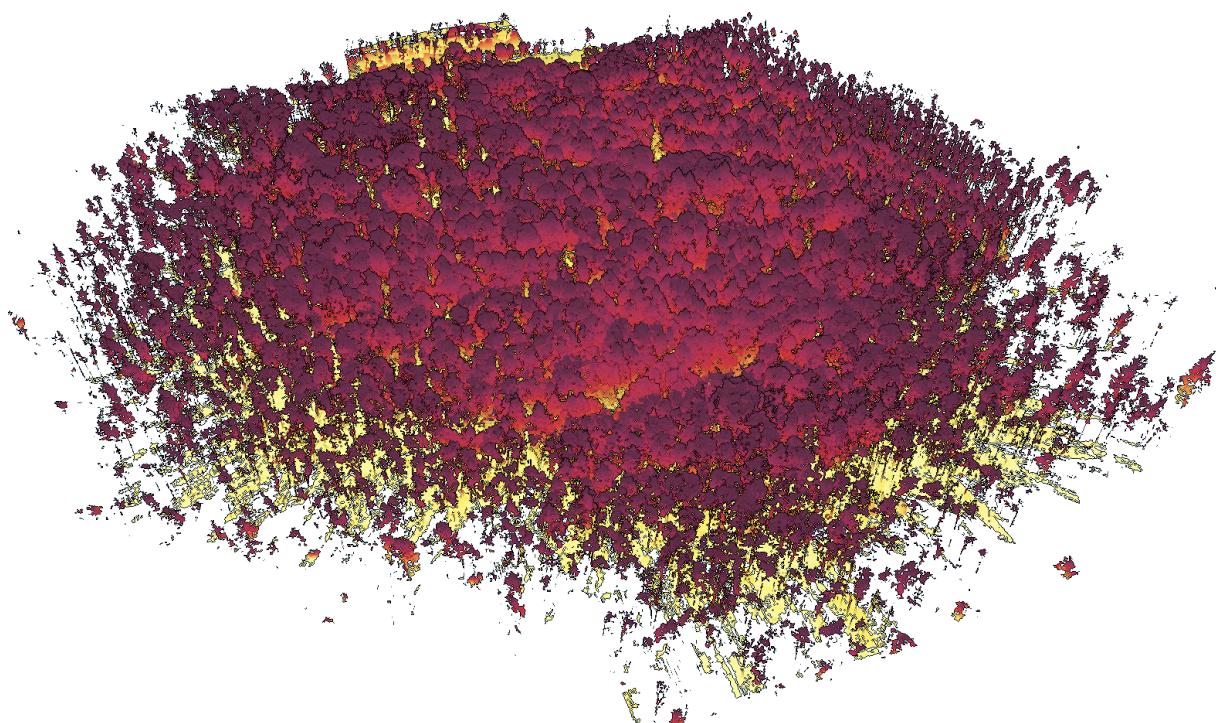
Abstrakt

Diese Arbeit beschäftigt sich mit der Verarbeitung und Visualisierung von Punktwolken von Waldstücken. Dabei wird der komplette Ablauf vom Datensatz bis zur Visualisierung der einzelnen Bäume mit relevanten Informationen durchgeführt.

Ein Datensatz ist dabei eine ungeordnete Liste von Punkten, für die nur die dreidimensionale Position bekannt ist.

Die Punkte werden automatisch in einzelne Bäume unterteilt, relevante Informationen für die Bäume und die einzelnen Punkte berechnet und alle Daten werden für die Visualisierung in Echtzeit vorbereitet.

Die vorgestellten Methoden und Algorithmen sind im zugehörigen Softwareprojekt umgesetzt, womit die Analyse und Visualisierung getestet wird.



Inhaltsverzeichnis

I. Einleitung	1
1. Motivation	1
2. Themenbeschreibung	1
3. Struktur der Arbeit	2
Abschnitt I: Einleitung	2
Abschnitt II: Stand der Technik	2
Abschnitt III: Methodik	2
Abschnitt IV: Implementierung	2
Abschnitt V: Auswertung	3
II. Stand der Technik	4
III. Methodik	5
1. Segmentierung der Punktfolge in Bäume	5
1.1. Ablauf	5
1.2. Bereiche bestimmen	5
1.3. Koordinaten bestimmen	7
1.4. Punkte zuordnen	7
2. Analyse von Segmenten	9
2.1. Eigenschaften	9
2.2. Eigenschaften für die Visualisierung	12
3. Triangulierung	12
3.1. Ziel	13
3.2. Ball-Pivoting Algorithmus	13
4. Visualisierung	19
4.1. Punkte	19
4.2. Detailstufen	22
4.3. Eye-Dome Lighting	24
IV. Implementierung	26
1. Technik	26
2. Benutzung	27
2.1. Installation	27
2.2. Ausführen	27
2.3. Import	27
2.4. Visualisierung	28
3. Struktur vom Quelltext	29
4. Format für eine Punktfolge	30
4.1. Daten	30
5. Import	31
5.1. Parallelisierung	32
6. Punkte	33
7. Segmente	34

7.1. Auswahl	34
7.2. Visualisierung	35
7.3. Exportieren	35
8. Detailstufen	35
 V. Auswertung	37
1. Testdaten	37
2. Importgeschwindigkeit	37
3. Segmentierung in Bäume	38
4. Analyse von Segmenten	40
5. Triangulierung	40
6. Visualisierung	41
7. Fazit	43
8. Ausblick	44
 VI. Appendix	45
1. Systemeigenschaften	45
2. Messwerte vom Import	46
3. KD-Baum	48
3.1. Konstruktion	48
3.2. Suche mit festem Radius	49
3.3. Suche mit fester Anzahl	49
3.4. Schnelle Suche	49
4. Baum (Datenstruktur)	49
4.1. Konstruktion	49
4.2. Suchanfrage	50
 Glossar	51
 Bibliographie	52

I. Einleitung

1. Motivation

Größere Gebiete wie Teile von Wäldern können mit 3D-Scanners abgetastet werden. Der Scanner ist dabei an einem Flugzeug oder einer Drohne befestigt, womit der gewünschte Bereich abgeflogen wird. Dabei entsteht eine Punktwolke, bei der für jeden Punkt die Position bekannt ist.

Aus den Punkten sind relevante Informationen aber nicht direkt ersichtlich. Eine manuelle Weiterverarbeitung ist durch die großen Datenmengen unrealistisch, weshalb automatisierte Methoden benötigt werden.

Die automatisierte Unterteilung in einzelne Bäume und die Berechnung der charakteristischen Eigenschaften der Bäume bildet dabei eine Grundlage für die Auswertung vom gesamten Waldstück. Durch die interaktive Visualisierung der berechneten Daten können diese intuitiv inspiert werden.

2. Themenbeschreibung

Das Ziel dieser Arbeit ist eine Erforschung des Ablaufs von einem Scan von einem Waldstücke bis zur Analyse der Daten mit zugehöriger interaktiven Visualisierung der Ergebnisse.

Ein Beispiel für so eine Visualisierung ist in Abbildung 1 gegeben. Es sind die Punkte vom Datensatz zu sehen, welche basierend auf der relativen Höhe im Baum eingefärbt sind. Dadurch ist der Boden in Gelb und zu den Baumspitzen hin ändert sich die Farbe zu Rot.

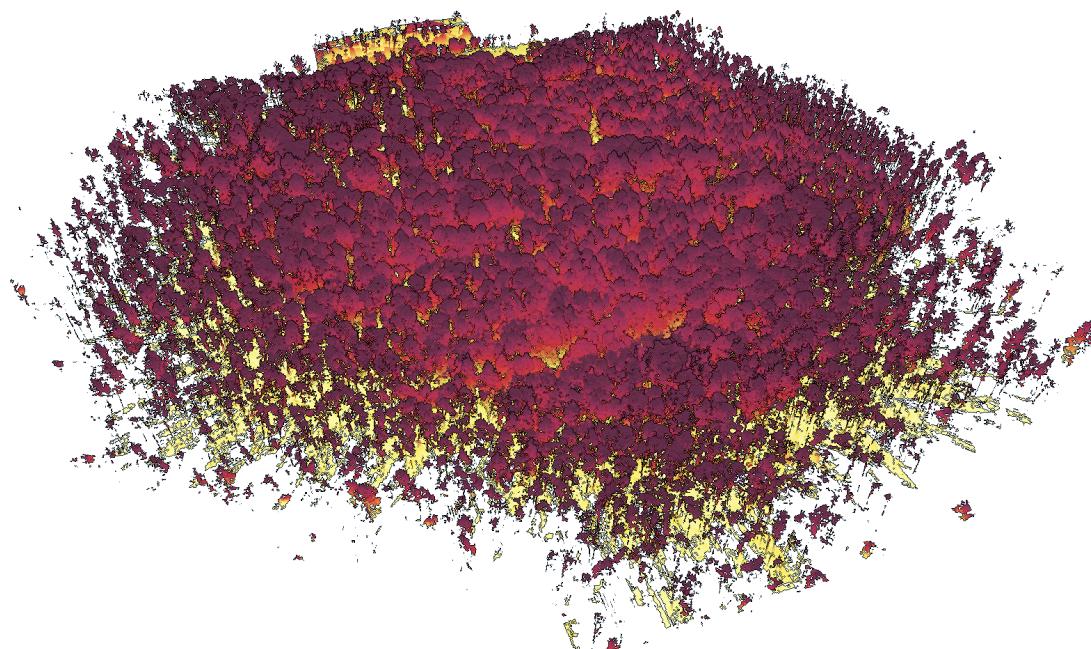


Abbildung 1: Waldstück mit Einfärbung der Punkte nach Höhe.

Als Eingabe wird der Datensatz vom Waldstücke benötigt. Dabei wird davon ausgegangen, dass ein Datensatz eine ungeordnete Menge von Punkten enthält, für die nur die Position im dreidimensionalen Raum bekannt ist. Je nach Scanner können die Punkte im Datensatz entsprechend der räumlichen Verteilung geordnet sein oder weitere Eigenschaften wie die Farbe der Punkte enthalten. Die weiteren Daten werden nicht bei der Analyse betrachtet, wodurch die Auswertung auch für Datensätze ohne die weiteren Daten funktioniert.

Für die Analyse der Daten muss die Menge der Punkte zuerst in einzelne Bäume segmentiert werden, dass Punkte vom gleichem Baum zum gleichem Segment gehören. Danach können die einzelnen Bäume ausgewertet werden. Durch die Beschränkung auf Waldstücke kann die Segmentierung und die folgende Auswertung auf Bäume spezialisiert werden.

Bei der Auswertung werden charakteristische Eigenschaften für die Bäume, aber auch für die einzelnen Punkte im Baum bestimmt. Für Bäume werden Eigenschaften bestimmt, welche den ganzen Baum beschreiben. Für Punkte werden die Eigenschaften für die lokale Umgebung mit den umliegenden Punkten bestimmt.

Die Visualisierung präsentiert die berechneten Ergebnisse. Dabei werden die Eigenschaften visuell zu den zugehörigen Bäumen oder Punkten zugeordnet und können interaktive inspiert werden. Dabei können die Punkte vom gesamten Datensatz oder einzelne Segmente mit den Punkten oder als Triangulierung angezeigt werden.

3. Struktur der Arbeit

Abschnitt I: Einleitung

In der Einleitung wird das Thema und Struktur der Arbeit vorgestellt.

Abschnitt II: Stand der Technik

Der Stand der Technik beschäftigt sich mit zugehörigen wissenschaftlichen und technischen Arbeiten. **To-do: Update mit Stand der Technik**

Abschnitt III: Methodik

Im Abschnitt Methodik werden die benutzten Algorithmen vorgestellt und erforscht. Dazu gehört die Segmentierung in einzelnen Bäume, Analyse und Triangulierung dieser und die technischen Grundlagen für die Visualisierung der Ergebnisse.

Abschnitt IV: Implementierung

Die Methodik ist die Grundlage für die Implementierung. Das Softwareprojekt mit der technischen Umsetzung der Algorithmen wird vorgestellt. Dafür wird die Bedienung vom Softwareprojekt, der Ablauf vom Import, das Anzeigen aller Punkte und von einzelnen Segmenten erklärt.

Abschnitt V: Auswertung

Schlussendlich werden die Algorithmen aus der Methodik mithilfe der Implementierung ausgewertet. Dafür werden die benutzten Datensätze vorgestellt, mit denen der Ablauf bis zur Visualisierung getestet wird. Die Auswertung enthält die gemessenen Werte und die daraus folgende Bewertung der Arbeit.

II. Stand der Technik

To-do: Stand der Technik

To-do: Historisch?

To-do: mehr

Punktwolken können mit unterschiedlichen Lidar Scanverfahren aufgenommen werden. Aufnahmen vom Boden oder aus der Luft bieten dabei verschiedene Vor- und Nachteile [1]. Bei einem Scan von Boden aus, kann nur eine kleinere Fläche abgetastet werden, dafür mit erhöhter Genauigkeit, um einzelne Bäume genau zu analysieren [2]. Aus der Luft können größere Flächen erfasst werden, wodurch Waldstücke aufgenommen werden können, aber die Datenmenge pro Baum ist geringer [3].

Nach der Datenerfassung können relevante Informationen aus den Punkten bestimmt werden, dazu gehört eine Segmentierung in einzelne Bäume [4] und die Berechnung von Baumhöhe oder Kronenhöhe [3].

Ein häufiges Format für Lidar-Daten ist das LAS Dateiformat [5]. Bei diesem werden die Messpunkte mit den bekannten Punkteigenschaften gespeichert. Je nach Messtechnologie können unterschiedliche Daten bei unterschiedlichen Punktwolken bekannt sein, aber die Position der Punkte ist immer gegeben. Aufgrund der großen Datenmengen werden LAS Dateien häufig im komprimierten LASzip Format [6] gespeichert. Die Kompression ist Verlustfrei und ermöglicht eine Kompressionsrate zwischen 5 und 15 je nach Eingabedaten.

LAStools [7] ist eine Sammlung von Software für die allgemeine Verarbeitung von LAS Dateien. Dazu gehört die Umwandlung in andere Dateiformate, Analyse der Daten und Visualisierung der Punkte. Durch den allgemeinen Fokus ist die Software nicht für die Verarbeitung von Waldteilen ausgelegt, wodurch Funktionalitäten wie Berechnungen von Baumeigenschaften mit zugehöriger Visualisierung nicht gegeben sind.

III. Methodik

1. Segmentierung der Punktfolge in Bäume

1.1. Ablauf

Für die Segmentierung werden alle Punkte in gleich breite parallele Scheiben entlang der Höhe unterteilt. Danach werden die Scheiben von Oben nach Unten einzeln verarbeitet, um die Segmente zu bestimmen. Dafür werden die Punkte in einer Scheibe zu zusammenhängenden Bereichen zusammengefasst. Mit den Bereichen werden die Koordinaten der Bäume bestimmt, welche in der momentanen Scheibe existieren. Die Punkte in der Scheibe werden dann dem nächsten Baum zugeordnet.

1.2. Bereiche bestimmen

Für jede Scheibe werden konvexe zusammenhängende Bereiche bestimmt, dass die Punkte in unterschiedlichen Bereichen einen Mindestabstand voneinander entfernt sind. Dafür wird mit einer leeren Menge von Bereichen gestartet und jeder Punkt zu der Menge hinzugefügt. Wenn ein Punkt vollständig in einem Bereich enthalten ist, wird der Bereich nicht erweitert. Ist der Punkt außerhalb, aber näher als den Mindestabstand zu einem der Bereiche, so wird der Bereich erweitert. Ist der Punkt von allen bisherigen Bereichen weiter entfernt, so wird ein neuer Bereich angefangen.

Dadurch entstehen Bereiche wie in Abbildung 2. Bei einer Baumspitze entsteht ein kleiner Bereich. Wenn mehrere Bäume sich berühren, werden die Bäume zu einem größeren Bereich zusammengefasst.



Abbildung 2: Beispiel für berechnete Segmente für zwei aufeinanderfolgende Scheiben.

Bei der Berechnung sind alle momentanen Bereiche in einer Liste gespeichert. Ein Bereich ist dabei eine Liste von Eckpunkten. Die Eckpunkte sind dabei sortiert, dass für einen Eckpunkt der nächste Punkt entlang der Umrandung vom Bereich der nächste Punkt in der Liste ist. Für den letzten Punkt ist der erste Punkt in der Liste der nächste Eckpunkt.

Um die Distanz von einem Punkt zu einem Bereich zu berechnen, wird der größte Abstand nach Außen vom Punkt zu allen Kanten berechnet. Für jede Kante mit den Eckpunkten $a = \begin{pmatrix} a_x \\ a_y \end{pmatrix}$ und $b = \begin{pmatrix} b_x \\ b_y \end{pmatrix}$ wird zuerst der Vektor $d = \begin{pmatrix} d_x \\ d_y \end{pmatrix} = b - a$ berechnet. Der normalisierte Vektor $o = \frac{1}{|d|} \begin{pmatrix} d_y \\ -d_x \end{pmatrix}$ ist orthogonal zu d und zeigt aus dem Bereich hinaus, solange a im Uhrzeigersinn vor b auf der Umrandung liegt. Für den Punkt p kann nun der Abstand zur Kante mit dem Skalarprodukt $o \cdot (p - a)$ berechnet werden. Wenn der Punkt auf der Innenseite der Kante liegt, ist der Abstand negativ. In Abbildung 3 ist eine Veranschaulichung gegeben.

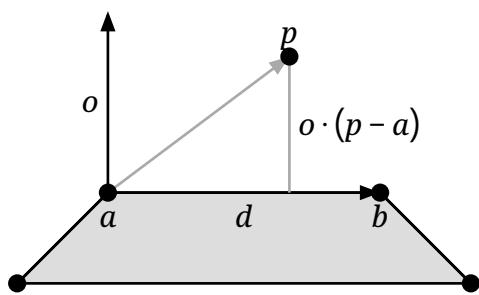


Abbildung 3: Berechnung vom Abstand vom Punkt p zur Kante zwischen a und b .

Um einen Punkt zu einem Bereich hinzuzufügen, werden alle Kanten entfernt, bei denen der Punkt außerhalb liegt, und zwei neue Kanten zum Punkt werden hinzugefügt. Dafür werden die Eckpunkte entfernt, bei denen der neue Punkt außerhalb der beiden angrenzenden Kanten liegt. An der Stelle, wo die Punkte entfernt wurden, wird stattdessen der neue Eckpunkt eingefügt. In Abbildung 4 ist das Ergebnis vom Austausch zu sehen.

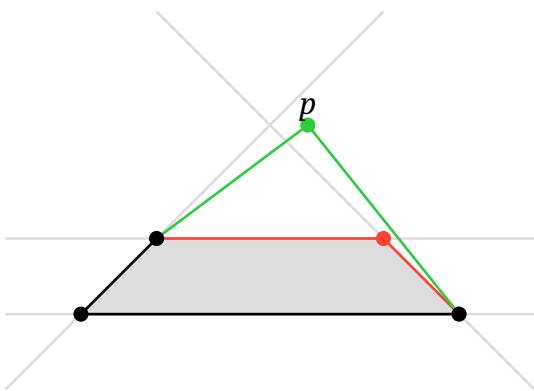


Abbildung 4: Hinzufügen vom Punkt p zum Bereich. Die Kanten in Rot werden entfernt und die Kanten in Grün werden hinzugefügt.

Nachdem alle Punkte zu den Bereichen hinzugefügt würden, werden die Bereiche gefiltert. Dafür werden alle Bereiche entfernt, deren Fläche kleiner als ein Schwellwert ist. Weil die Bereiche konvex sind, können diese trivial in Dreiecke wie in Abbildung 5 unterteilt werden und dann die Flächen der Dreiecke summiert werden.

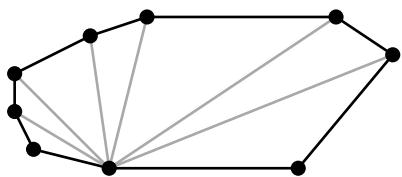


Abbildung 5: Unterteilung von einem Bereich in Dreiecke. Alle Dreiecke haben einen beliebigen Eckpunkt als ersten Punkt gemeinsam und die beiden anderen Punkte sind die Eckpunkte der Kanten ohne den ersten Punkt.

Weil die konvexe Hülle von allen Punkten in einem Bereich gebildet wird, können Bereiche sich überscheiden, obwohl die Punkte der Bereiche voneinander entfernt sind. Bei dem Hinzuzufügen von neuen Punkten werden die Bereiche sequentiell iteriert. Dabei wird bei überschneidenden Bereichen das erste präferiert, wodurch dieses weiter wächst. Um den anderen Bereich zu entfernen, werden Bereiche entfernt, deren Zentren in einem anderen Bereich liegen.

1.3. Koordinaten bestimmen

Für die Bäume der momentanen Scheibe werden die Koordinaten gesucht. Die Menge der Koordinaten startet mit der leeren Menge für die höchste Scheibe. Bei jeder Scheibe wird die Menge der Koordinaten mit den gefundenen Bereichen aktualisiert. Dafür werden für alle Bereiche in der momentanen Scheibe zuerst die Schwerpunkte wie in Abbildung 6 berechnet.

