

Wörter: 7003

## To-dos

1: Mit Bild(ern) auflockern.	1
2: Stand der Technik	3
3: Mehr Baumeigenschaften	9
4: Baumeigenschaften + ? → Segmente	10
5: Segmente + Eigenschaften + ? → Klassifizierung?	10
6: Auswahl $\alpha$	15
7: GitHub-Release	24
8: Benutzung aus dem Readme hier?	24
9: Voxel Distance?   Point on ray?	29
10: Implementierung Segment export	29
11: Ergebnisse	31
12: Was ALS, ULS und TLS Daten	31
13: Kurze namen für Tabellen	31
14: TLS Daten	31
15: neues Bild	33
16: Fehler wenn spitze gerade so in scheibe	33
17: Berechnung Ergebnisse	33
18: Triangulierung Ergebnisse	33
19: Vergleich $\alpha$	33
20: Visualisierung Ergebnisse	34
21: Fazit	34
22: Sinnvolle Ordnung	35
23: Systemeigenschaften	40
24: Beschreibung	41
25: Beschreibung	41

# Masterarbeit

Berechnung charakteristischen Eigenschaften  
von botanischen Bäumen mithilfe von 3D-Punkt-  
wolken.

**Name:** Anton Wetzel

E-Mail: [anton.wetzel@tu-ilmenau.de](mailto:anton.wetzel@tu-ilmenau.de)

Matrikelnummer: 60451

Studiengang: Informatik Master

**Betreuer:** Tristan Nauber

E-Mail: [tristan.nauber@tu-ilmenau.de](mailto:tristan.nauber@tu-ilmenau.de)

**Professor:** Prof. Dr.-Ing. Patrick Mäder

E-Mail: [patrick.maeder@tu-ilmenau.de](mailto:patrick.maeder@tu-ilmenau.de)

Fachgebiet: Data-intensive Systems and Visualizati-  
on Group

Datum: 16.02.2024

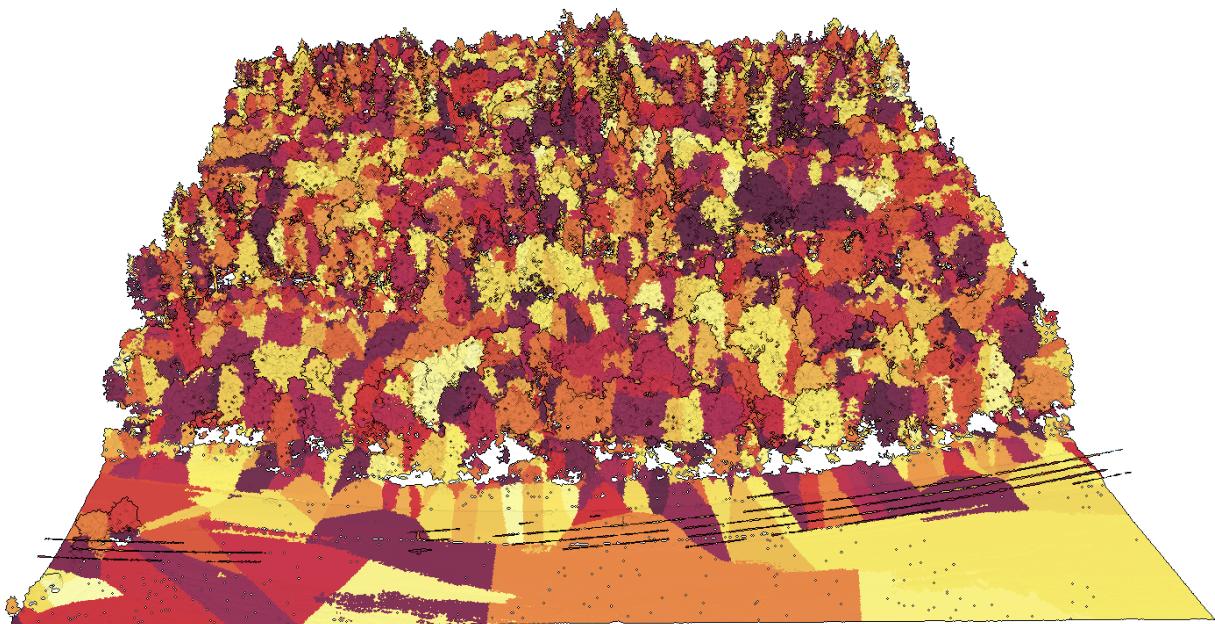


## **Abstrakt**

Diese Arbeit beschäftigt sich mit der Verarbeitung und Visualisierung von Punktwolken von Waldstücken. Dabei wird der komplette Ablauf von den initialen Daten mit den Positionen der Punkte vom gesamten Waldstück bis zur Visualisierung der einzelnen Bäume mit relevanten Informationen durchgeführt.