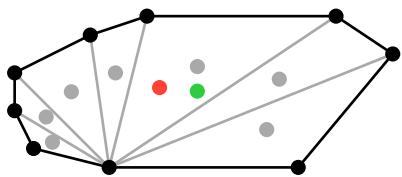


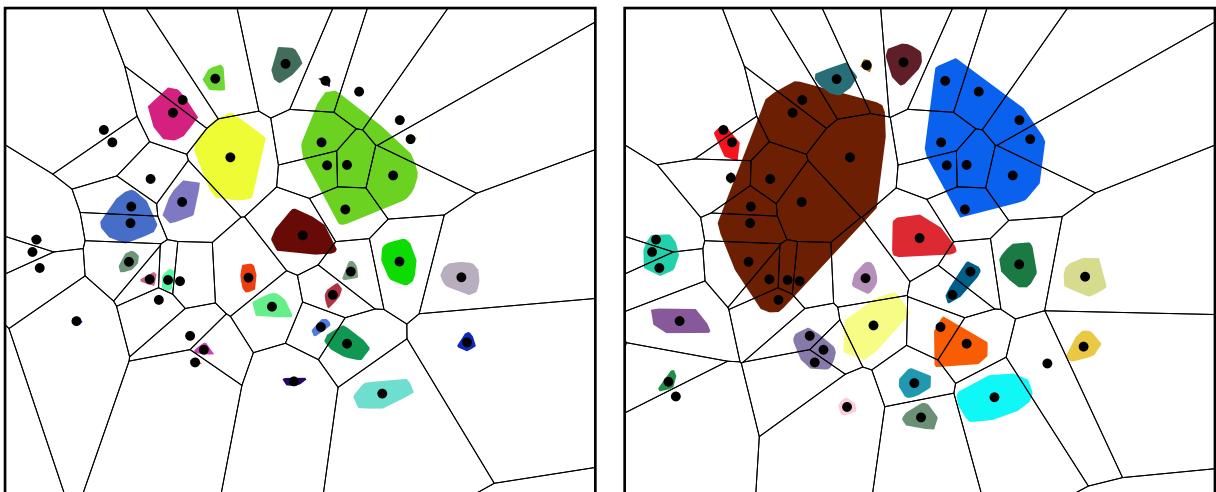
Abbildung 6: Berechnung vom Schwerpunkt von einem konvexen Bereich. Der korrekte Schwerpunkt in Grün ist die durchschnittliche Position der Schwerpunkte der Dreiecke nach der Fläche vom Dreieck gewichtet. In Rot ist die durchschnittliche Position der Eckpunkte.

Danach werden die Koordinaten aus den vorherigen Scheiben mit den Schwerpunkten von der momentanen Scheibe aktualisiert. Für jede Koordinate wird der nächste Schwerpunkt näher als eine maximale Distanz bestimmt. Wenn ein naher Schwerpunkt gefunden wurde, wird die Koordinate mit der Position vom Schwerpunkte aktualisiert. Wenn kein naher Schwerpunkt existiert, so bleibt die Position gleich.

Für alle Schwerpunkte, welche nicht nah an einen der vorherigen Koordinaten liegen, wird ein neues Segment angefangen. Dafür wird der Schwerpunkt zur Liste der Koordinaten hinzugefügt.

1.4. Punkte zuordnen

Mit den Koordinaten wird das Voronoi-Diagramm berechnet, welches den Raum in Bereiche unterteilt, dass alle Punkte in einem Bereich für eine Koordinate am nächsten an dieser Koordinate liegen. Für jeden Punkt wird nun der zugehörige Bereich im Voronoi-Diagramm bestimmt und der Punkt zum zugehörigen Segment zugeordnet. Ein Beispiel für eine Unterteilung ist in Abbildung 7 zu sehen.



(a) Höhere Scheibe

(b) Tiefere Scheibe

Abbildung 7: Berechnete Koordinaten für die Punkte mit zugehörigen Bereichen und Voronoi-Diagramm.

Der Ablauf wird für alle Scheiben durchgeführt, wodurch alle Punkte zu Segmenten zugeordnet werden. Ein Beispiel für eine Segmentierung ist in Abbildung 8 gegeben. Die unterschiedlichen Segmente sind durch unterschiedliche Einfärbung der zugehörigen Punkte markiert.

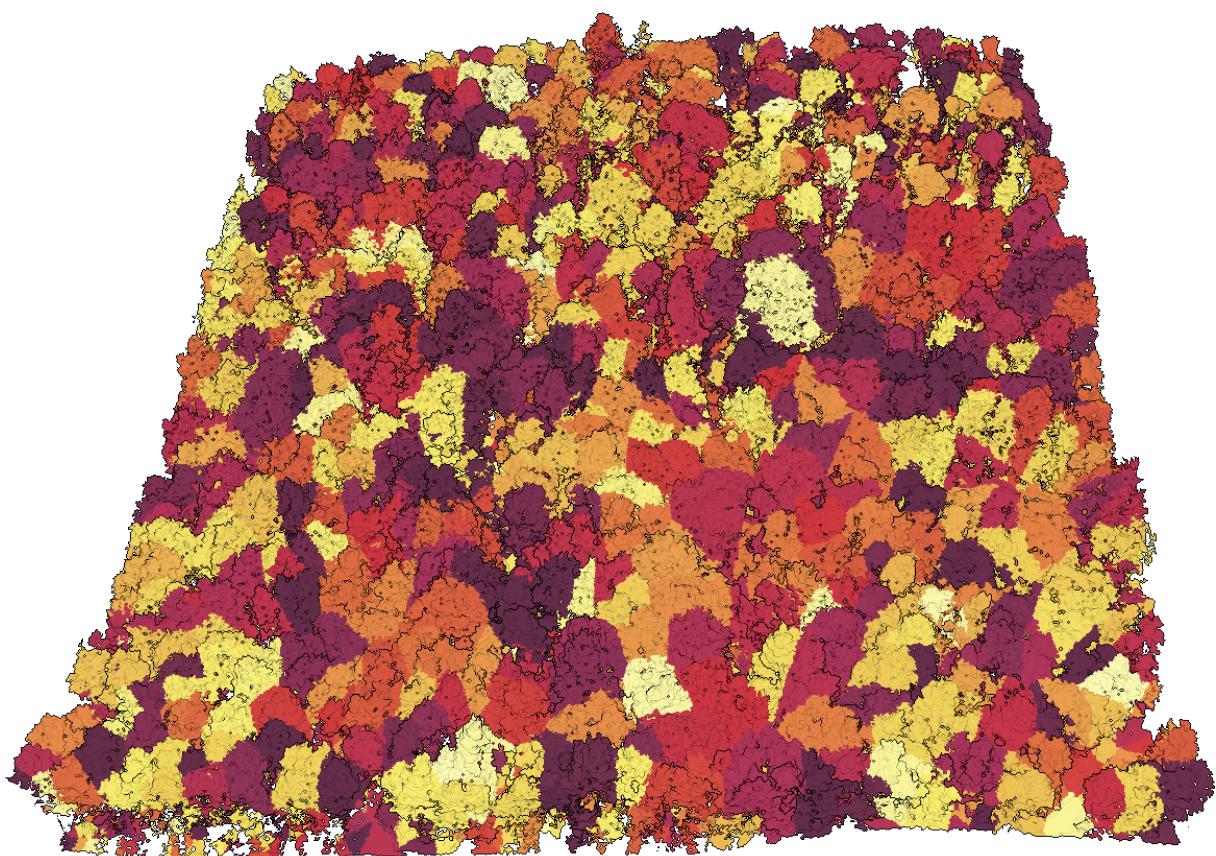


Abbildung 8: Segmentierung von einem Waldstück.

2. Analyse von Segmenten

2.1. Eigenschaften

Die charakteristischen Eigenschaften werden für jedes Segment einzeln berechnet. Dabei sind alle Punkte im Segment als Liste der Länge n verfügbar. Für den Punkt $i \in \mathbb{N}_0^{n-1}$ ist nur die globale Position $p_i = (p_{ix}, p_{iy}, p_{iz})$ gegeben. Die Punkte sind dabei ungeordnet, wodurch aufeinanderfolgende Punkte in der Liste weit voneinander entfernte Positionen haben können.

Um einen Punkt i zu analysieren, wird die zugehörige Nachbarschaft N_i benötigt. Die Nachbarschaft enthält dabei die nächsten Punkte nach Abstand sortiert. Dafür wird für alle Punkte ein KD-Baum erstellt. Mit diesem können effizient für eine Position p_i und ein beliebiges $k \in \mathbb{N}$ die k -nächsten Punkte bestimmt werden. Die Konstruktion und Verwendung vom KD-Baum wird in [Kapitel VI-3](#) erklärt. In der Nachbarschaft ist dann N_0 der ursprüngliche Punkt i , N_1 der nächste Punkt und N_{k-1} der $k - 1$ nächste Punkt.

2.1.1. Punkthöhe

Für jeden Punkt wird die relative Höhe im Segment bestimmt. Dafür wird zuerst für alle Positionen die Mindesthöhe h_{\min} und Maximalhöhe h_{\max} bestimmt.

$$h_{\min} = \min_{i \in \mathbb{N}_0^{n-1}} p_{iy} \quad h_{\max} = \max_{i \in \mathbb{N}_0^{n-1}} p_{iy}$$

Mit der Mindest- und Maximalhöhe kann für den Punkt i die relative Höhe h_i bestimmt werden.

$$h_i = \frac{p_{iy} - h_{\min}}{h_{\max} - h_{\min}}$$

Die relative Höhe liegt immer im Bereich $[0; 1]$ und wird größer, je höher der Punkt liegt. Ein Beispiel ist in Abbildung 9 zu sehen.

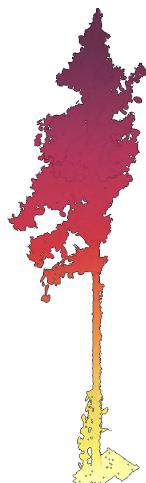


Abbildung 9: Punktwolke basierend auf der relativen Höhe der Punkte eingefärbt.
Der Farbverlauf geht von Gelb für den niedrigsten Punkt zu Rot für den höchsten Punkt.

2.1.2. Krümmung

Die Krümmung der ursprünglichen abgetasteten Oberfläche wird für jeden Punkt geschätzt. Dafür wird für den Punkten i die Verteilung der Positionen der Punkte in der Nachbarschaft N_i betrachtet. Zuerst wird für die Nachbarschaft der geometrische Schwerpunkt s_i bestimmt.

$$s_i = \frac{1}{k} * \sum_{j \in N_i} p_j$$

Mit dem Schwerpunkte kann die Kovarianzmatrix C_i bestimmt werden.

$$C_i = \frac{1}{k} * \sum_{j \in N_i} \begin{pmatrix} (p_{jx} - s_{ix})^2 & (p_{jx} - s_{ix}) * (p_{jy} - s_{iy}) & (p_{jx} - s_{ix}) * (p_{jz} - s_{iz}) \\ (p_{jy} - s_{iy}) * (p_{jx} - s_{ix}) & (p_{jy} - s_{iy})^2 & (p_{jy} - s_{iy}) * (p_{jz} - s_{iz}) \\ (p_{jz} - s_{iz}) * (p_{jx} - s_{ix}) & (p_{jz} - s_{iz}) * (p_{jy} - s_{iy}) & (p_{jz} - s_{iz})^2 \end{pmatrix}$$

Ohne die Verschiebung um s_i würde die globale Position der Punkte die Kovarianzmatrix beeinflussen. Von der Kovarianzmatrix werden die Eigenwerte und zugehörige Eigenvektoren bestimmt.

Die normierten Eigenvektoren v_{i0} , v_{i1} und v_{i2} bilden eine Orthonormalbasis und der zugehörige Eigenwert $\lambda_{i\alpha}$ für $v_{i\alpha}$ ist die Ausdehnung der Punkte entlang des Basisvektors. Der kleinste Eigenwert gehört zu Dimension mit der geringsten Ausdehnung. Je kleiner der kleinste Eigenwert, desto näher liegen die Punkte in der Nachbarschaft an der Ebene, aufgespannt durch die anderen beiden Eigenvektoren. Ein Beispiel für die Eigenvektoren in 2D ist in Abbildung 10 gegeben.

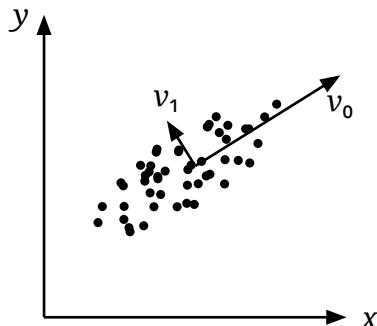


Abbildung 10: Eigenvektoren v_0 und v_1 der Kovarianzmatrix für eine Punktmenge. Die Länge der Vektoren ist anhängig vom zugehörigen Eigenwert.

Mit den Eigenwerten $\lambda_{i\alpha}$ absteigend nach größer sortiert wird die Krümmung c_i berechnet.

$$c_i = \frac{3\lambda_{i2}}{\lambda_{i0} + \lambda_{i1} + \lambda_{i2}}$$

c_i liegt dabei im abgeschlossenen Bereich $[0; 1]$, weil $0 \leq \lambda_{i0} \leq \lambda_{i1} \leq \lambda_{i2}$ ist. In Abbildung 11 in ein Beispiel für ein Segment mit den Punkten basierend auf der Krümmung eingefärbt.

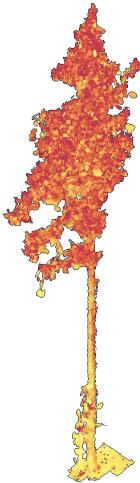


Abbildung 11: Punktwolke basierend auf der Krümmung eingefärbt.
Der Farbverlauf geht von Gelb für wenig bis Rot für maximale Krümmung.

2.1.3. Ausdehnung

Für die Berechnung der horizontalen Ausdehnung wird der Baum entlang der Horizontalen in 5 cm breite Scheiben S_α unterteilt.

$$S_\alpha = \{i \mid i \in \mathbb{N}_0^{n-1}, p_{iy} \in [h_{\min} + 0.05\alpha \text{ cm}] \}$$

Die Ausdehnung wird für jede Scheibe einzeln berechnet. Dafür wird zuerst der geometrische Schwerpunkt s_α der Punkte in der Scheibe entlang der x- und z-Achse berechnet.

$$s_\alpha = \frac{1}{|S_\alpha|} \sum_{i \in S_\alpha} (p_{ix}, p_{iz})$$

Mit dem Schwerpunkt wird die durchschnittliche Varianz v_α der Abweichung in der Scheibe berechnet.

$$v_\alpha = \frac{1}{|S_\alpha|} \sum_{i \in S_\alpha} |(p_{ix}, p_{iz}) - s_\alpha|$$

Die größte Varianz von allen Scheiben wird verwendet, um die Varianzen auf den Bereich $[0; 1]$ zu normieren. Für jeden Punkt wird die Varianz der zugehörigen Scheibe zugeordnet. In Abbildung 12 ist ein Segment mit den Punkten basierend der Ausdehnung der zugehörigen Scheibe eingefärbt.

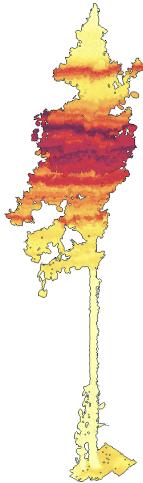


Abbildung 12: Punktwolke basierend auf der Ausdehnung eingefärbt.
Der Farbverlauf geht von Gelb für gerin-
ge bis Rot für größte Ausdehnung.

Die Ausdehnung eignet sich zur Unterscheidung von Stamm und Krone. Beim Stamm sind die Punkte näher einander, während bei der Krone die Punkte weiter verteilt sind. Für die Unterteilung wird die erste Scheibe von Unten gesucht, dessen normierte Varianz größer als ein Schwellwert ist.

2.2. Eigenschaften für die Visualisierung

Für die Visualisierung werden die Position, Orientierung und Größe von einem Punkte benötigt. Die Position ist in den Eingabedaten gegeben und die anderen Eigenschaften werden mit der lokalen Umgebung vom Punkt berechnet.

Für die Orientierung wird die Normale bestimmt, welche orthogonal zur geschätzten zugehörigen Oberfläche vom Punkt ist. Dafür werden die Eigenvektoren aus [Kapitel III-2.1.2](#) verwendet. Der Eigenvektor, welcher zum kleinsten Eigenwert gehört, ist dabei orthogonal zur geschätzten Ebene mit der größten Ausdehnung. Für die Punktgröße wird der durchschnittliche Abstand zu den umliegenden Punkten bestimmt. Dadurch werden die Punkte in Bereichen mit hoher Punktdichte kleiner. In Abbildung 13 ist ein Beispiel gegeben.

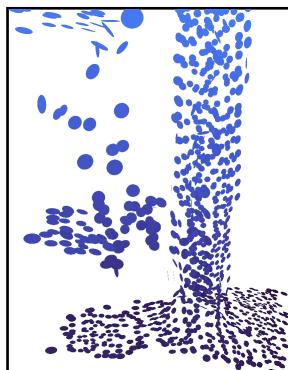


Abbildung 13: Ausschnitt von ei-
ner Punktwolke.
Die Punkte beim Stamm sind
kleiner als die umliegenden
Punkte und die Orientierung
ändert sich beim Übergang
vom Stamm zum Boden.

3. Triangulierung

3.1. Ziel

Das Ziel der Triangulierung ist eine Approximation der ursprünglichen Oberfläche von den gescannten Bäumen, welche weiterverarbeitet oder angezeigt werden kann. Die meisten Programme und Hardware sind auf das Anzeigen von Dreiecken spezialisiert und können diese effizienter als Punkte darstellen. Dafür wird die Triangulierung von einem Segment bestimmt.

3.2. Ball-Pivoting Algorithmus

3.2.1. Überblick

Beim Ball-Pivoting Algorithmus werden die Dreiecke der Oberfläche bestimmt, welche von einer Kugel mit Radius α (α -Kugel) erreicht werden können. Dabei berührt die Kugel die drei Eckpunkte vom Dreieck und alle anderen Punkte sind außerhalb der Kugel.

In Abbildung 14 ist ein Beispiel in 2D gegeben. Dabei werden die Linien gesucht, dass der zugehörige Kreis keine weiteren Punkte enthält.

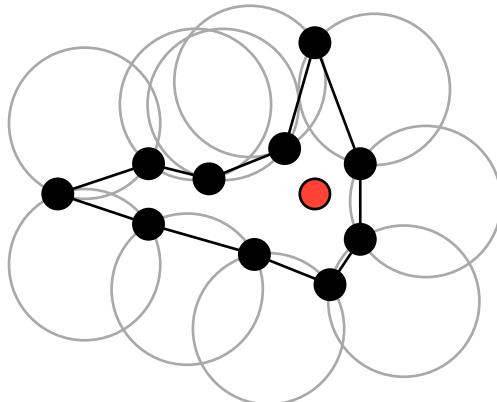


Abbildung 14: Ball-Pivoting Algorithmus in 2D. Für die äußeren Punkte in Schwarz wird eine Oberfläche gefunden. Für den inneren Punkt in Rot kann kein Nachbar gefunden werden, weil alle zugehörigen Kreise weitere Punkte enthalten würden.

Die gefundenen Dreiecke bilden eine Hülle, welche alle Punkte beinhaltet. Je kleiner α ist, desto genauer ist die Hülle um die Punkte und Details werden besser wiedergegeben. Dafür werden mehr Dreiecke benötigt und Lücken im Datensatz sind auch in der Hülle vorhanden.

3.2.2. α -Kugel für ein Dreieck

Für ein Dreieck (p_1, p_2, p_3) wird die Position der zugehörigen α -Kugel benötigt. Dafür wird zuerst das Zentrum c vom Umkreis vom Dreieck bestimmt. Von diesem sind alle Eckpunkte gleich weit entfernt. Ist der Abstand d_c vom Zentrum zu den Ecken größer als α , so gibt es keine zugehörige α -Kugel. Für Abstände kleiner gleich α ist das Zentrum der Kugel $d = \sqrt{\alpha^2 - d_c^2}$ vom Zentrum vom Umkreis entfernt. Der Vektor $o = (p_2 - p_1) \times (p_3 - p_1)$ ist orthogonal zum Dreieck, womit die Position vom Zentrum der α -Kugel mit $c + d \cdot \frac{o}{|o|}$ berechnet werden kann.

Durch die Berechnung von α ist die Reihenfolge der Punkte relevant. Vertauschen von zwei Punkten berechnet die α -Kugel auf der anderen Seite des Dreiecks.

3.2.3. Ablauf

3.2.4. KD-Baum berechnen

Zuerst wird ein KD-Baum für die Punkte im Segment berechnet. Der KD-Baum ermöglicht die effiziente Bestimmung von den nächsten Punkten für eine beliebige Position. Dabei werden nur die Punkte bestimmt, welche näher als ein Maximalabstand von der Position entfernt sind.

Mit dem KD-Baum kann auch überprüft werden, ob in einer α -Kugel keine weiteren Punkte liegen. Dafür wird der erste Punkt gesucht, der in der Kugel liegt. Wenn kein Punkt gefunden wird, ist die Kugel leer.

3.2.5. Startdreieck bestimmen

Als Anfang wird ein Dreieck mit zugehöriger α -Kugel benötigt, dass keine weiteren Punkte innerhalb der Kugel liegen. Dafür werden alle Punkte iteriert.

Für den momentanen Punkt werden die umliegenden Punkte mit einem Abstand von 2α oder weniger bestimmt. Für weiter entfernte Punkte gibt es keine α -Kugel, welche beide Punkte berühren würde.

Mit dem momentanen Punkt und alle möglichen Kombination von zwei Punkten aus den umliegenden Punkten wird ein Dreieck gebildet. Für das Dreieck werden nun die zwei möglichen α -Kugeln bestimmt, welche zum Dreieck gehören.

Wenn ein Dreieck mit zugehöriger α -Kugel gefunden wurde, welche keine weiteren Punkte enthält, kann dieses Dreieck als Startdreieck verwendet werden. Das Dreieck wird zur Triangulierung hinzugefügt und die drei zugehörigen Kanten bilden die momentanen äußeren Kanten, von denen aus die Triangulierung berechnet wird.

3.2.6. Triangulierten Bereich erweitern

Solange es noch eine äußere Kante (p_1, p_2) gibt, kann die Triangulierung erweitert werden. Für die Kante ist bereits ein Dreieck und die zugehörige α -Kugel mit Zentrum c bekannt. Die Kante dient als Pivot, um welches die α -Kugel gerollt wird. Der erste Punkt p , welcher von der Kugel berührt wird, bildet mit p_1 und p_2 ein neues Dreieck.

In Abbildung 15 ist ein Beispiel für eine Erweiterung in 2D gegeben. Im zweidimensionalen werden Kanten gesucht und Punkte werden als Pivot verwendet.

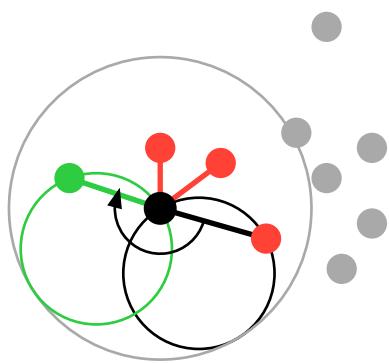


Abbildung 15: Erweiterung der gefundenen Oberfläche in 2D. Die vorherige Kante und der momentane Pivot-Punkt sind in Schwarz. Der α -Kreis rollt entlang der markierten Richtung. In Grün ist der erste Punkt und zugehörigen Kreis, welche berührt werden. Die weiteren Punkte in Rot sind Kandidaten, liegen aber weiter in der Rotation, weshalb die grüne Kante zur Oberfläche hinzugefügt wird.

Mögliche Kandidaten bestimmen

Um den ersten Punkt p zu finden, werden zuerst alle möglichen Punkte bestimmt, welche von der Kugel bei der kompletten Rotation berührt werden können. Dafür wird der Mittelpunkt $mp = \frac{p_1 + p_2}{2}$ der Kante und der Abstand $d = |p_1 - mp| = |p_2 - mp|$ berechnet. Der Abstand zwischen dem Zentrum der Kugel und den Endpunkten von der Kante ist immer α , dadurch ist der Abstand vom Zentrum zum Mittelpunkt $d_c = \sqrt{\alpha^2 - d^2}$. In Abbildung 16 ist die Konstruktion veranschaulicht.

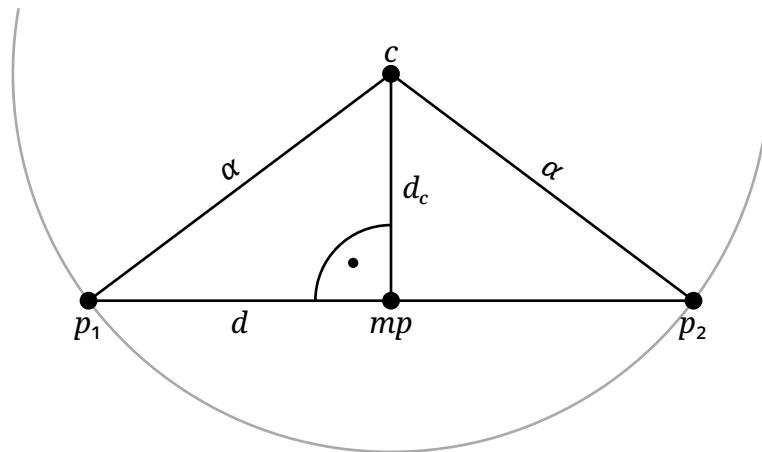


Abbildung 16: Konstruktion des Abstands der α -Kugel vom Mittelpunkt der Kante

Die möglichen Punkte sind vom Zentrum der Kugel c maximal α entfernt und c ist vom Mittelpunkt mp genau d_c weit entfernt. Deshalb werden mit dem KD-Baum die Punkte in der Kugel mit Zentrum mp und Radius $\alpha + d_c$ bestimmt.

Besten Kandidaten bestimmen

Für jeden Kandidaten p wird berechnet, wie weit die Kugel um die Kante gerollt werden muss, bis die Kugel den Kandidaten berührt. Dafür wird zuerst das Zentrum c_p der α -Kugel bestimmt, welche p_1 , p_2 und p berührt. Die Kugel wird dabei wie in Abbildung 17 bestimmt, dass die Kugel auf der korrekten Seite vom potenziellen Dreieck liegt. p kann so liegen, dass es keine zugehörige α -Kugel gibt, in diesem Fall wird p nicht weiter betrachtet. Für die restlichen Kandidaten wird der Winkel φ berechnet, wie weit um die Kante die Kugel gerollt wurde.

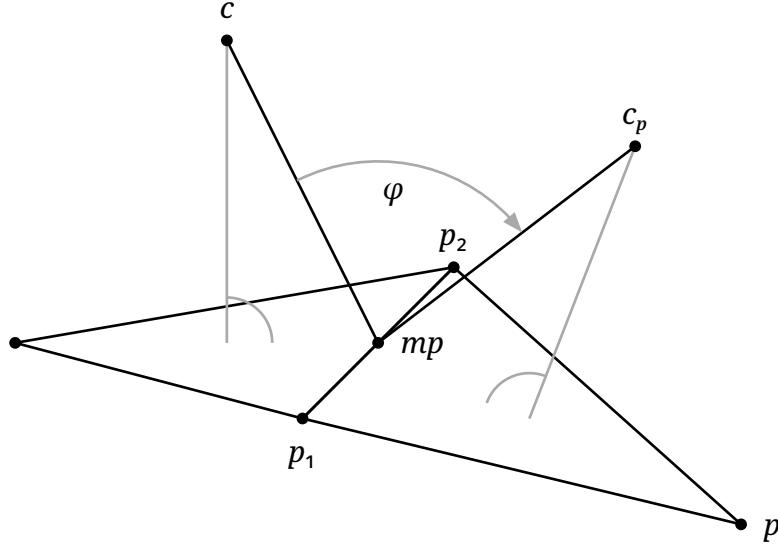


Abbildung 17: Berechnung von Zentrum der α -Kugel und zugehöriger Winkel für einen Kandidatenpunkt

Mit mp , c und cp wird der Winkel φ bestimmt. Dafür werden die Vektoren $a = c - mp$ und $b = cp - mp$ bestimmt und normalisiert. Der Kosinus von φ ist dabei das Skalarprodukt $s = a \cdot b$. Zusätzlich wird das Kreuzprodukt $k = a \times b$ bestimmt, um die Winkel mit gleichen Kosinus zu unterscheiden.

$$\varphi = \begin{cases} \arccos(s) & \text{falls } k \geq 0 \\ \pi - \arccos(s) & \text{falls } k < 0 \end{cases}$$

Von allen Kandidaten wird der Punkt p_3 ausgewählt, für den φ am kleinsten ist.

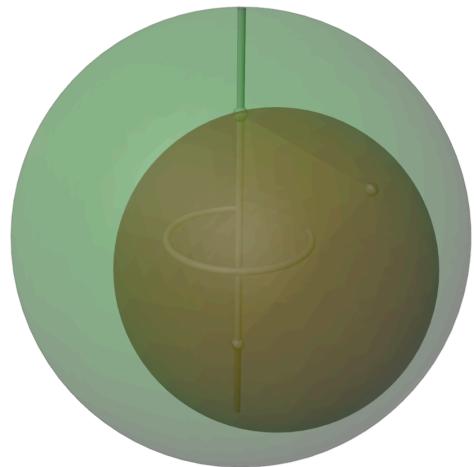
Es muss nicht kontrolliert werden, ob ein Punkt in der α -Kugel von (p_1, p_2, p_3) liegt, weil diese immer leer ist. Würde ein weiterer Punkt in der Kugel liegen, so würde der zugehörige Winkel φ von diesem Punkt kleiner sein, weil der Punkt beim Rollen um die Kante früher von der Kugel berührt wird. Weil p_3 aber zum kleinsten Winkel gehört, ist die zugehörige α -Kugel immer leer. Dies gilt aber nur, wenn die Kugel vor dem Rollen leer ist.

Triangulierung erweitern

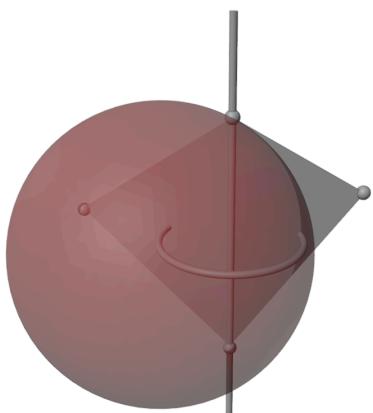
Das neu gefundene Dreieck mit den Eckpunkten (p_1, p_2, p_3) wird zur Triangulierung hinzugefügt. Die Kante (p_1, p_2) wird von den äußeren Kanten entfernt, dafür werden die neuen Kanten zwischen (p_1, p_3) und (p_3, p_2) hinzugefügt. Wenn eine der neuen Kante in den äußeren Kanten bereits vorhanden ist, wird diese nicht hinzugefügt, sondern von den äußeren Kanten entfernt, weil das zugehörige zweite Dreieck bereits gefunden wurde. Eine Veranschaulichung ist in Abbildung 18 gegeben.



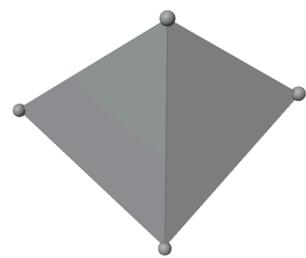
(a) Kante mit zugehörigem Dreieck, α -Kugel und Ring mit Radius d_c



(b) Kugel mit Radius $\alpha + d_c$, welche alle Kandidaten enthält



(c) Erster Punkt, welcher entlang der Rotation die Kugel berührt



(d) Triangulierung, mit dem neuen Dreieck hinzugefügt

Abbildung 18: Erweiterung der Triangulierung in 3D.

3.2.7. Komplettes Segment triangulieren

Solange es noch äußere Kanten gibt, kann von diesen aus die Triangulierung erweitert werden. Dabei muss beachtet werden, dass durch Ungenauigkeiten bei der Berechnung und Punkte mit gleicher Position eine Kante mehrfach gefunden werden kann. Um eine erneute Triangulierung von bereits triangulierten Bereichen zu verhindern, werden alle inneren Kanten gespeichert und neue Kanten nur zu den äußeren Kanten hinzugefügt, wenn diese noch nicht in den inneren Kanten vorhanden sind. Bei der Erweiterung wird die äußere Kante zu den inneren Kanten hinzugefügt.

Wenn es keine weiteren äußeren Kanten gibt, muss ein neues Startdreieck gefunden werden. Dabei werden nur die Punkte in Betracht gezogen, welche zu noch keinem Dreieck gehören. Wenn kein Startdreieck gefunden werden kann, ist das Segment vollständig trianguliert.

3.2.8. Vorauswahl

Vor der Triangulierung wird mit einem Mindestabstand die Menge der Punkte berechnet, welche betrachtet werden. Dafür wird eine Teilmenge der Punkte bestimmt, dass die Punkte paarweise mindestens den Mindestabstand voneinander entfernt sind.

Für die Berechnung wird ein Greedy-Algorithmus verwendet. Am Anfang werden alle Punkte zur Teilmenge hinzugefügt und danach werden alle Punkte in der Teilmenge iteriert. Für jeden Punkt werden mit dem KD-Baum die Punkte in der Nachbarschaft bestimmt, welche näher als der Mindestabstand zum momentanen Punkt liegen. Die Punkte werden aus der Teilmenge entfernt.

3.2.9. Auswahl von α

In Abbildung 19 wurde die Triangulation für die gleiche Punktwolke mit unterschiedlichen Werten für α berechnet. Mit einem größerem α wird das Ergebnis immer weiter vereinfacht. Bei einem kleinen Wert für α können Lücken in der Triangulierung entstehen, wenn die Punkte weiter als 2α voneinander entfernt sind.

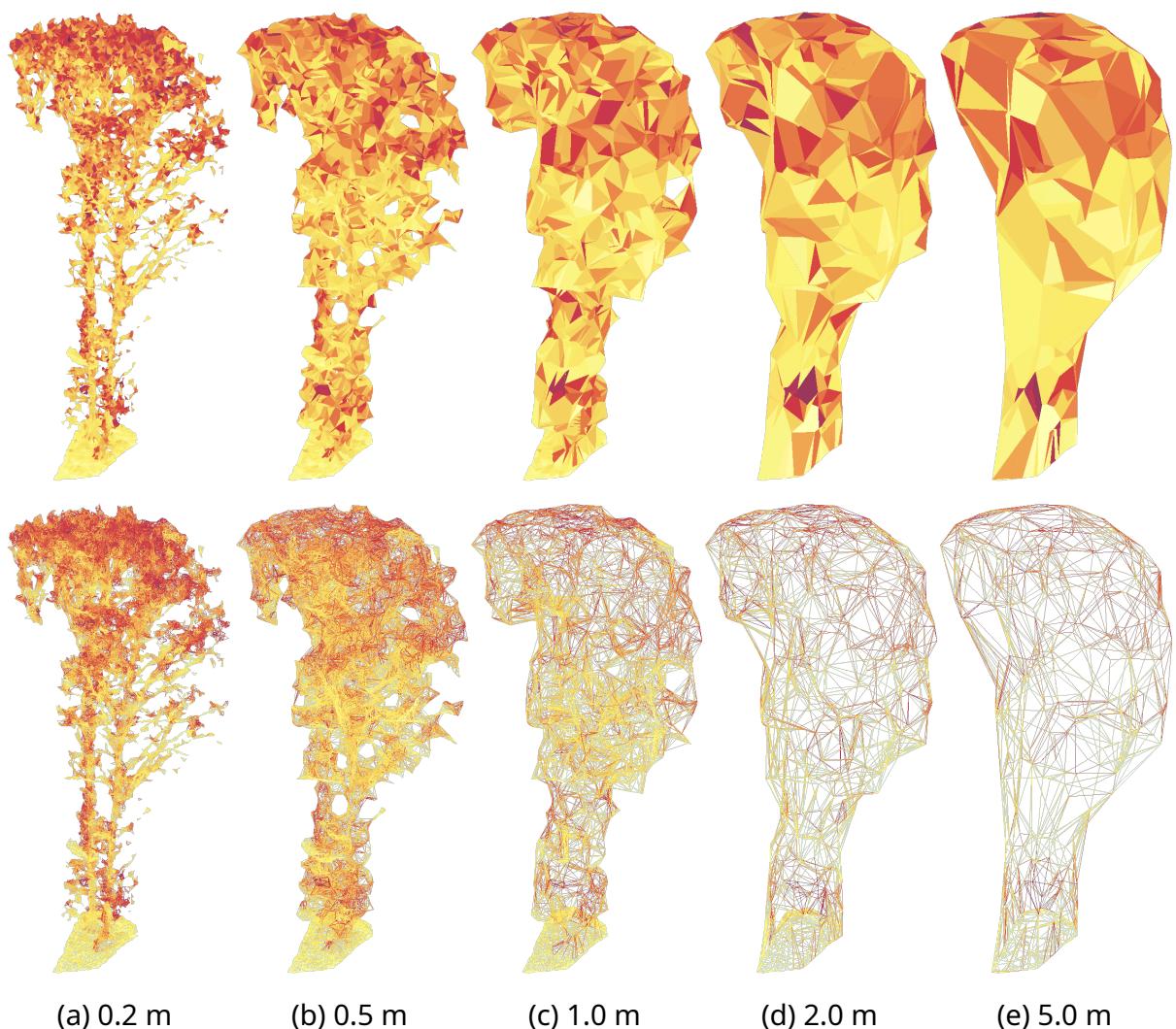


Abbildung 19: Triangulation für unterschiedliche α . Im oberen Bild sind die Dreiecke ausgefüllt und im unteren Bild umrandet.

Der Bereich für die Suche vom nächsten Kandidaten für die Erweiterung von der Triangulierung ist abhängig von α . Dadurch steigt der Berechnungsaufwand mit größerem α , weil mehr Kandidaten betrachtet werden müssen.

Im Idealfall wird α so klein gewählt, dass keine gewünschten Details verloren gehen und so groß, dass keine Lücken in der Triangulierung entstehen.

4. Visualisierung

4.1. Punkte

Grafikpipelines haben mehrere primitive Formen, welche gerendert werden können. Die verfügbaren primitiven Formen sind meistens Punkte, Linien und Dreiecke, wobei Dreiecke immer verfügbar sind. Für das Anzeigen von komplizierter Modelle werden mehrere primitiven Formen zusammengesetzt.

Der primitive Punkt hat dabei keine Größe, sondern wird mit genau einem Pixel dargestellt. Um einen Punkt mit einer Größe anzeigen, werden Dreiecke als primitive Form verwendet. Bei einem Dreieck kann beliebige Eckpunkte haben und die Grafikpipeline färbt alle Pixel ein, welche zwischen den Eckpunkten liegen.

Um einen Kreis zu rendern, kann ein beliebiges Polygon gerendert werden, solange der gewünschte Kreis vollständig enthalten ist. Die Pixel, welche außerhalb vom Kreis liegen, werden beim Rendern verworfen, wodurch nur der Kreis übrig bleibt. Je mehr Ecken das Polygon hat, desto kleiner ist der Bereich vom Polygon, der nicht zum Kreis gehört. Jede Ecke und der benötigte Bereich erhöhen den benötigten Arbeitsaufwand.

4.1.1. Mögliche Polygone

Zuerst wird ein Kreis mit Position $(0, 0)$ und Radius 1 benötigt. Mithilfe der Position vom Punkt und der Kamera wird der Kreis transformiert, dass die korrekten Pixel eingefärbt werden.

Das kleinste passende Dreieck ist ein gleichseitiges Dreieck. In Abbildung 20 ist die Konstruktor für die Seitenlänge gegeben. Ein mögliches Dreieck hat die Eckpunkte $(-\tan(60^\circ), -1)$, $(\tan(60^\circ), -1)$ und $(0, 2)$. Für das Dreieck werden dadurch drei Ecken und eine Fläche von $\frac{w \cdot h}{2} = \frac{\tan(60^\circ) \cdot 2 \cdot 3}{2} = \tan(60^\circ) \cdot 3 \approx 5.2$ benötigt.

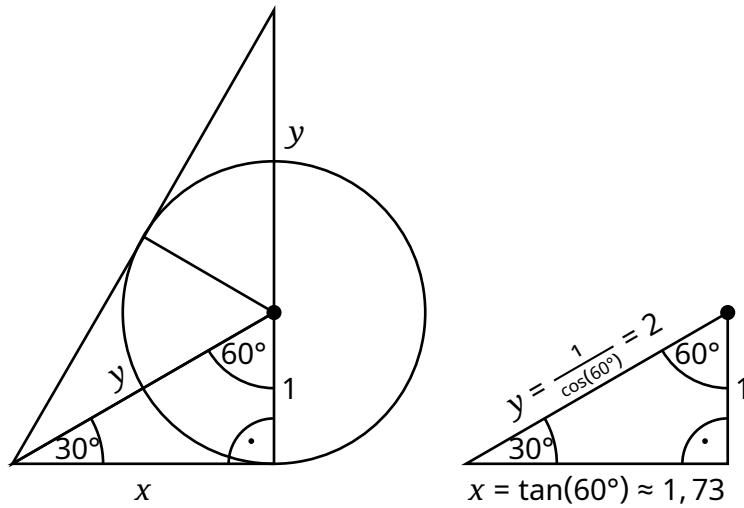


Abbildung 20: Seitenlänge für das kleinste gleichseitige Dreieck, welches den Einheitskreis enthält.