Dafür werden die Punkte in einzelne Bäume segmentiert, relevante Informationen für die Bäume und die einzelnen Punkte berechnet und alle Daten für die Visualisierung in Echtzeit vorbereitet.

Im zugehörigen Softwareprojekt sind die Algorithmen und Visualisierung umgesetzt, wodurch diese an die besonderen Daten der Waldstücke angepasst werden können.



# Inhaltsverzeichnis

<b>I. Einleitung .....</b>	<b>1</b>
1. Motivation .....	1
2. Themenbeschreibung .....	1
3. Struktur der Arbeit .....	1
<b>II. Stand der Technik .....</b>	<b>3</b>
<b>III. Methodik .....</b>	<b>4</b>
1. Segmentierung der Punktewolke in Bäume .....	4
1.1. Ablauf .....	4
1.2. Bereiche bestimmen .....	4
1.3. Koordinaten bestimmen .....	6
1.4. Punkte zuordnen .....	6
2. Analyse von Segmenten .....	7
2.1. Eigenschaften .....	8
2.2. Segmentierung von einem Baum .....	10
2.3. Eigenschaften für Visualisierung .....	10
2.4. Baumart .....	10
3. Triangulierung .....	10
3.1. Ziel .....	10
3.2. Ball-Pivoting-Algorithmus .....	10
4. Visualisierung .....	15
4.1. Punkte .....	15
4.2. Detailstufen .....	18
4.3. Eye-Dome-Lighting .....	20
<b>IV. Implementierung .....</b>	<b>22</b>
1. Technik .....	22
2. Benutzung .....	23
2.1. Installation .....	23
2.2. Import .....	24
2.3. Visualisierung .....	24
3. Import .....	25
3.1. Parallelisierung .....	26
4. Punkte .....	27
5. Segmentierung .....	28
5.1. Auswahl .....	29
5.2. Visualisierung .....	29
5.3. Exportieren .....	29
6. Detailstufen .....	29
<b>V. Auswertung .....</b>	<b>31</b>
1. Testdaten .....	31

2. Import .....	31
3. Segmentierung in Bäume .....	32
4. Analyse von Segmenten .....	33
5. Triangulierung .....	33
6. Visualisierung .....	34
7. Fazit .....	34
8. Diskussion .....	34
9. Ausblick .....	34
<b>VI. Appendix .....</b>	<b>35</b>
1. KD-Baum .....	35
1.1. Konstruktion .....	35
1.2. Suche mit festem Radius .....	36
1.3. Suche mit fester Anzahl .....	36
1.4. Schnelle Suche .....	36
2. Baum (Datenstruktur) .....	37
2.1. Konstruktion .....	37
2.2. Suchanfrage .....	37
3. Messwerte vom Import .....	38
4. Punktwolkenformat .....	40
4.1. Daten .....	40
5. Quelltextstruktur .....	41
<b>Glossar .....</b>	<b>42</b>
<b>Bibliographie .....</b>	<b>43</b>

# I. Einleitung

To-do: Mit Bild(ern) auflockern.

## 1. Motivation

Größere Gebiete wie Teile von Wäldern können als 3D-Punktwolken gescannt werden, aber relevante Informationen sind nicht direkt aus den Daten 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. In dieser Arbeit wird mit den Daten eine interaktive Visualisierung ermöglicht, welche eine manuelle Auswertung ermöglicht.

## 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. Als Eingabe wird der Datensatz vom Waldstücke benötigt. Dabei enthält ein Datensatz eine ungeordnete Menge von Punkten, für die nur die Position im dreidimensionalen Raum bekannt ist.

Für die Analyse der Daten muss die Menge der Punkte in einzelne Bäume segmentiert werden, dass Punkte vom gleichen Baum zum gleichen 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 die charakteristischen Eigenschaften für die Bäume, aber auch für jeden Punkt 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 interaktiv inspiert werden.

## 3. Struktur der Arbeit

In [Abschnitt III](#) wird die Methodik für die Analyse der Daten erklärt. Dazu gehört die Segmentierung in einzelnen Bäume, Analyse und Triangulierung dieser und die technischen Grundlagen für die Visualisierung der Ergebnisse.