Das kleinste mögliche Viereck ist das Quadrat mit Seitenlänge 2. In Abbildung 21 ist die Konstruktion gegeben. Um diesen anzuzeigen, werden zwei Dreiecke benötigt. Für die beiden Dreiecke werden dadurch sechs Ecken und eine Fläche von $a^2 = 2^2 = 4$ benötigt.

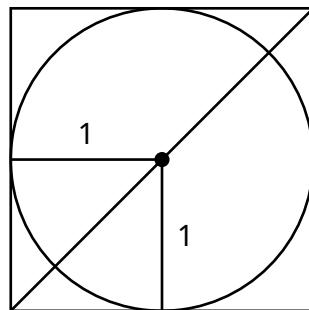


Abbildung 21: Seitenlänge für das kleinste Quadrat, welches den Einheitskreis enthält.

In Abbildung 22 ist ein Vergleich für eine Punktfolge gerendert mit unterschiedlichen Polygonen. Für Polygone mit mehr Ecken, wird der benötigte Bereich kleiner, es werden aber auch mehr Ecken benötigt.

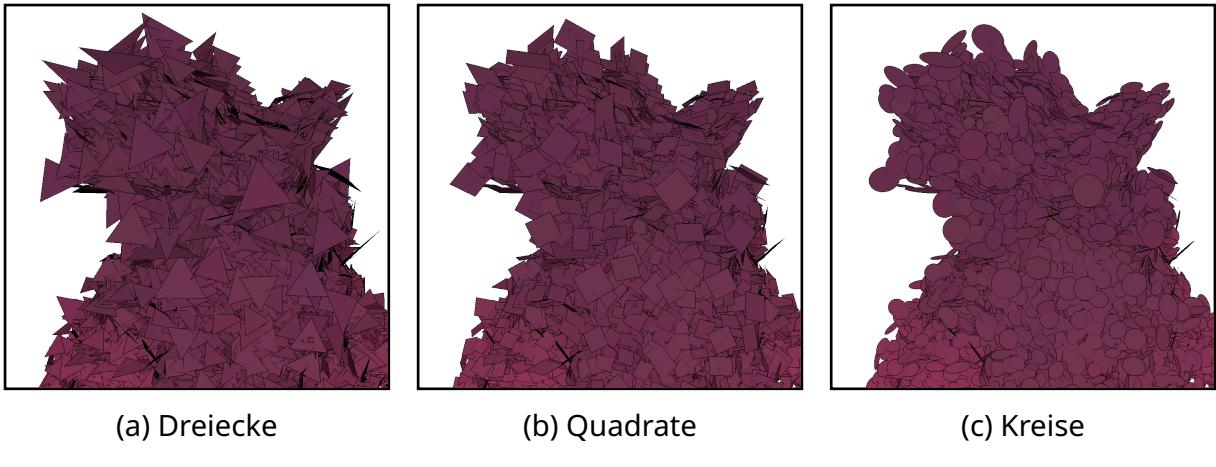


Abbildung 22: Die gleiche Punktwolke mit unterschiedlichen Polygonen und Kreisen für die Punkte.

4.1.2. Anzeigen im dreidimensionalen Raum

Für jeden Punkt wird mit der Position p , Normalen n und Größe s die Position der Eckpunkte der Dreiecke im dreidimensionalen Raum bestimmt. Dafür werden zwei Vektoren bestimmt, welche paarweise zueinander und zur Normalen orthogonal sind.

Für den ersten Vektor a wird mit der Normalen $n = (n_x, n_y, n_z)$ das Kreuzprodukt $a = (n_x, n_y, n_z) \times (n_y, n_z, -n_x)$ bestimmt. Weil $|n| > 0$ ist, sind $(n_y, n_z, -n_x)$ und n unterschiedlich. a muss noch für die weiteren Berechnungen normalisiert werden. Für den zweiten Vektor b wird das Kreuzprodukt $b = n \times a$ bestimmt. Weil das Kreuzprodukt zweier Vektoren orthogonal zu beiden Vektoren ist, sind n , a und b paarweise orthogonal. Ein Beispiel ist in Abbildung 23 gegeben.

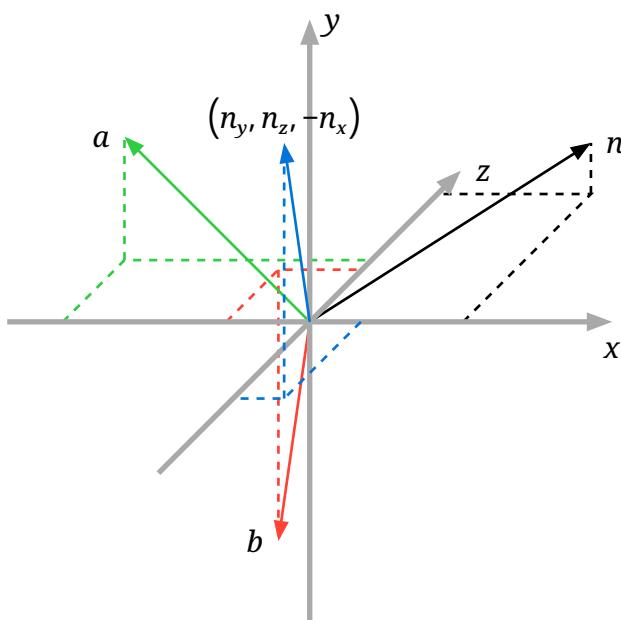


Abbildung 23: Beispiel für die Berechnung von a und b paarweise orthogonal zu n .

Die Vektoren a und b spannen eine Ebene auf, welche orthogonal zu n ist. Für den Eckpunkt i vom Dreieck, mit den Koordinaten (x_i, y_i) , wird die Position $p_i = p + a * x_i * s + b * y_i * s$ berechnet werden. In Abbildung 24 ist die Berechnung dargestellt.

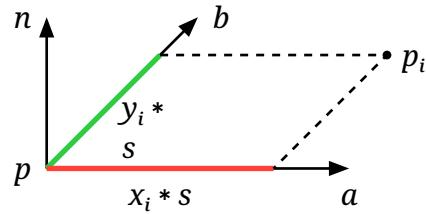


Abbildung 24: Berechnung von einem Eckpunkt.

4.2. Detailstufen

Je nach Scanner und Größe des abgetasteten Gebietes kann die Punktwolke unterschiedlich viele Punkte beinhalten. Durch Hardwarelimitierungen ist es nicht immer möglich, alle Punkte gleichzeitig anzuzeigen, während eine interaktive Wiedergabe gewährleistet ist.

Besonders für weit von der Kamera entfernte Punkte ist es nicht notwendig, alle Punkte genau anzuzeigen. Deshalb wird für weit entfernte Punkte eine vereinfachte Version berechnet und anstelle der originalen Punkte verwendet. Diese besteht aus weniger Punkten und benötigt dadurch weniger Ressourcen.

Für die gesamte Punktwolke wird ein Octree mit den Punkten erstellt. Am Anfang besteht der Octree aus einem Leaf-Knoten und die Punkte zum Octree hinzugefügt. Dafür wird der Leaf-Knoten bestimmt, der zur Position vom Punkt gehört. Enthält der Leaf-Knoten weniger Punkte als die festgelegte Maximalanzahl, so wird der Punkt zum Knoten hinzugefügt. Wenn der Leaf-Knoten bereits voll ist, so wird dieser unterteilt. Der Leaf-Knoten wird in acht Kinderknoten unterteilt und die Punkte vom Leaf-Knoten werden auf die Kinderknoten verteilt, wodurch der Leaf-Knoten zum Branch-Knoten wird. Für die Unterteilung wird der Knoten entlang der x-, y- und z-Achse in der Mitte geteilt.

Alle Punkte gehören nach der Unterteilung zu einem Leaf-Knoten im Octree. Für jeden Branch-Knoten wird dann eine Punktwolke berechnet, welche als Vereinfachung der Punkte der zugehörigen Kinderknoten verwendet werden kann. In Abbildung 25 sind die unterschiedlichen Stufen vom Octree mit zugehörigen Detailstufen visualisiert.

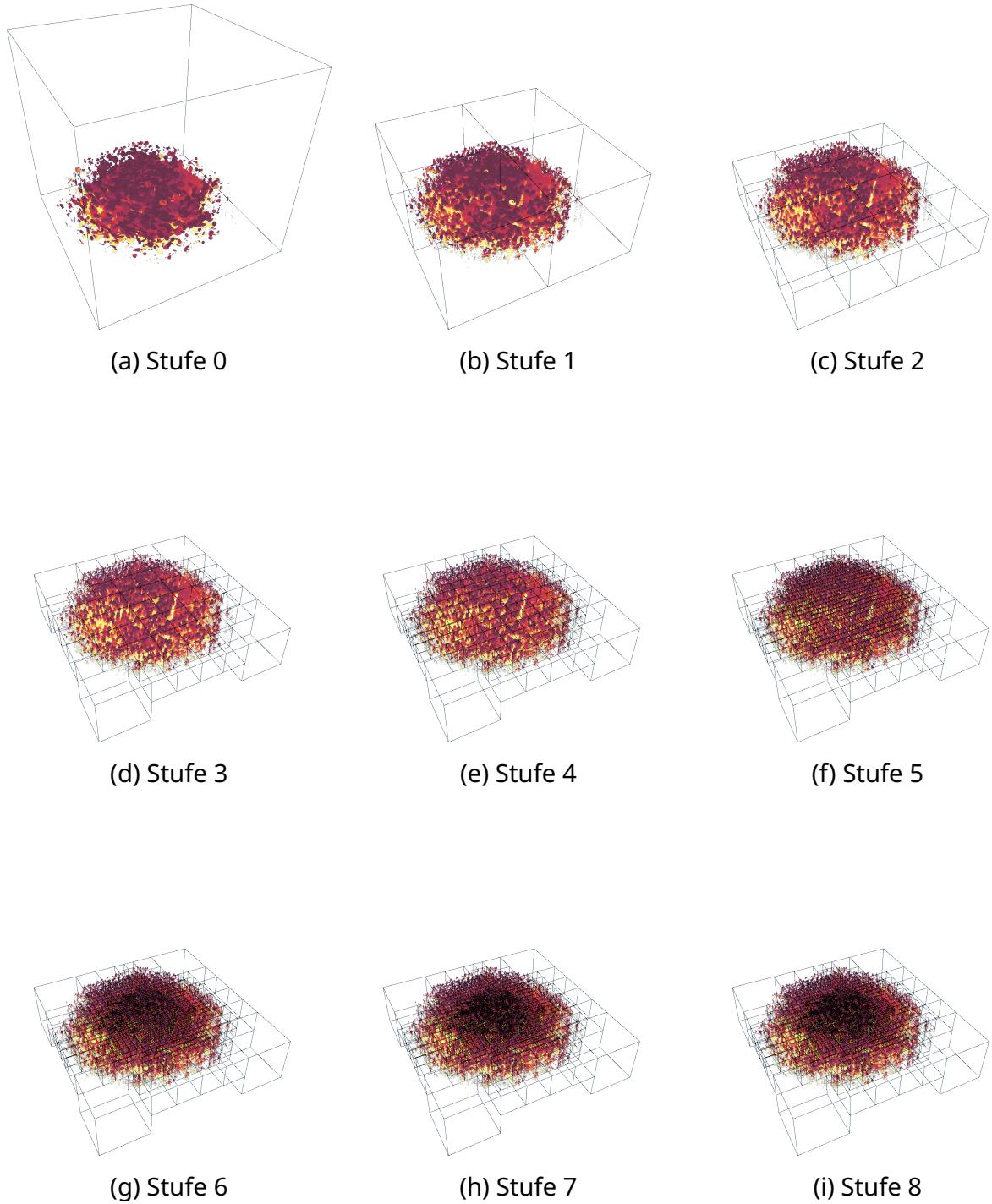


Abbildung 25: Unterschiedliche Detailstufen. Jeder Würfel enthält bis zu 32768 Punkte. In der höchsten Stufe werden alle Punkte im Datensatz angezeigt.

4.2.1. Berechnung der Detailstufen

Die Detailstufen werden wie bei „Fast Out-of-Core Octree Generation for Massive Point Clouds“ [8] von den untersten Branch-Knoten bis zum Root-Knoten berechnet. Dabei wird mit den Detailstufen der Kinderknoten die Detailstufe für den momentanen Knoten berechnet.

Dadurch haben zwar Berechnungen der größeren Detailstufen für Knoten näher an der Wurzel nur Zugriff auf bereits vereinfachte Daten, aber die Anzahl der Punkte, mit denen die Detailstufe berechnet wird, ist viel kleiner. Solange die Detailstufen eine gute Vereinfachung der ursprünglichen Punkte sind, kann so der Berechnungsaufwand stark verringert werden.

Für die Berechnung einer Detailstufe wird der Voxel, welcher zu dem Knoten gehört, in eine feste Anzahl von gleich großen Teilvoxeln unterteilt. Für jeden Teilvoxel werden zuerst alle Punkt aus den Kinderknoten bestimmt, welche im Teilvoxel liegen. Liegt kein Punkt im Teilvoxel, so wird dieser übersprungen. Aus den Punkten im Teilvoxel wird ein repräsentativer Punkt bestimmt. Dafür werden Position, Normale und Größe gemittelt und die Eigenschaften von einem der Punkte übernommen. Die Detailstufe besteht aus allen repräsentativen Punkten für die Teilvoxel, welche nicht leer waren.

Bei der nächst größeren Detailstufe ist der Voxel vom Branch-Knoten doppelt so groß. Durch die feste Anzahl der Teilvoxel verdoppelt sich auch die Größe der Teilvoxel, wodurch die Punkte weiter vereinfacht werden.

4.3. Eye-Dome Lighting

Um die Punktwolke anzuzeigen, werden die Punkte aus dem dreidimensionalen Raum auf den zweidimensionalen Monitor projiziert. Dabei gehen die Tiefeninformationen verloren. Mit der Rendertechnik Eye-Dome Lighting werden die Kanten von Punkten hervorgehoben, bei denen die Tiefe sich stark ändert.

Beim Rendern von 3D-Szenen wird für jedes Pixel die momentane Tiefe vom Polygon an dieser Stelle gespeichert. Das wird benötigt, dass bei überlappenden Polygone das nächste Polygon an der Kamera angezeigt wird. Nachdem die Szene gerendert ist, wird mit den Tiefeninformationen für jedes Pixel der Unterschied zu den umliegenden Pixeln bestimmt. Ein Beispiel für die Tiefeninformationen ist in Abbildung 26 gegeben.

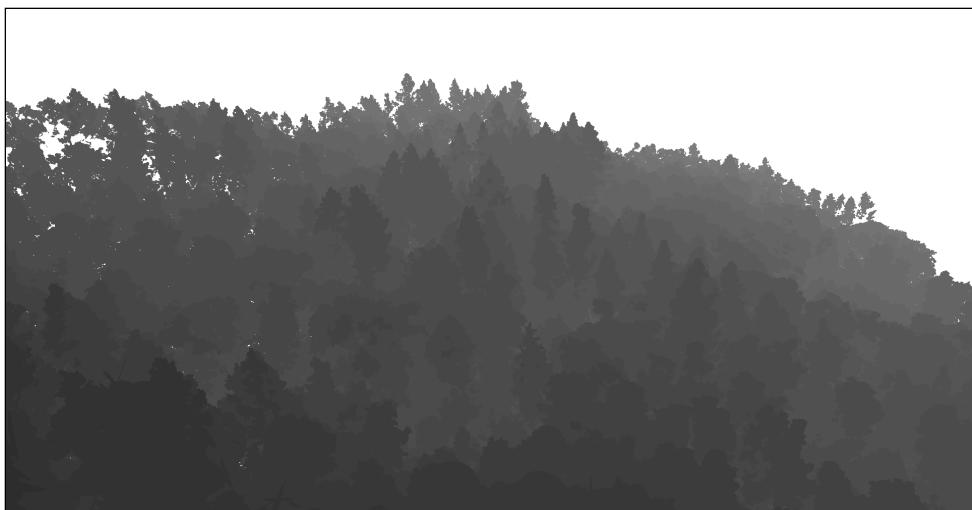
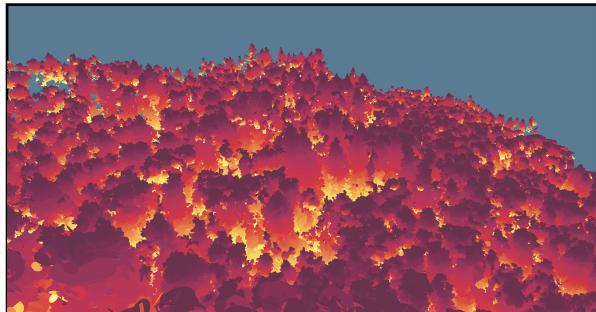
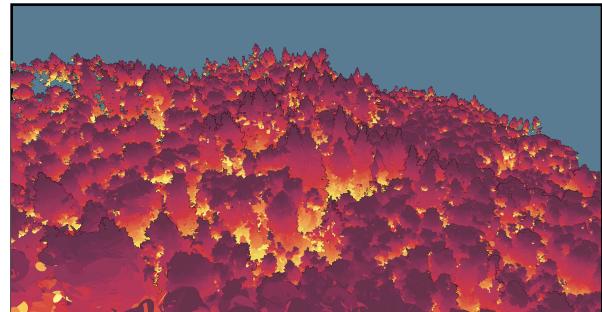


Abbildung 26: Tiefeninformationen nach dem Rendern der Szene. Je heller eine Position ist, desto weiter ist das Polygon zugehörig zur Koordinate von der Kamera entfernt.

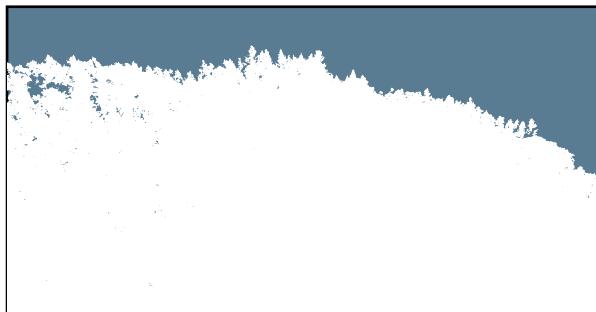
Der Effekt entsteht dadurch, dass für jedes Pixel der maximale Unterschied in der Tiefe zu den umliegenden Pixeln bestimmt wird. Je größer der Unterschied, desto mehr wird das zugehörige Pixel verdunkelt. Eine Veranschaulichung ist in Abbildung 27 gegeben.



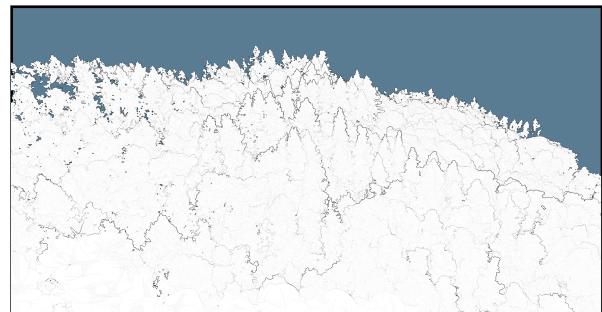
(a) Ohne Eye-Dome Lighting



(b) Mit Eye-Dome Lighting



(c) Einfarbig ohne Eye-Dome Lighting



(d) Einfarbig mit Eye-Dome Lighting

Abbildung 27: Waldstück mit und ohne Eye-Dome Lighting. Die Punkte sind zusätzlich in Weiß angezeigt, um den Effekt hervorzuheben.

IV. Implementierung

1. Technik

Das Softwareprojekt ist auf [GitHub](#)¹ verfügbar. Als Programmiersprache wird Rust und als Grafikkartenschnittstelle WebGPU verwendet. Rust ist eine performante Programmiersprache mit einfacher Integration für WebGPU. WebGPU bildet eine Abstraktionsebene über der nativen Grafikkartenschnittstelle, dadurch ist die Implementierung unabhängig vom Betriebssystem. Alle verwendeten Bibliotheken sind in Tabelle 1 gelistet.