Danach wird in [Abschnitt IV](#) die Implementierung der Methoden ausgeführt. Dazu gehört die konkrete technische Umsetzung der Algorithmen. Dazu gehört die Benutzung

von Softwareprojekt, der Ablauf vom Import, das Anzeigen der Segmente und einzelner Punkte

Die Auswertung wird in [Abschnitt V](#) durchgeführt. Dafür werden die benutzen Testdaten vorgestellt und die Ergebnisse der Implementierung analysiert.

## II. Stand der Technik

To-do: Stand der Technik

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 Punktewolke in Bäume

#### 1.1. Ablauf

Die Punkte werden 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 Bereichen zusammengefasst. Für die Bereiche werden die zugehörigen Koordinaten der Bäume bestimmt und jeder Punkte wird zur nächsten Koordinate 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.

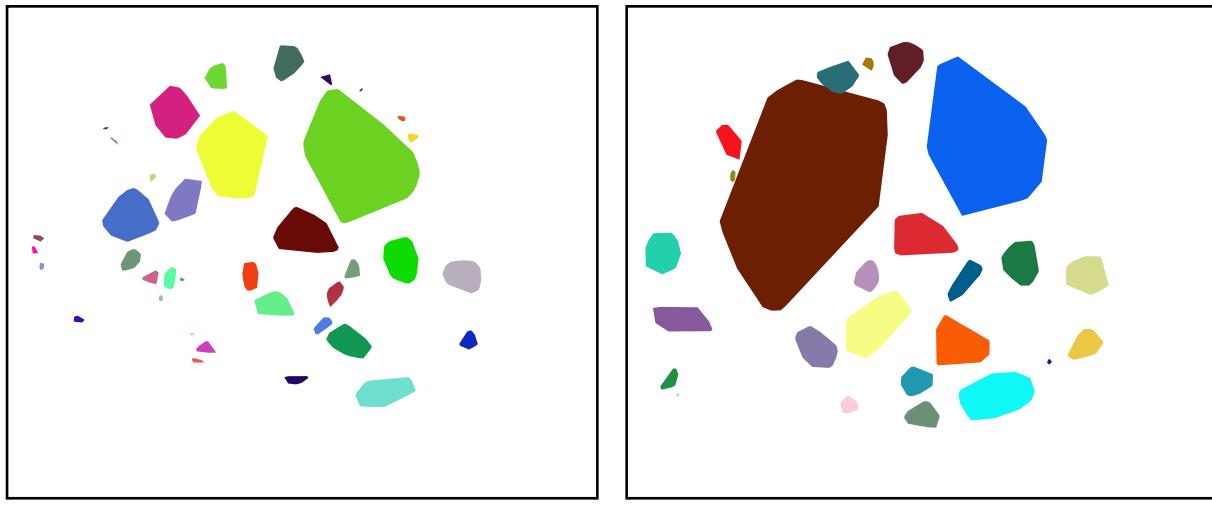


Abbildung 1: Beispiel für berechnete Segmente für zwei Scheiben. Größere Bereiche gehören zu mehreren Bäumen.

Die Bereiche sind als Liste gespeichert, wobei für jeden Bereich die Eckpunkte als Liste gegeben sind. Die Eckpunkte sind dabei sortiert, dass für einen Eckpunkt der nächste Punkt entlang der Umrandung 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 Umrundung liegt. Für den Punkt  $p$  kann nun der Abstand zur Kante mit dem Skalarprodukt  $o \cdot (p - a)$  berechnet werden. Wenn der Punkte auf der Innenseite der Kante liegt, ist der Abstand negativ.

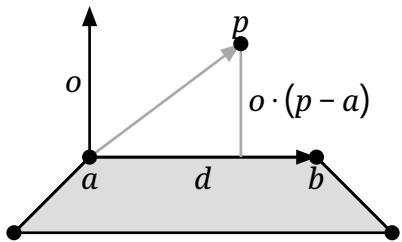


Abbildung 2: Berechnung vom Abstand vom Punkt  $p$  zur Kante zwischen  $a$  und  $b$ .

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 beiden Eckpunkte gesucht, bei denen eine zugehörige Kante entfernt wird und die andere nicht. Um die Kanten zwischen den Punkten zu entfernen, werden alle Punkte zwischen den beiden Punkten entfernt und stattdessen der neue Punkt eingefügt, um die beiden neuen Kanten zu ergänzen.