Name	Version	Funktionalität
nalgebra	0.32.4	Lineare Algebra
pollster	0.3	Auf asynchrone Berechnungen warten
rfd	0.14	Dialogfenster zum Öffnen und Speichern von Dateien
crossbeam	0.8	Synchronisierung zwischen Threads
log	0.4	Logs erzeugen
env_logger	0.11	Wiedergabe von Logs
image	0.24	Laden und Speichern von Bildern
wgpu	0.19	WebGPU Implementierung
winit	0.29	Fenstermanagement
bytemuck	1.14	Konversation von Daten zu Bytes
serde	1.0	Serialisierung von Datentypen
serde_json	1.0	Serialisierung als JSON
rand	0.8	Generierung von Zufallszahlen
num_cpus	1.15	Prozessoranzahl bestimmen
laz	0.8	Dekomprimieren von LASzip Dateien
thiserror	1.0	Fehlermanagement
tempfile	3.8.1	Temporäre Dateien erstellen
rayon	1.8.0	Multithreading
termsize	0.1	Größe vom Terminal bestimmen
egui	0.26	Benutzerinterface
egui-winit	0.26	Systemereignisse zum Interface weiterleiten
egui-wgpu	0.26	Interface rendern
clap	4.4	Kommandozeilenargumente verarbeiten
voronator	0.2.1	Voronoi-Diagramm bestimmen
cfg-if	1.0.0	Konditionales Kompilieren von Quelltext
static_assertions	1.1.0	Systemeigenschaften überprüfen
colored	2.1.0	Farbiger Text im Terminal

Tabelle 1: Benutzte Bibliotheken

¹<https://github.com/antonWetzel/treee>

Als Datensätze werden Dateien im LASzip-Format verwendet. Dieses Format wird häufig für Punkt wolken verwendet. Weitere Formate können einfach eingebunden werden, so lange eine Rust-Bibliothek existiert, welche das Format einlesen kann.

2. Benutzung

2.1. Installation

Für den Import und die Visualisierung wird das kompilierte Programm benötigt. Dieses kann mit dem Quelltext selber kompiliert werden oder bereits kompilierte Versionen können vom [GitHub-Release²](#) heruntergeladen werden. Die Schritte zum selber kompilieren sind im [Readme³](#) verfügbar.

2.2. Ausführen

In Tabelle 2 sind die Befehle gelistet, um den Importer und die Visualisierung zu starten. Wenn das Programm ohne Argumente oder direkt ohne Terminal gestartet wird, kann die gewünschte Funktion interaktiv ausgewählt werden. Für den Import können weitere Optionen angegeben werden, um den Ablauf an den Datensatz anzupassen.

Befehl	Funktion
treeee	Interaktive Umgebung starten
treeee importer	Importer starten
treeee help importer	Verfügbare Optionen für den Importer anzeigen
treeee viewer	Visualisierung starten

Tabelle 2: Mögliche Befehle für das Programm.

2.3. Import

Für den Import wird der Datensatz und der Ordner zum Speichern der Ergebnisse benötigt. Beide können über die Befehlszeile angegeben werden oder über ein Dialogfenster ausgewählt werden. Alle weiteren Optionen sind in Tabelle 3 gelistet. Am Ende vom Import wird im Ordner für die Ergebnisse die project.json Datei und zugehörige Daten abgespeichert, welche von der Visualisierung geöffnet werden können.

²<https://github.com/antonWetzel/treeee/releases>

³<https://github.com/antonWetzel/treeee?tab=readme-ov-file#treeee>

Flag	Standardwert	Funktion
--max-threads	unbegrenzt	Maximale Anzahl an parallelen Threads
--min-segment-size	100	Mindestanzahl von Punkten für ein Segment
--segmenting-slice-width	1.0	Breite der horizontalen Scheiben für die Segmentierung in Meter
--segmenting-max-distance	1.0	Mindestabstand zwischen Bereichen in Meter
--neighbors-count	31	Maximale Anzahl der Punkte in der Nachbarschaft von einem Punkt
--neighbors-max-distance	1.0	Maximale Distanz vom Punkt zu den Punkten in der Nachbarschaft
--lod-size-scale	0.95	Skalierungsfaktor für die Fläche der kombinierten Punkte

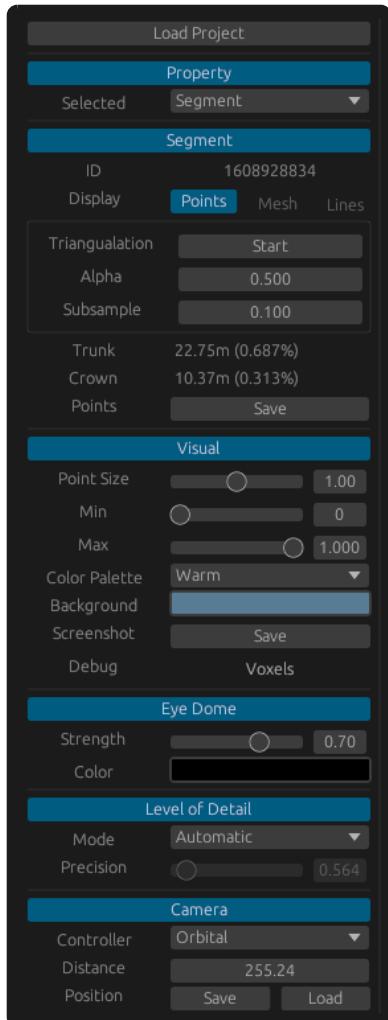
Tabelle 3: Mögliche Optionen für den Import.

2.4. Visualisierung

Um eine Punktfolke zu öffnen, wird die `project.json` Datei eingelesen. In der Datei ist die Struktur vom Octree und Informationen über die Segmente enthalten. Die Punktdaten sind in separaten Dateien gespeichert und werden noch nicht geladen.

Je nach Position der Kamera werden die benötigten Punkte geladen, welche momentan sichtbar sind. Dadurch können auch Punktfolken angezeigt werden, die mehr Punkte enthalten als gleichzeitig interaktiv anzeigbar. Auch wenn ein einzelnes Segment angezeigt wird, ist nur das Segment geladen, welches ausgewählt wurde.

Mit dem Benutzerinterface kann die Visualisierung angepasst werden und Informationen werden angezeigt. Die Optionen und einsehbaren Informationen sind in Abbildung 28 erklärt.



- **Load Project**
 - Die geladene Punktwolke ändern
- **Property**
 - Die angezeigte Eigenschaft ändern
- **Segment**
 - Triangulation starten
 - Punkte, Linien oder Dreiecke anzeigen
 - Informationen über das ausgewählte Segment
 - Segment speichern
- **Visual**
 - Punktegröße ändern
 - Punkte basierend auf der ausgewählten Eigenschaft filtern
 - Farbpalette und Hintergrundfarbe ändern
 - Screenshot speichern
 - Knoten für die Detailstufen anzeigen
- **Eye Dome**
 - Stärke und Farbe vom Eye-Dome Lighting ändern
- **Level of Detail**
 - Auswahl und Qualität der Detailstufen anpassen
- **Camera**
 - Steuerung der Kamera ändern
 - Kameraposition speichern oder wiederherstellen

Abbildung 28: Benutzeroberfläche mit den verfügbaren Optionen und Informationen.

3. Struktur vom Quelltext

Das Softwareprojekt ist in mehrere Module unterteilt, um den Quelltext zu strukturieren. In Tabelle 4 und Abbildung 29 sind die Module mit zugehöriger Funktionalität und Abhängigkeiten gelistet. Die wichtigsten Module sind `importer` und `viewer`, welche den Import und die Visualisierung beinhalten. Das Modul `treee` vereint beide zu einem Programm.

Name	Funktionalität
input	Maus- und Tastatureingaben verarbeiten
data-file	Daten zusammengefasst in einer Datei speichern
project	Format für eine Punktfolge und zugehörige Daten
k-nearest	Nachbarschaftssuche mit KD-Bäumen
render	Rendern von Punktfolgen, Linien und Dreiecken mit wgpu
viewer	Visualisierung von Punktfolgen
triangulation	Triangulation von Punktfolgen
importer	Import von Punktfolgen
treee	Gemeinsames Interface für importer und viewer

Tabelle 4: Module vom Projekt mit zugehöriger Funktionalität.

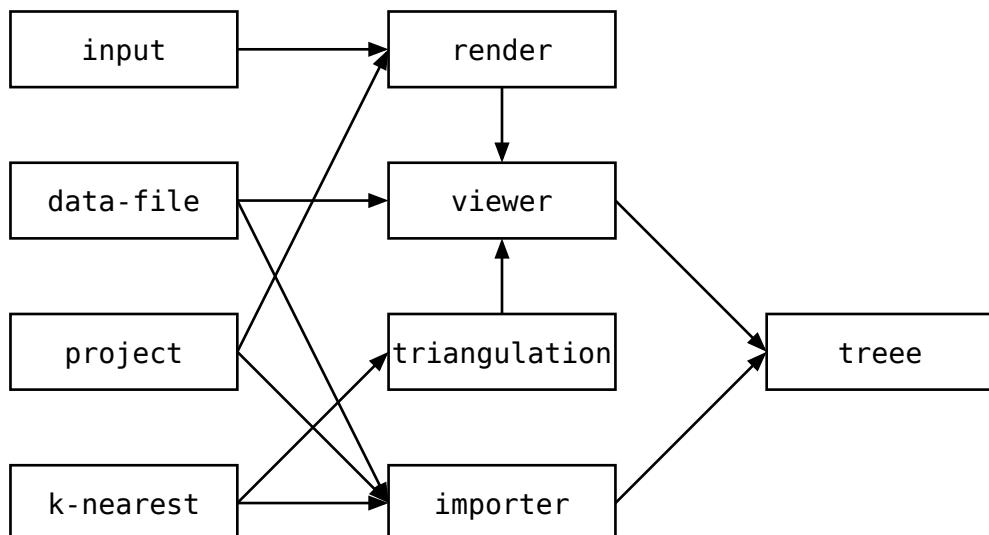


Abbildung 29: Abhängigkeiten der Module untereinander.

4. Format für eine Punktfolge

Die Struktur von einer Punktfolge ist in der `project.json` Datei gespeichert. Dazu gehören die verfügbaren Eigenschaften und der Octree. Alle benötigten Daten für `project.json` werden in [project/src/lib.rs](#)⁴ definiert.

4.1. Daten

In separaten Dateien werden die Daten für die Punkte oder Eigenschaften gespeichert. Das verwendete Dateiformat ermöglicht es, die Dateien inkrementell zu erstellen. Am Anfang wird nur benötigt, wie viele Einträge die Datei speichern kann. Danach können die Einträge in beliebiger Reihenfolge abgespeichert werden.

⁴<https://github.com/antonWetzel/treee/blob/main/project/src/lib.rs>

Die Struktur ist in Abbildung 30 gegeben. Am Anfang der Datei wird für jeden Eintrag die Startposition s_i und die Länge l_i vom zugehörigen Datensegment d_i gespeichert. Danach folgen die Datensegmente in beliebiger Reihenfolge π .

Informationen							Daten			
s_0	l_0	s_1	l_1	...	s_{n-1}	l_{n-1}	$d_{\pi(0)}$	$d_{\pi(1)}$...	$d_{\pi(n-1)}$

Abbildung 30: Struktur einer Datei zum Speichern von Daten.

Um den Eintrag i mit den Daten d zur Datei hinzufügen, wird zuerst s_i auf das momentane Ende der Datei und l_i auf die Länge von d gesetzt. Danach wird d am Ende der Datei hinzugefügt. Um die Daten für den Eintrag i zu lesen, wird zuerst s_i und l_i ausgelesen und danach der zugehörige Bereich geladen.

5. Import

Um einen Datensatz zu analysieren, muss dieser zuerst importiert werden, bevor er von der Visualisierung angezeigt werden kann. Der Import wird in mehreren getrennten Phasen durchgeführt. Dabei wird der Berechnungsaufwand für eine Phase so weit wie möglich parallelisiert. Die Phasen sind:

1. Daten laden
2. Segmente bestimmen
3. Segmente analysieren und den Octree erstellen
4. Detailstufen bestimmten und Octree speichern

Der zugehörige Datenfluss ist in Abbildung 31 zu sehen. Nach der ersten Phase sind die Punktedaten bekannt und nach der zweiten Phase auf die Segmente aufgeteilt. In der dritten Phase werden dann die Segmente verarbeiten und der Octree aufgebaut. Nach der vierten Phase ist auch der Octree vollständig und die Punktfolke wird abgespeichert.

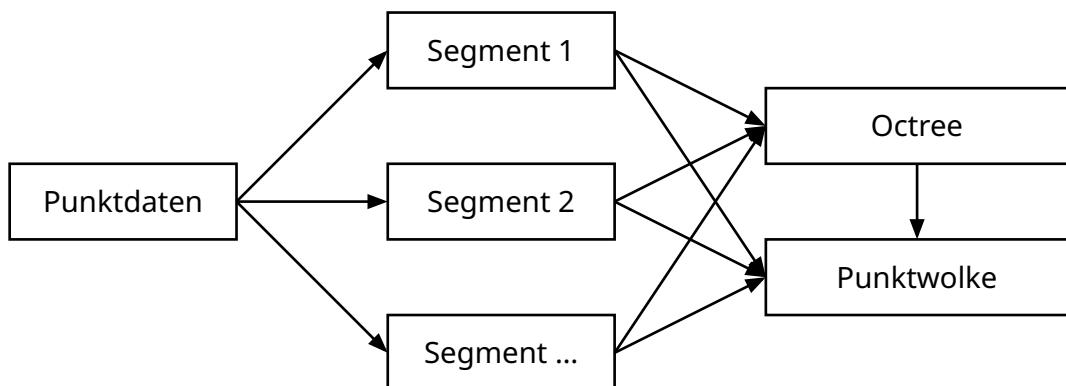
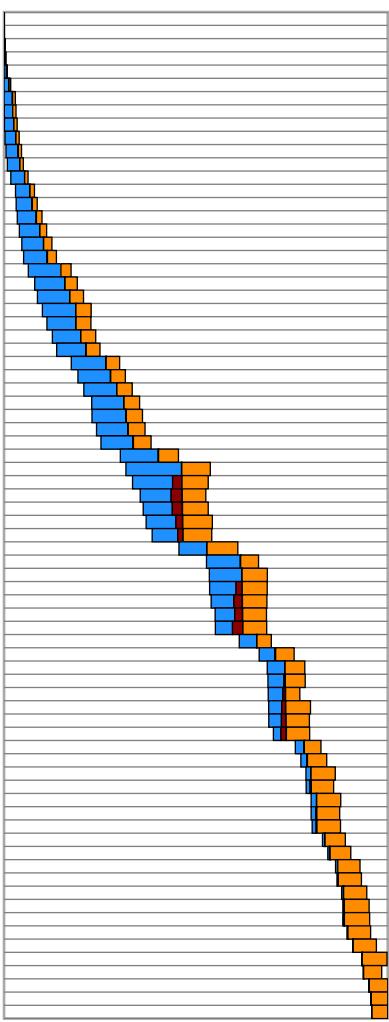


Abbildung 31: Datenfluss beim Import.

5.1. Parallelisierung

Die Punktdaten werden in LASzip Dateien zu Blöcken zusammengefasst. Jeder Block wird separat komprimiert, wodurch mehrere Blöcke auch parallel dekomprimiert werden können. Ein weiterer Thread sammelt die dekomprimierten Blöcke für die Segmentierung.

Für die Segmentierung wird über die einzelnen horizontalen Scheiben parallelisiert. Der genaue Ablauf ist in Abbildung 32 erklärt. Die Segmente werden wieder von einem weiteren Thread gesammelt.



Bei der Segmentierung werden die Punkte von oben nach unten in Scheiben verarbeitet. Jede Scheibe wird in den folgenden Stufen verarbeitet.

1. Zusammenhängenden Bereiche von den Punkten bestimmen.
2. Mit den Bereichen und den Koordinaten der vorherigen Scheibe die Koordinaten für die momentane Scheibe berechnen.
3. Punkte auf die Koordinaten verteilen.

Dabei wird für die zweite Stufe die Koordinaten aus der vorherigen Scheibe benötigt.

In der Grafik ist der Arbeitsaufwand abgebildet. Von oben nach unten sind die Scheiben und von links nach rechts die Zeit abgebildet. Die erste Stufe ist in Blau, das Warten auf die vorherige Scheibe in Rot und die dritte Stufe in Orange. Der Berechnungsaufwand der zweiten Stufe ist sehr kurz, wodurch dieser nicht in der Grafik sichtbar ist.

Die Berechnung wurde mit sieben Threads durchgeführt, wodurch bis zu sieben Scheiben in parallel verarbeitet werden können. Durch die Datenabhängigkeit kann aber die zweite Stufe erst verarbeitet werden, wenn von der vorherigen Scheibe die zweite Stufe beendet ist. Wenn die erste Stufe länger dauert, müssen deshalb andere Threads warten.

Abbildung 32: Parallelisierung der Segmentierung.

Die Analyse der Segmente und die Berechnung der Detailstufen sind trivial parallelisierbar. Die Analyse der Segmente wird für mehrere Segmente parallel durchgeführt, weil keine Abhängigkeiten zwischen den Daten existieren. Bei den Detailstufen können bei einem Knoten die Kinderknoten parallel verarbeitet werden.

6. Punkte

Die benötigten Daten für einen Punkt sind das Polygon als Basis, Position, Normale, Größe und ausgewählte Eigenschaft. Das Polygon ist gleich für alle Punkte und muss deshalb nur einmal zur Grafikkarte übertragen werden und wird für alle Punkte wiederverwendet.

Für die Grafikpipeline wird das Polygon in Dreiecke zerlegt. In Abbildung 33 sind die getesteten Varianten gegeben. Die Dreiecke werden dann mit der Kamera projiziert und es werden alle Pixel bestimmt, welche in den Dreiecken liegen. Für jedes Pixel kann entschieden werden, ob dieser im Ergebnis gespeichert wird. Dafür wird bei den Eckpunkten die Koordinaten ohne die Transformation der Kamera abgespeichert, dass diese später verfügbar sind. Für jedes Pixel wird von der Pipeline die interpolierten Koordinaten berechnet. Nur wenn der Betrag der interpolierten Koordinaten kleiner als eins ist, wird der Pixel im Ergebnis abgespeichert.

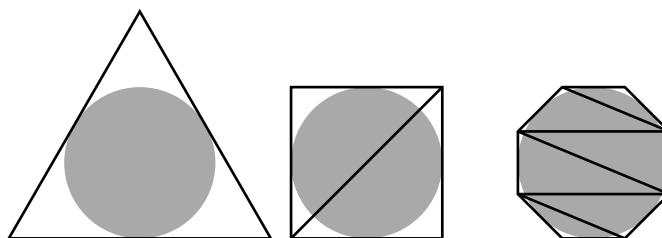


Abbildung 33: Zerlegung von unterschiedlichen Polygonen in Dreiecke.

In Abbildung 34 sind die Zeiten für das Rendern von unterschiedlichen Polygonen als Basis gegeben. Die beste Option ist das Dreieck als Polygon. Für die Zerlegung vom Polygon mit n Ecken in Dreiecke werden $n - 2$ Dreiecke und somit $3n - 6$ Ecken benötigt. Der benötigte Aufwand entsteht größtenteils durch die Ecken, wodurch das Quadrat circa doppelt und das Achteck sechsmal so lange zum Rendern benötigen.

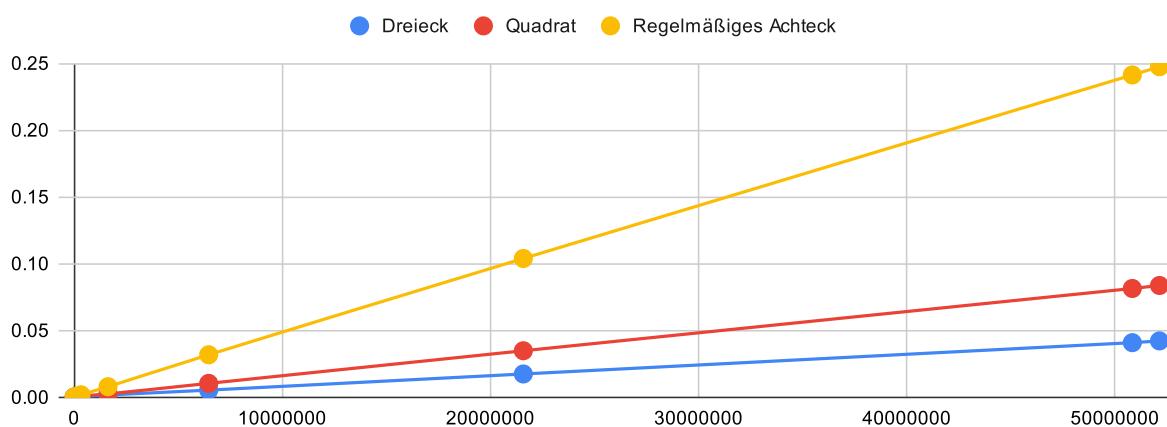


Abbildung 34: Renderzeit bei unterschiedlichen Polygonen als Basis in Sekunden abhängig von der Anzahl der Punkte.

Die ausgewählte Eigenschaft wird durch Einfärbung der Punkte angezeigt. Dabei kann die ausgewählte Eigenschaft geändert werden, ohne die anderen Informationen über die Punkte neu zu laden. Dafür wird die Eigenschaften separat als Wert zwischen 0 und n gespeichert und mit einer Farbpalette in einen Farbverlauf umgewandelt. n kann dabei maximal $2^{32} - 1$ sein, weil 32 Bit verwendet werden. Mit n und einer Farbpalette unabhängig von den Daten wird der Wert in die Farbe für den Punkt umgewandelt. Die verfügbaren Farbpaletten sind in Abbildung 35 zu sehen.



Abbildung 35: Verfügbare Farbpaletten.

7. Segmente

7.1. Auswahl

Um ein bestimmtes Segment auszuwählen, wird das momentan sichtbare Segment bei der Mausposition berechnet. Als Erstes werden die Koordinaten der Maus mit der Position und Orientierung der Kamera in eine dreidimensionale Position und Richtung umgewandelt. Der Ursprung und die Richtung bilden zusammen einen Strahl.

Im Octree wird vom Root-Knoten aus die Leaf-Knoten gesucht, welche den Strahl enthalten. Dafür wird bei einem Branch-Knoten die acht Kinderknoten betrachtet. Für jeden Kinderknoten wird überprüft, ob der Strahl den Bereich vom Knoten scheidet und gegebenenfalls wird der Abstand zur Kamera berechnet. Weil der Voxel zugehörig zum Knoten entlang der Achsen vom Koordinatensystem ausgerichtet ist, kann mit dem Algorithmus in Abbildung 36 überprüft werden, ob der Strahl den Voxel berührt.

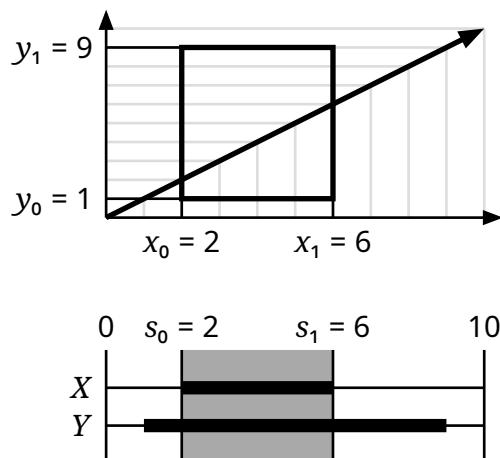


Abbildung 36: Schnittmenge von Strahl und Quadrat in 2D. [9]
Zuerst wird für jede Achse der Bereich bestimmt, für den der Strahl im Quadrat liegen kann.
Die Schnittmenge ist die Überschneidung von den Bereichen für alle Achsen.

Nachdem alle Kinderknoten gefunden wurden, die den Strahl enthalten, wird in diesen nach Abstand zur Kamera aufsteigend weiter gesucht.

Für einen Leaf-Knoten wird der Punkt gesucht, welcher zuerst vom Strahl berührt wird. Dafür werden alle Punkte im Knoten betrachtet. Für jeden Punkt wird zuerst die Distanz

vom Strahl bestimmt. Wenn die Distanz kleiner als der Radius vom Punkt ist, wird der Abstand zum Ursprung vom Strahl berechnet. Der Punkt mit dem kleinsten Abstand zum Ursprung ist der ausgewählte Punkt. Wenn kein Punkt gefunden wird, wird der nächste Knoten entlang des Strahls betrachtet.

Weil die Knoten nach Distanz sortiert betrachtet werden, kann die Suche abgebrochen werden, sobald ein Punkt gefunden wurde. Alle weiteren Knoten sind weiter entfernt, wodurch die enthaltenen Punkte nicht näher zum Ursprung vom Strahl liegen können.

7.2. Visualisierung

Im Octree kann zu den Punkten in einem Leaf-Knoten mehrere Segmente gehören. Um die Segmente einzeln anzuzeigen, wird jedes Segment zusätzlich separat abgespeichert. Sobald ein Segment ausgewählt wurde, wird dieses geladen und anstatt des Octrees angezeigt. Dabei werden alle Punkte des Segments ohne vereinfachte Detailstufen verwendet.

Die momentan geladenen Knoten vom Octree bleiben dabei geladen, um einen schnellen Wechsel zurück zur vollständigen Punktfolge zu ermöglichen.

7.3. Exportieren

Die Segmente können im Stanford Polygon Format (PLY) Format exportiert werden. Jeder Punkt wird dabei so transformiert, dass die Höhe entlang der z-Achse mit 0 für den tiefsten Punkt gespeichert wird. Die horizontale Position der Punkte wird entlang der x- und y-Achse so verschoben, dass die Ausdehnung vom Segment in der positiven und negativen Richtung gleich ist.

8. Detailstufen

Beim Anzeigen wird vom Root-Knoten aus zuerst geprüft, ob der momentane Knoten von der Kamera aus sichtbar ist. Wenn ein Knoten nicht sichtbar ist, so wird dieser nicht geladen und angezeigt. In Abbildung 37 ist ein Beispiel für das Filtern bei unterschiedlichen Detailstufen gegeben. Weil nur ein Teil vom Knoten von der Kamera aus sichtbar sein muss, können bei größeren Knoten der Großteil der Punkte außerhalb der Kamera liegen und werden trotzdem angezeigt.

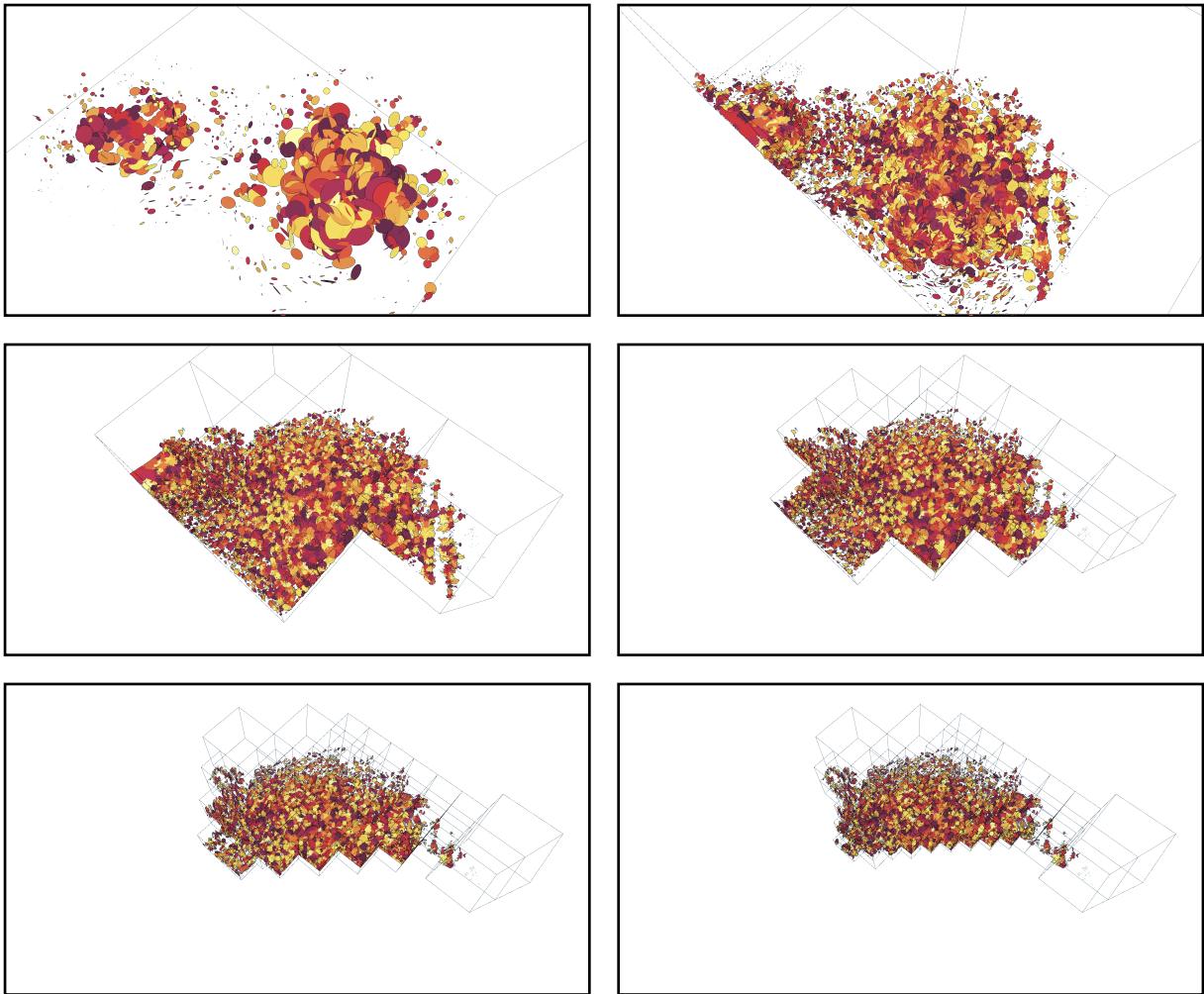


Abbildung 37: Sichtbare Knoten für unterschiedliche Detailstufen. Ein Knoten wird gerendert, solange ein Teil vom Knoten im Sichtfeld der Kamera liegt.

Die Auswahl der Detailstufen kann dabei geändert werden. Im Normalfall wird die gewünschte Detailstufe abhängig vom Abstand zur Kamera ausgewählt. Dadurch werden in der Nähe der Kamera genauere Detailstufen oder die originalen Punkte angezeigt und weit von der Kamera entfernt werden die Detailstufen immer weiter vereinfacht. Eine andere Option ist es, die gleiche Detailstufe für alle Knoten zu verwenden.

V. Auswertung

1. Testdaten

Der benutzte Datensatz [10] beinhaltet 12 Hektar Waldfläche in Deutschland, Baden-Württemberg. Die Daten sind mit Laserscans aufgenommen, wobei die Scans von Flugzeugen (ALS⁵), Drohnen (ULS⁶) und vom Boden (TLS⁷) aus durchgeführt wurden. Dabei entstehen 3D-Punktwolken, welche im komprimiert LAS Dateiformat gegeben sind.

Für die meisten Waldstücke existiert ein ALS, ULS und TLS. Dabei enthält der ALS die wenigsten und der TLS die meisten Punkte. Bei ALS und ULS sind die Punkte gleichmäßig über das gescannte Gebiet verteilt. Bei TLS wird von einem zentralen Punkt aus gescannt, wodurch die Punktedichte nach außen immer weiter abnimmt.

2. Importgeschwindigkeit

Für den Import sind in [Kapitel VI-2](#) die Messdaten für unterschiedliche Datensätze gegeben. Die Eigenschaften vom verwendeten System zum Messen sind in [Kapitel VI-1](#) gelistet. Dabei wurden nur die ALS und ULS verwendet. Durch die hohe Punktanzahl und die ungleichmäßige Verteilung bei den TLS sind diese nicht gut für die Analyse geeignet.

In Abbildung 38 ist der Durchsatz beim Import angegeben. Dabei wird eine Importgeschwindigkeit von circa 400 000 Punkte pro Sekunde für die Datensätze erreicht.

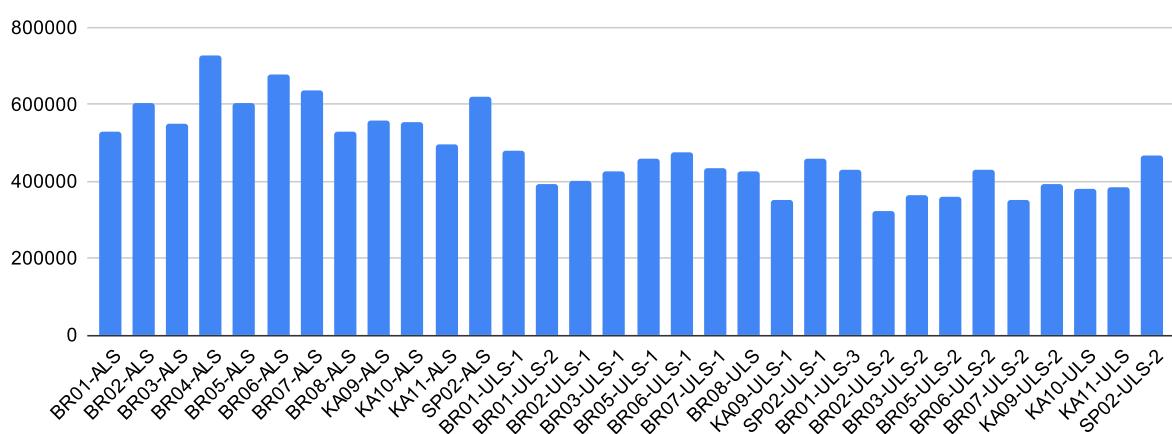


Abbildung 38: Geschwindigkeit vom Import in Punkte pro Sekunde.

In Abbildung 39 ist die Dauer für die einzelnen Phasen vom Import aufgeschlüsselt. Bei jeder Phase wird jeder Punkt betrachtet, aber am längsten wird für die Segmentierung und Berechnungen von Informationen benötigt, weil diese mehr Aufwand pro Punkt benötigen.

⁵aircraft laser scan

⁶uncrewed aerial vehicle laser scan

⁷terrestrial laser scan

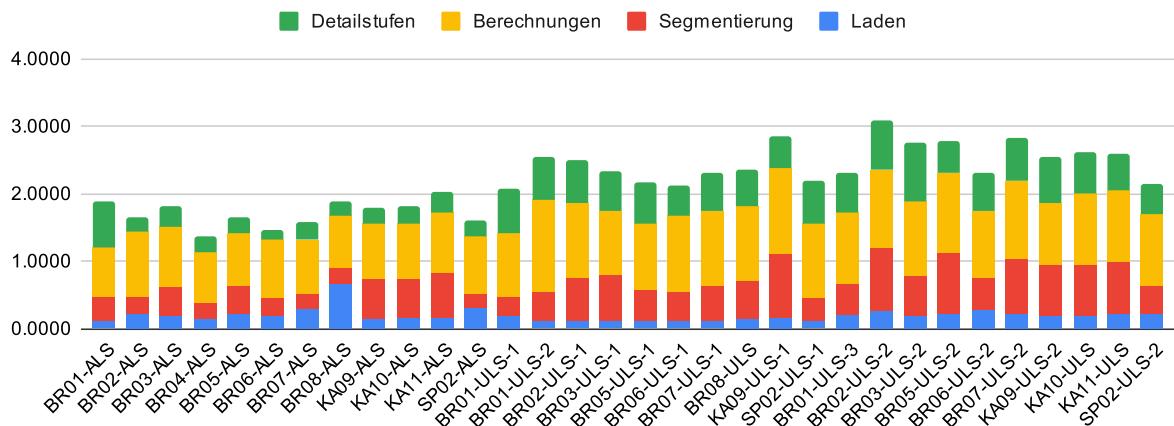


Abbildung 39: Dauer für die einzelnen Importphasen in μs pro Punkt.

3. Segmentierung in Bäume

Ein Beispiel für eine Segmentierung ist in Abbildung 40 gegeben. Die meisten Bäume werden korrekt erkannt und zu unterschiedlichen Segmenten zugeordnet. Je weiter die Spitzen der Bäume voneinander getrennt sind, desto besser können die Bäume voneinander getrennt werden.

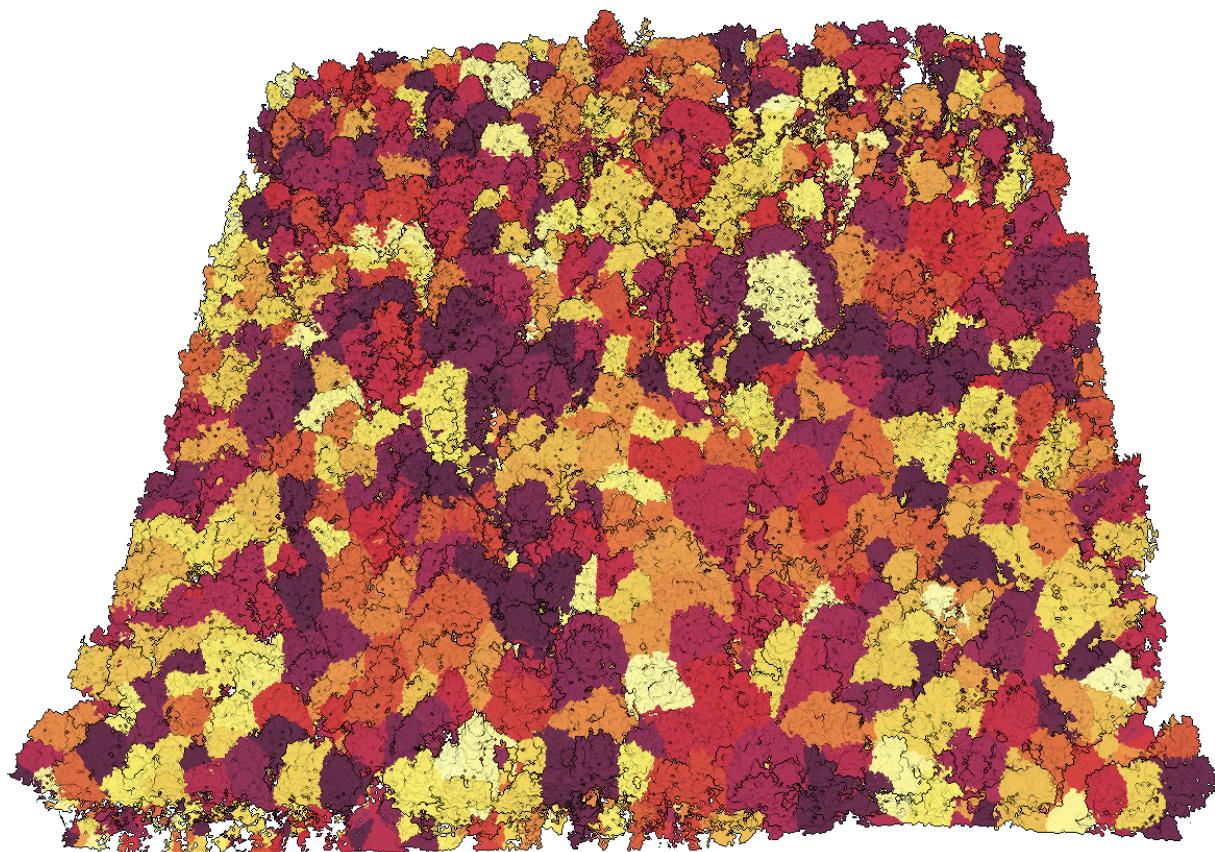


Abbildung 40: Segmentierung von einer Punktwolke.

Punkte, welche zu keinem Baum gehören, werden trotzdem zu den Segmenten zugeordnet. Bei Bereichen ohne Bäume entstehen dadurch Segmente wie in Abbildung 41. Die Punkte in freien Flächen werden zu eigenen Segmenten zusammengefasst. Wenn die Punkte in der Nähe von einem Baum liegen, werden diese zu dem Baum hinzugefügt.

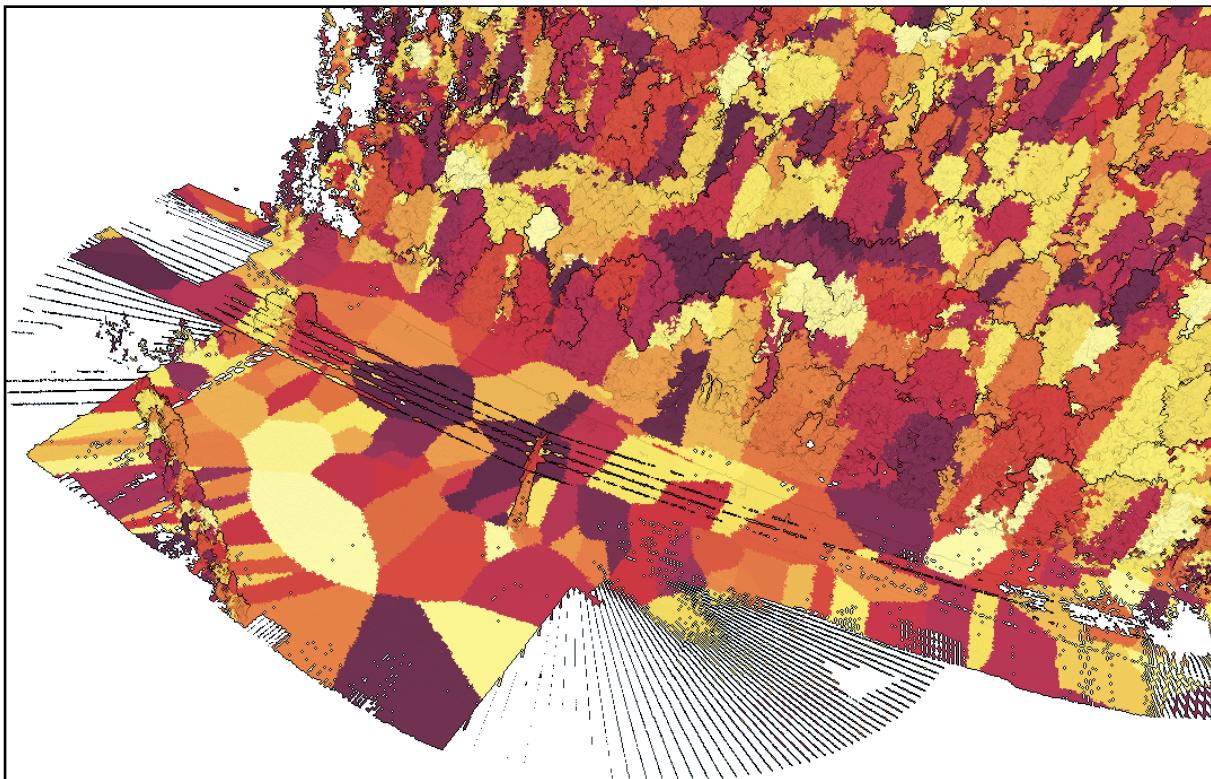


Abbildung 41: Segmente bei Gebieten ohne Bäume.

Kleine Bereiche werden vor der Zuordnung entfernt. Dadurch wird vermieden, dass ein Baum in mehrere Segmente unterteilt wird. Wenn die Spitze von einem Baum gerade so in einer Scheibe liegt, so ist der zugehörige Bereich klein und wird gefiltert. Dadurch wird kein neues Segment für den Baum erstellt und die Punkte werden dem nächsten Baum zugeordnet. Der Effekt ist in Abbildung 42 zu sehen.

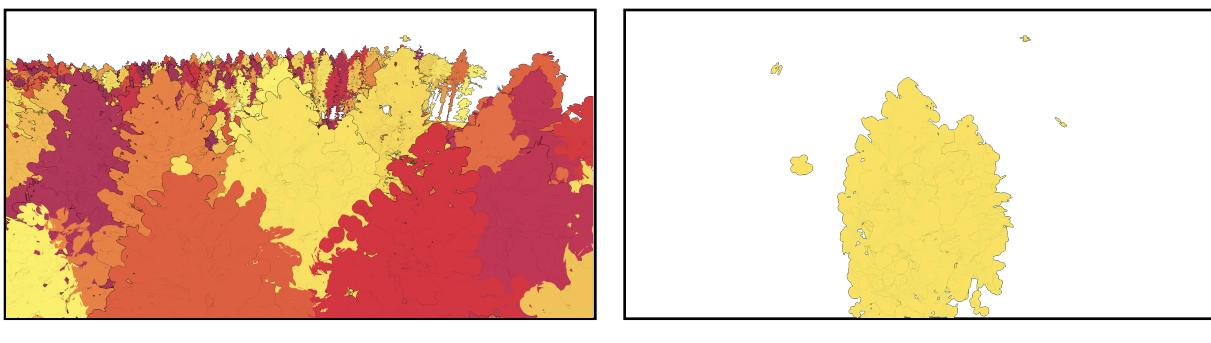


Abbildung 42: Segmentierungsfehler bei Baumspitzen.

4. Analyse von Segmenten

In Abbildung 43 und Abbildung 44 ist ein Segment basierend auf den berechneten Eigenarten eingefärbt gegeben und zusätzlich mit den größten und kleinste Werte gefiltert.

Die Punkte mit der größten Krümmung gehören zu den Blättern, was eine teilweise Filterung ermöglicht. Die Punkte beim Stamm haben eine geringere Krümmung, aber auch Punkte, die zu den Blättern gehören, können eine geringe Krümmung haben.

Punkte zugehörig zu einer geringen horizontalen Ausdehnung gehören immer zum Stamm oder der Spitze der Krone, wodurch der Stamm identifiziert werden kann.

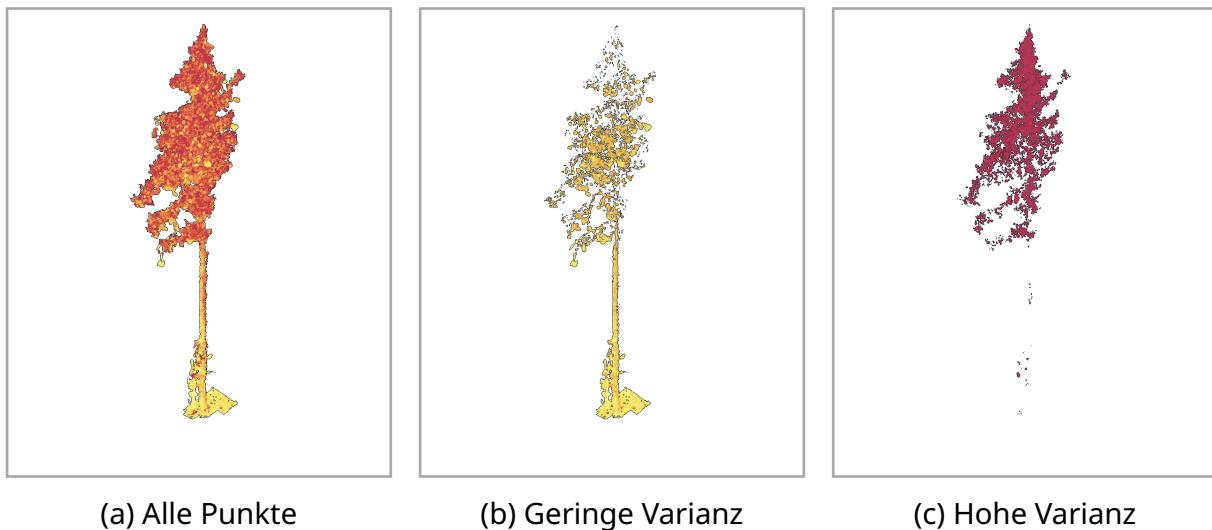


Abbildung 43: Punktfolke basierend auf der Krümmung eingefärbt.

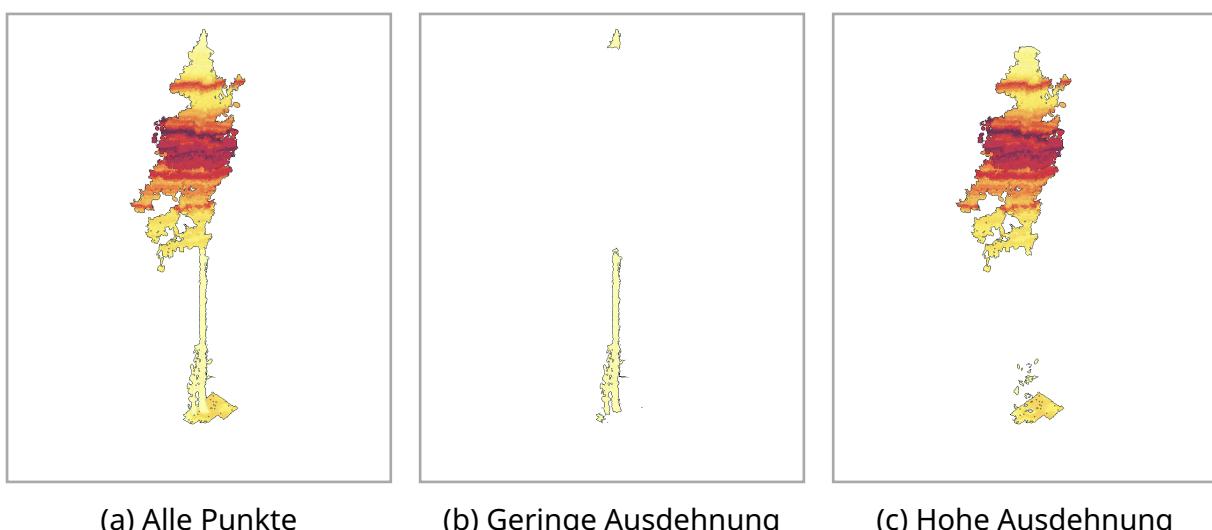


Abbildung 44: Punktfolke basierend auf der Ausdehnung eingefärbt.

5. Triangulierung

Ein Beispiel für die Triangulation ist in Abbildung 45 gegeben. Mit dem Ball-Pivoting Algorithmus wird eine äußere Hülle für die Punkte bestimmt, wodurch der Algorithmus auch für eine Baumkrone mit Blättern geeignet ist. Beim Baumstamm liegen alle Punkte auf der Oberfläche, wodurch diese problemlos trianguliert werden können. Bei der Krone sind die Punkte im Raum verteilt, wodurch diese nicht auf einer eindeutigen Oberfläche liegen.

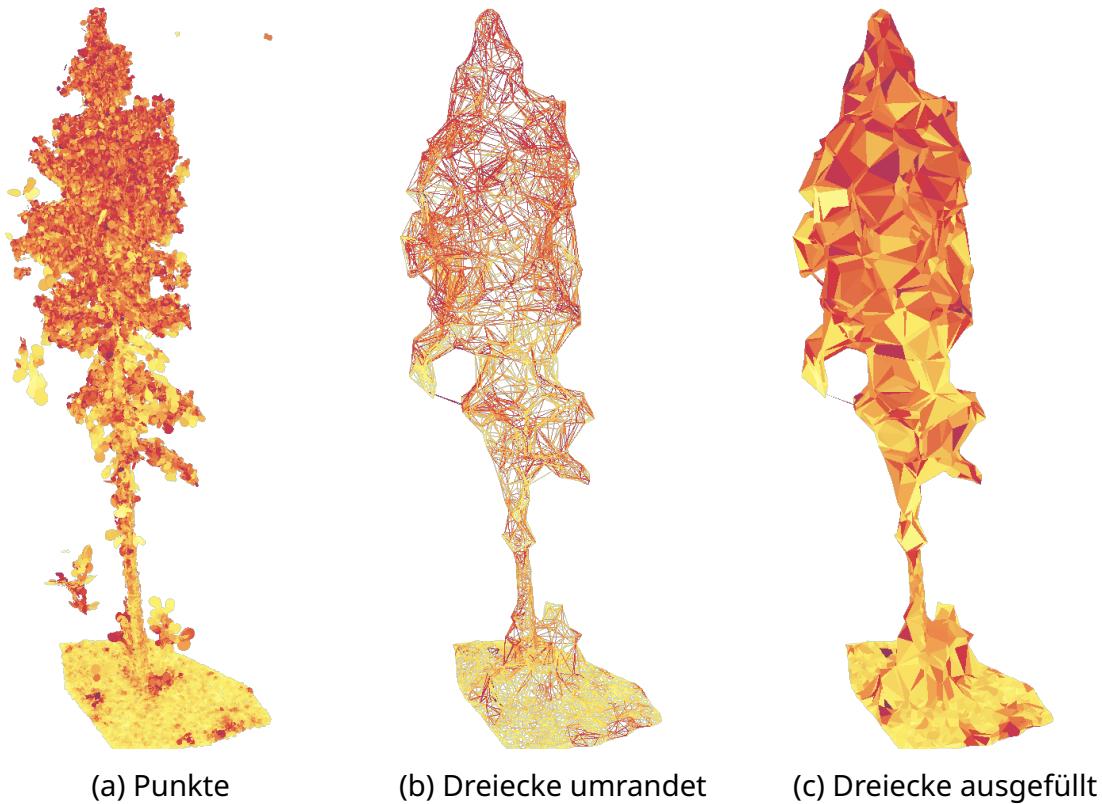


Abbildung 45: Beispiel für eine Triangulierung von einem Baum mit $\alpha = 1m$.

6. Visualisierung

In Abbildung 46 ist die benötigte Renderzeit für die Beispiele aus Abbildung 47 gegeben.

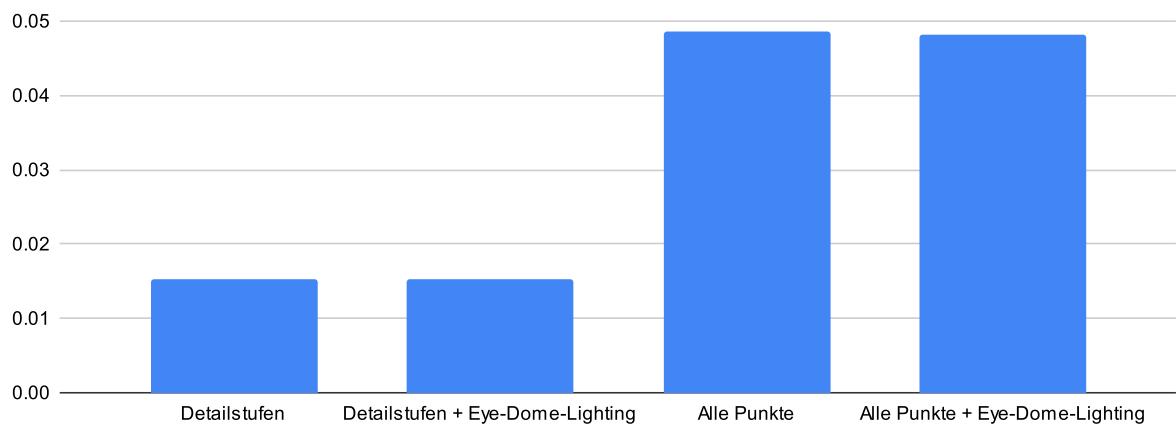


Abbildung 46: Renderzeit in Sekunden für einen Datensatz mit 59 504 504 Punkten.

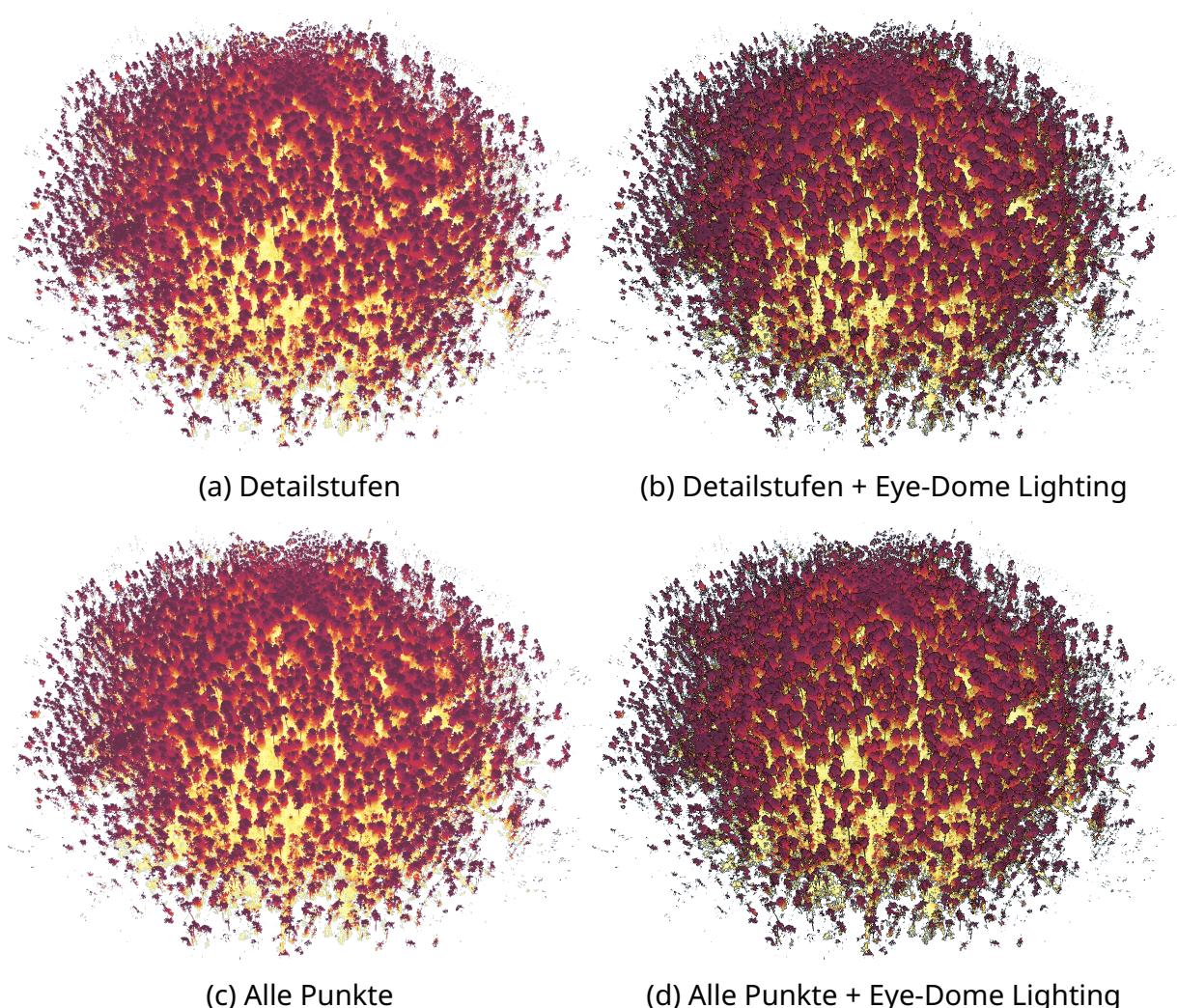


Abbildung 47: Visualisierung von einem Datensatz mit 59 504 504 Punkten.

Die Detailstufen ermöglichen eine Visualisierung vom kompletten Datensatz in Echtzeit ohne einen sichtbaren Detailverlust. Der zusätzliche Speicherbedarf für die Detailstufen ist in Abbildung 48 gelistet. Für die Detailstufen wird zusätzlich die Hälfte vom Speicherbedarf für die Punkte benötigt.

Das Eye-Dome Lighting ermöglicht eine bessere Wahrnehmung der verlorenen Tiefeninformationen. Der Berechnungsaufwand ist dabei unabhängig von der Anzahl der sichtbaren Punkte, wodurch der Effekt auch bei einer großen Anzahl von Punkten die Renderzeit nicht beeinflusst.

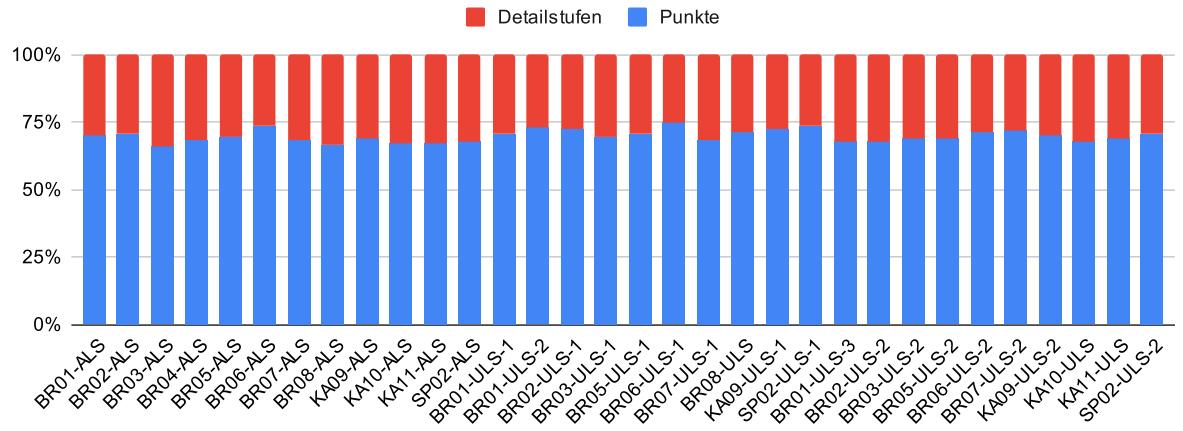


Abbildung 48: Speicherbedarf für die Punkte und Detailstufen.

7. Fazit

Die Software ermöglicht den Übergang von den Punktdaten ohne weitere Informationen zu einer interaktiven Visualisierung vom Waldstück. Dadurch kann sich ein Überblick über das gescannte Waldstück gemacht werden, wodurch das gewünschte Ziel erreicht ist. Trotzdem gibt es noch Fehler bei der Methodik und Implementierung, welche ausgebessert werden können.

Die Segmentierung unterteilt die Punkte in einzelne Bäume. Wenn die Kronen der Bäume klar getrennte Spitzen haben, werden diese problemlos unterteilt. Dadurch werden manche Waldstücke gut segmentiert, aber je näher die Kronen der Bäume zueinander sind, desto wahrscheinlicher werden mehrere Bäume zu einem Segment zusammengefasst. Vor der Segmentierung muss der Mindestabstand zwischen Segmenten und die Breite der Scheiben festgelegt werden. Die Parameter müssen passend für den Datensatz gewählt werden, was eine Anpassungsmöglichkeit, aber auch eine Fehlerquelle ermöglicht.

Bei der Analyse von einem Baum werden Daten für jeden Punkt im Baum und für den gesamten Baum berechnet. Für die einzelnen Punkte werden Punktgröße, Normale für die Visualisierung und die lokale Krümmung problemlos berechnet. Die Berechnung der Ausdehnung ist funktioniert für die meisten Bereiche vom Baum. Punkte vom Waldboden werden zu den Bäumen zugeordnet, wodurch die Ausdehnung am Boden höher als die Ausdehnung vom eigentlichen Stamm ist.

Die Triangulierung berechnet ein Mesh für die Segmente. Dabei wird eine äußere Hülle bestimmt, was für Bäume geeignet ist.

Die Visualisierung kann die berechneten Daten ohne Probleme visualisieren. Durch die Detailstufen können auch größere Datenmengen interaktiv angezeigt werden.

8. Ausblick

Momentan werden die ermittelten Daten nur für die Visualisierung verwendet. Um die Daten als Basis für weitere Analysen zu verwenden, müssen diese in einem festgelegten Format gespeichert werden.

Vor der Visualisierung müssen die Daten importiert werden. Je größer die Datenmenge, desto länger dauert der Import und während des Imports können die Daten noch nicht inspiziert werden. Die Möglichkeit die Zwischenergebnisse vom Importprozess anzuzeigen würde das Anpassen von Importparametern erleichtern.

VI. Appendix

1. Systemeigenschaften

Betriebssystem	Windows 11
Prozessor	Intel(R) Core(TM) i7-9700KF CPU @ 3.60 GHz
Grafikkarte	NVIDIA GeForce GTX 1660 SUPER
RAM	2 x G.Skill F4-3200C16-8GIS
Festplatte	SanDisk SSD PLUS 2000GB

Tabelle 5: Überblick über die Systemeigenschaften

Physische Kerne	8
Logische Kerne	8
Maximale Taktrate	4,6 GHz
Cachegröße (L1, L2, L3)	512 KiB, 2 MiB, 12 MiB

Tabelle 6: Prozessoreigenschaften

Basistaktung	1530 Mhz
Boost-Taktung	1785 Mhz
Speicherkonfiguration	6 GB GDDR6
Speicherschnittstelle	192 Bit

Tabelle 7: Grafikkarteneigenschaften

Größe	2 × 8 GiB
Taktrate	2133 MHz

Tabelle 8: RAM-Eigenschaften

Aktion	1 GiB	5 GiB	10 GiB
Lesen	776 MiB/s	384 MiB/s	218 MiB/s
Schreiben	1739 MiB/s	1929 MiB/s	270 MiB/s

Tabelle 9: Sequenzielle Lese- und Schreibgeschwindigkeit der Festplatte mit 4 KiB Blöcken für unterschiedliche Dateigrößen.

2. Messwerte vom Import

Datensatz	Datei	Daten Punkte	Segment Punkte	Detailstufen Punkte
BR01-ALS	ALS-on_BR01_2019-07-05_300m	12531758	12504511	5380805
BR02-ALS	ALS-on_BR02_2019-07-05_200m	5469363	5452383	2244994
BR03-ALS	ALS-on_BR03_2019-07-05_300m	12284905	12231654	6264253
BR04-ALS	ALS-on_BR04_2019-07-05_140m	2638216	2625729	1207650
BR05-ALS	ALS-on_BR05_2019-07-05_300m	13401630	13355278	5898786
BR06-ALS	ALS-on_BR06_2019-07-05_200m	6269955	6261724	2265152
BR07-ALS	ALS-on_BR07_2019-07-05_140m	3196739	3179689	1485584
BR08-ALS	ALS-on_BR08_2019-07-05_140m	2871871	2854955	1432741
KA09-ALS	ALS-on_KA09_2019-07-05_300m	14536939	14531495	6574366
KA10-ALS	ALS-on_KA10_2019-07-05_300m	13413120	13396658	6611872
KA11-ALS	ALS-on_KA11_2019-07-05_300m	14840932	14830163	7193771
SP02-ALS	ALS-on_SP02_2019-07-05_140m	2810265	2799533	1326886
BR01-ULS-1	ULS-off_BR01_2019-12-04	52179533	52146485	21454777
BR01-ULS-2	ULS-off_BR01_2020-03-27	62548947	62504831	23143124
BR02-ULS-1	ULS-off_BR02_2020-04-01	76639411	76562491	29516882
BR03-ULS-1	ULS-off_BR03_2020-03-31	63391431	63303160	28182668
BR05-ULS-1	ULS-off_BR05_2020-03-26	52169435	52117995	21483555
BR06-ULS-1	ULS-off_BR06_2020-03-27	60182129	60115839	20478704
BR07-ULS-1	ULS-off_BR07_2020-03-26	57159079	57091559	26186466
BR08-ULS	ULS-off_BR08_2020-03-31	66483369	66433040	26873864
KA09-ULS-1	ULS-off_KA09_2019-12-10	59967504	59863457	22675116
SP02-ULS-1	ULS-off_SP02_2020-04-01	64818104	64784916	23432531
BR01-ULS-3	ULS-on_BR01_2019-09-12	39420629	39323345	18779424
BR02-ULS-2	ULS-on_BR02_2019-08-24	41555427	41416491	19723069
BR03-ULS-2	ULS-on_BR0308_2019-08-24	83344946	83175989	37389042
BR05-ULS-2	ULS-on_BR05_2019-09-12	50247658	49963353	22930728
BR06-ULS-2	ULS-on_BR06_2019-09-12	48369613	48156311	19670848
BR07-ULS-2	ULS-on_BR07_2019-08-24	46833772	46696161	18257769
KA09-ULS-2	ULS-on_KA09_2019-09-13	50863340	50716258	22022669
KA10-ULS	ULS-on_KA10_2019-09-13	48167924	48020812	23068621
KA11-ULS	ULS-on_KA11_2019-09-06	58549518	58446138	26664882
SP02-ULS-2	ULS-on_SP02_2019-09-03	30811866	30753803	12805708

Tabelle 10: Messwerte (1).

Datensatz	Segmente	Punkte Laden (s)	Segmentierung (s)	Berechnungen (s)	Detailstufen (s)
BR01-ALS	4097	1.372	4.426	9.302	8.517
BR02-ALS	2491	1.132	1.503	5.184	1.222
BR03-ALS	5106	2.266	5.228	11.155	3.679
BR04-ALS	1459	0.378	0.601	2.006	0.638
BR05-ALS	4648	3.011	5.379	10.637	3.234
BR06-ALS	2007	1.241	1.600	5.400	0.984
BR07-ALS	1225	0.940	0.691	2.559	0.820
BR08-ALS	1196	1.909	0.689	2.219	0.620
KA09-ALS	4054	2.055	8.421	12.183	3.479
KA10-ALS	4038	2.202	7.471	11.122	3.412
KA11-ALS	5194	2.394	9.873	13.190	4.501
SP02-ALS	1379	0.845	0.607	2.361	0.719
BR01-ULS-1	2474	10.167	13.862	49.542	35.177
BR01-ULS-2	4262	6.977	26.354	86.745	38.713
BR02-ULS-1	7736	8.712	49.823	83.656	49.818
BR03-ULS-1	7659	7.656	42.610	60.220	38.040
BR05-ULS-1	4096	6.527	23.085	52.046	31.743
BR06-ULS-1	5839	7.629	25.536	68.081	25.841
BR07-ULS-1	5557	6.409	29.543	63.194	32.952
BR08-ULS	5803	8.803	38.348	73.435	35.639
KA09-ULS-1	6159	10.408	55.632	76.126	28.899
SP02-ULS-1	3441	7.140	21.351	72.885	40.132
BR01-ULS-3	3306	7.947	17.703	42.123	23.683
BR02-ULS-2	4766	11.024	39.092	47.874	30.400
BR03-ULS-2	7238	15.692	48.614	92.353	73.063
BR05-ULS-2	7412	11.243	46.079	58.984	23.256
BR06-ULS-2	7770	13.316	23.665	47.293	27.872
BR07-ULS-2	5873	10.479	38.555	53.047	30.825
KA09-ULS-2	5293	9.557	38.339	47.001	34.867
KA10-ULS	5044	9.564	35.523	51.319	29.783
KA11-ULS	5539	12.231	45.555	62.772	31.474
SP02-ULS-2	2583	6.853	12.473	32.998	13.471

Tabelle 11: Messwerte (2).

3. KD-Baum

Für eine Menge von Punkten kann ein KD-Baum bestimmt werden. Mit diesem kann effizient bestimmt werden, welche Punkte innerhalb einer Kugel mit beliebiger Position und Radius liegen. Ein Beispiel für einen KD-Baum ist in Abbildung 49 gegeben.

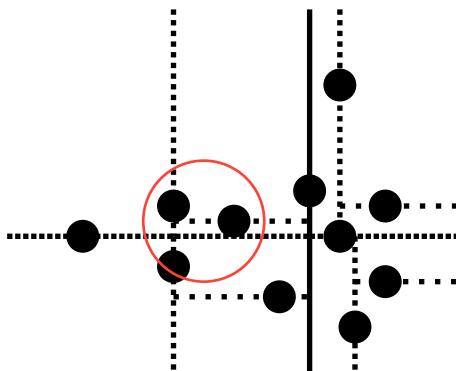


Abbildung 49: KD-Baum für Punkte in 2D. Für jede Unterteilung ist die Trennegrade gepunkteter gezeichnet. Weil der rote Kreis vollständig auf einer Seite der ersten Unterteilung ist, müssen die Punkte auf der anderen Seite nicht betrachtet werden.

3.1. Konstruktion

Für die Konstruktion von einem KD-Baum werden nur die Positionen der Punkte benötigt.

Zuerst wird für die Punkte entlang der ersten Dimension der Median bestimmt. Dabei wird der *Quickselect*-Algorithmus [11] verwendet. Der Median hat als Index die halbe Anzahl der Punkte. Ist die Anzahl der Punkte ungerade, so kann der Index auf- oder abgerundet werden, solange bei der Suche die gleiche Strategie verwendet wird. Wie beim *Quicksort*-Algorithmus wird ein beliebiges Pivot-Element ausgewählt, mit diesem die Positionen entlang der Dimension unterteilt werden. Die Positionen werden einmal iteriert und kleinere Positionen vor dem Pivot und größere Positionen nach dem Pivot verschoben. Der Pivot ist am Index, wo es in der sortierten List wäre. Um den Median zu finden, wird nur der Teil von den Punkten betrachtet, welcher den zugehörigen Index beinhaltet. Die Unterteilung wird so lange wiederholt, bis der Median bekannt ist.

Durch den *Quickselect*-Algorithmus sind die Positionen nach der Bestimmung vom Median in kleinere und größere Positionen unterteilt. Die Ebene durch den Punkt teilt dabei den Raum und alle Punkte mit kleinerem Index liegen auf der anderen Seite als die Punkte mit größerem Index. Die beiden Hälften werden in der gleichen Weise unterteilt. Dabei wird die nächste, beziehungsweise für die letzte Dimension wieder die erste Dimension verwendet.

Der zugehörige Binärbaum muss nicht gespeichert werden, da diese implizit entsteht. Für jede Unterteilung wird die Position vom Median gespeichert, dass diese für die Suchanfragen benötigt werden.

3.2. Suche mit festem Radius

Bei dieser Suchanfrage werden alle Punkte gesucht, welche in einer Kugel mit bekanntem Zentrum und Radius liegen. Von der Root-Knoten aus wird der Baum dabei durchsucht. Bei jeder Unterteilung wird dabei überprüft, wie die Kugel zur teilenden Ebene liegt. Ist die Kugel vollständig auf einer Seite, so muss nur der zugehörige Teilbaum weiter durchsucht werden. Liegen Teile der Kugel auf beiden Seiten, so müssen beide Teilbaum weiter durchsucht werden.

Dabei wird bei jeder Unterteilung überprüft, ob die zugehörige Position in der Kugel liegt und gegebenenfalls zum Ergebnis hinzugefügt.

Mit der gleichen Methode kann effizient bestimmt werden, ob eine Kugel leer ist. Dafür wird beim ersten gefundenen Punkt in der Kugel die Suche abgebrochen.

3.3. Suche mit fester Anzahl

Bei dieser Suchanfrage wird für eine feste Anzahl k die k -nächsten Punkte für ein bestimmtes Zentrum gesucht. Dafür werden die momentan k -nächsten Punkte gespeichert und nach Entfernung sortiert. Die Entfernung zum k -ten Punkt wird als Maximaldistanz verwendet. Solange noch nicht k Punkte gefunden sind, kann ∞ oder ein beliebiger Wert als Maximalabstand verwendet werden.

Es wird wieder von der Wurzel aus der Baum durchsucht. Bei jeder Unterteilung wird zuerst in der Hälfte vom Baum weiter gesucht, die das Zentrum enthält. Dabei werden die Punkte zu den besten Punkten hinzugefügt, die näher am Zentrum als die Maximaldistanz liegen. Sobald k Punkte gefunden sind, wird dadurch die Maximaldistanz kleiner, weil der Punkte mit der alten Maximaldistanz nicht mehr zu den k -nächsten Punkten gehört.

Nachdem ein Teilbaum vollständig durchsucht ist, wird überprüft, ob Punkte aus dem anderen Teilbaum näher am Zentrum liegen können. Dafür wird der Abstand vom Zentrum zur Ebene bestimmt. Ist der Abstand größer als die Maximaldistanz, so kann kein Punkt näher am Zentrum liegen und der Teilbaum muss nicht weiter betrachtet werden.

3.4. Schnelle Suche

Sobald ein Teilbaum nur noch wenige Punkte beinhaltet, ist es langsamer zu überprüfen, welche Punkte näher sein können, als alle Punkte zu betrachten. Deshalb wird für Teilbäume mit weniger als 32 Punkten die Punkte linear iteriert, wodurch Rekursion vermieden wird.

4. Baum (Datenstruktur)

Ein Baum ermöglichen räumlich dünnbesetzte Daten effizient zu speichern. Dafür wird der Raum unterteilt, und nur für Bereiche mit Daten weitere Knoten gespeichert.

4.1. Konstruktion

Zuerst wird die räumliche Ausdehnung der Daten bestimmt. Dieser Bereich wird dem Root-Knoten zugeordnet. Solange noch zu viele Datenwerte im Bereich von einem Kno-

ten liegen, wird dieser weiter unterteilt. Dafür wird der zugehörige Bereich entlang aller Dimensionen halbiert und jeder Teilbereich einem Kinderknoten zugeordnet. Bei einem Quadtree in 2D entstehen dadurch vier Kinderknoten und bei einem Octree in 3D acht Kinderknoten. Der Daten gehören nicht mehr zum unterteilten Knoten, sondern zu den Kinderknoten. Der unterteilte Knoten speichert stattdessen die Kinderknoten.

In Abbildung 50 und Abbildung 51 sind Beispiele in 2D und 3D gegeben.

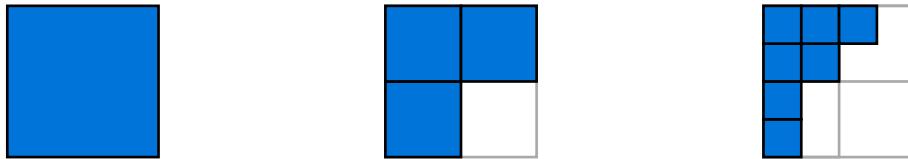


Abbildung 50: Unterschiedliche Stufen von einem Quadtree.

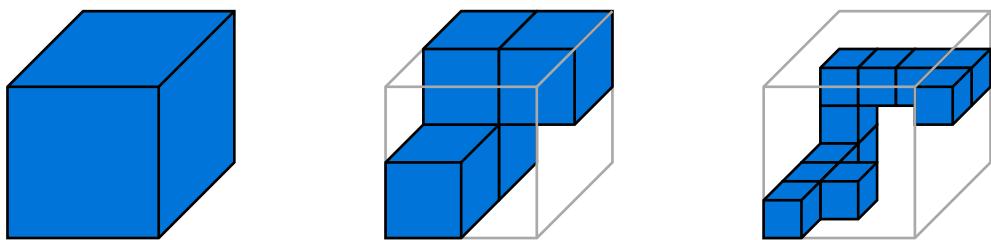


Abbildung 51: Unterschiedliche Stufen von einem Octree.

4.2. Suchanfrage

Bei einer Suchanfrage wird vom Root-Knoten aus der Leaf-Knoten gesucht, welche die gesuchte Position enthält. Dafür wird so lange der momentane Knoten ein Branch-Knoten ist berechnet, welcher der Kinderknoten die Position enthält und von diesem aus weiter gesucht.

Glossar

Koordinatensystem ist eine Menge von Achsen, mit den eine Position genau beschrieben werden kann. Im Normalfall werden kartesische Koordinaten verwendet, welche so orientiert sind, dass die x-Achse nach rechts, die y-Achse nach oben und die z-Achse nach hinten zeigt.

Punkt ist eine dreidimensionale Position, welcher zusätzlichen Informationen zugeordnet werden können.

Punktwolke ist eine Menge von Punkten. Für alle Punkte sind die gleichen zusätzlichen Informationen vorhanden.

Normale ist ein normalisierter dreidimensionaler Vektor, welcher die Orientierung einer Oberfläche von einem Objekt angibt. Der Vektor ist dabei orthogonal zur Oberfläche, kann aber in das Objekt oder aus dem Objekt gerichtet sein.

Voxel ist ein Würfel im dreidimensionalen Raum. Die Position und Größe vom Voxel kann explizit abgespeichert oder relative zu den umliegenden Voxeln bestimmt werden.

Tree ist eine Datenstruktur, bestehend aus Knoten, welche wiederum Kinderknoten haben können. Die Knoten selber können weitere Informationen enthalten.

Octree ist eine Baumdatenstruktur, bei dem ein Knoten acht Kinderknoten haben kann. Mit einem Octree kann ein Voxel aufgeteilt werden. Jeder Knoten gehört zu einem Voxel, welcher gleichmäßig mit den Kinderknoten weiter unterteilt wird.

Quadtree ist eine Baumdatenstruktur, bei dem ein Knoten vier Kinderknoten haben kann. Statt eines Voxels bei einem Octree, wird ein zweidimensionales Quadrat unterteilen.

Leaf-Knoten ist ein Knoten, welcher keine weiteren Kinderknoten hat. Für Punktwolken gehört jeder Punkt zu genau einem Leaf-Knoten.

Branch-Knoten ist ein Knoten, welcher weitere Kinderknoten hat.

Root-Knoten ist der erste Knoten im Tree, alle anderen Knoten sind direkte oder indirekte Kinderknoten vom Root-Knoten.

KD-Baum ist eine Datenstruktur, um im k -dimensionalen Raum für eine Position die nächsten Punkte zu bestimmen.

Bibliographie

- [1] J. J. Donager, A. J. Sánchez Meador, und R. C. Blackburn, „Adjudicating perspectives on forest structure: how do airborne, terrestrial, and mobile lidar-derived estimates compare?”, *Remote Sensing*, Bd. 13, Nr. 12, S. 2297–2298, 2021.
- [2] M. Disney, „Terrestrial LiDAR: a three-dimensional revolution in how we look at trees”, *New Phytologist*, Bd. 222, Nr. 4, S. 1736–1741, 2019, doi: <https://doi.org/10.1111/nph.15517>.
- [3] T. Suzuki, S. Shiozawa, A. Yamaba, und Y. Amano, „Forest Data Collection by UAV Lidar-Based 3D Mapping: Segmentation of Individual Tree Information from 3D Point Clouds”, *International Journal of Automation Technology*, Bd. 15, S. 313–323, 2021, doi: [10.20965/ijat.2021.p0313](https://doi.org/10.20965/ijat.2021.p0313).
- [4] A. Burt, M. Disney, und K. Calders, „Extracting individual trees from lidar point clouds using treeseg”, *Methods in Ecology and Evolution*, Bd. 10, Nr. 3, S. 438–445, 2019, doi: <https://doi.org/10.1111/2041-210X.13121>.
- [5] L. Graham, „The LAS 1.4 specification”, *Photogrammetric engineering and remote sensing*, Bd. 78, Nr. 2, S. 93–102, 2012.
- [6] M. Isenburg, „LASzip: lossless compression of LiDAR data”, *Photogrammetric engineering and remote sensing*, Bd. 79, Nr. 2, S. 209–217, 2013.
- [7] C. Hug, P. Krzystek, und W. Fuchs, „Advanced Lidar Data Processing with Las-tools”, 2004. [Online]. Verfügbar unter: <https://api.semanticscholar.org/CorpusID:14167994>
- [8] M. Schütz, S. Ohrhallinger, und M. Wimmer, „Fast Out-of-Core Octree Generation for Massive Point Clouds”, *Computer Graphics Forum*, Bd. 39, Nr. 7, S. 155–167, 2020, doi: <https://doi.org/10.1111/cgf.14134>.
- [9] A. Majercik, C. Crassin, P. Shirley, und M. McGuire, „A Ray-Box Intersection Algorithm and Efficient Dynamic Voxel Rendering”, *Journal of Computer Graphics Techniques (JCGT)*, Bd. 7, Nr. 3, S. 66–81, Sep. 2018, [Online]. Verfügbar unter: <http://jcgtrg.org/published/0007/03/04/>
- [10] H. Weiser *u. a.*, „Terrestrial, UAV-borne, and airborne laser scanning point clouds of central European forest plots, Germany, with extracted individual trees and manual forest inventory measurements”. [Online]. Verfügbar unter: <https://doi.org/10.1594/PANGAEA.942856>
- [11] C. A. R. Hoare, „Algorithm 65: Find”, *Commun. ACM*, Bd. 4, Nr. 7, S. 321–322, Juli 1961, doi: [10.1145/366622.366647](https://doi.org/10.1145/366622.366647).