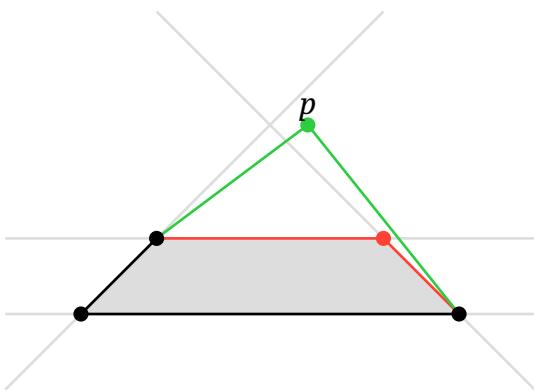


Abbildung 3: Hinzufügen vom Punkt  $p$  zum Bereich. Die Kanten in Rot werden entfernt und die Kanten in Grün werden hinzugefügt.

Nachdem alle Punkte zu den Bereichen hinzugefügt würden, werden kleine Bereiche entfernt. 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 4 unterteilt werden und dann die Flächen der Dreiecke summiert werden.

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 Hinzufügen 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 die bekannten Koordinaten bestimmt, welche im Bereich liegen.

Wenn keine Koordinate in dem Bereich liegt, so fängt der Bereich ein neues Segment an. Als Koordinate wird der geometrische Schwerpunkt vom Bereich verwendet. Liegen vorherige Koordinaten im Bereich, wird kein neues Segment angefangen. Liegt aber genau eine Koordinate im Bereich, wird die Koordinate mit dem Schwerpunkt vom Bereich aktualisiert.

Für die Berechnung vom Schwerpunkt wird der Bereich in Dreiecke unterteilt und der gewichtete Durchschnitt der Schwerpunkte der Dreiecke berechnet. Weil der Bereich konvex ist, kann ein beliebiger Punkt ausgewählt werden und alle Dreiecke mit dem Punkt und den zwei Punkten von einer Kante ohne den Punkt, bilden ein Dreieck. Das Gewicht für ein Dreieck ist der relative Anteil der Fläche vom Dreieck zur Gesamtfläche vom Bereich. Ein Beispiel ist in Abbildung 4 gegeben.

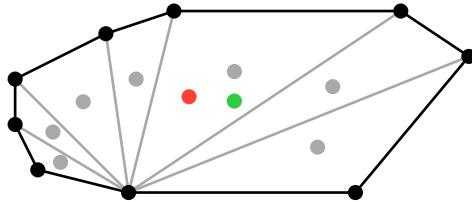
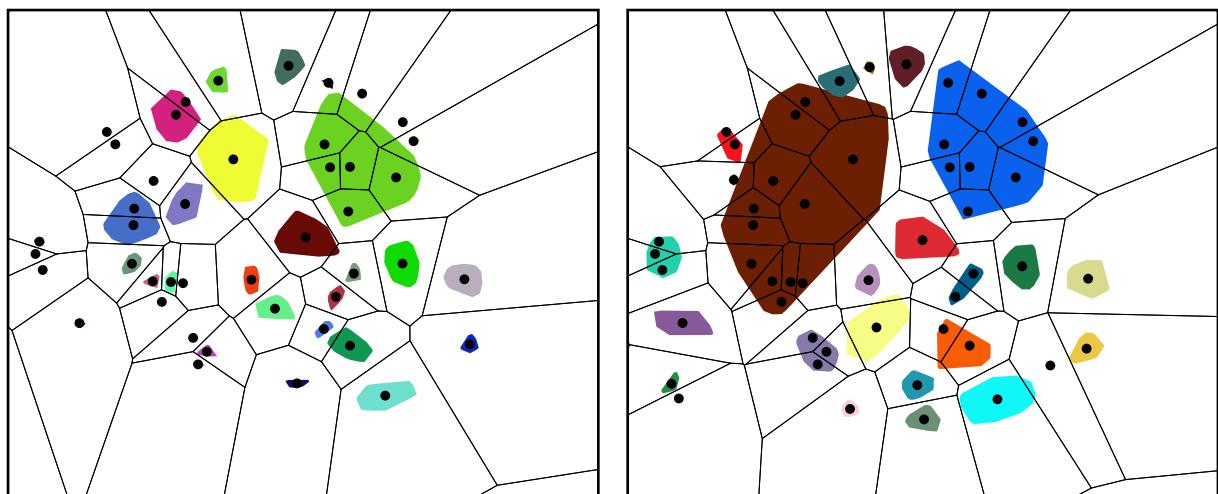


Abbildung 4: Unterteilung von einem Bereich in Dreiecke. Die Schwerpunkte der Dreiecke sind in Grau und vom gesamten Bereich in Grün. Die durchschnittliche Position der Eckpunkte ist in Rot.

### 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 5 zu sehen.



(a) höhere Scheibe

(b) tiefere Scheibe

Abbildung 5: Berechnete Koordinaten für die Punkte mit zugehörigen Bereichen und Voronoi-Diagramm.

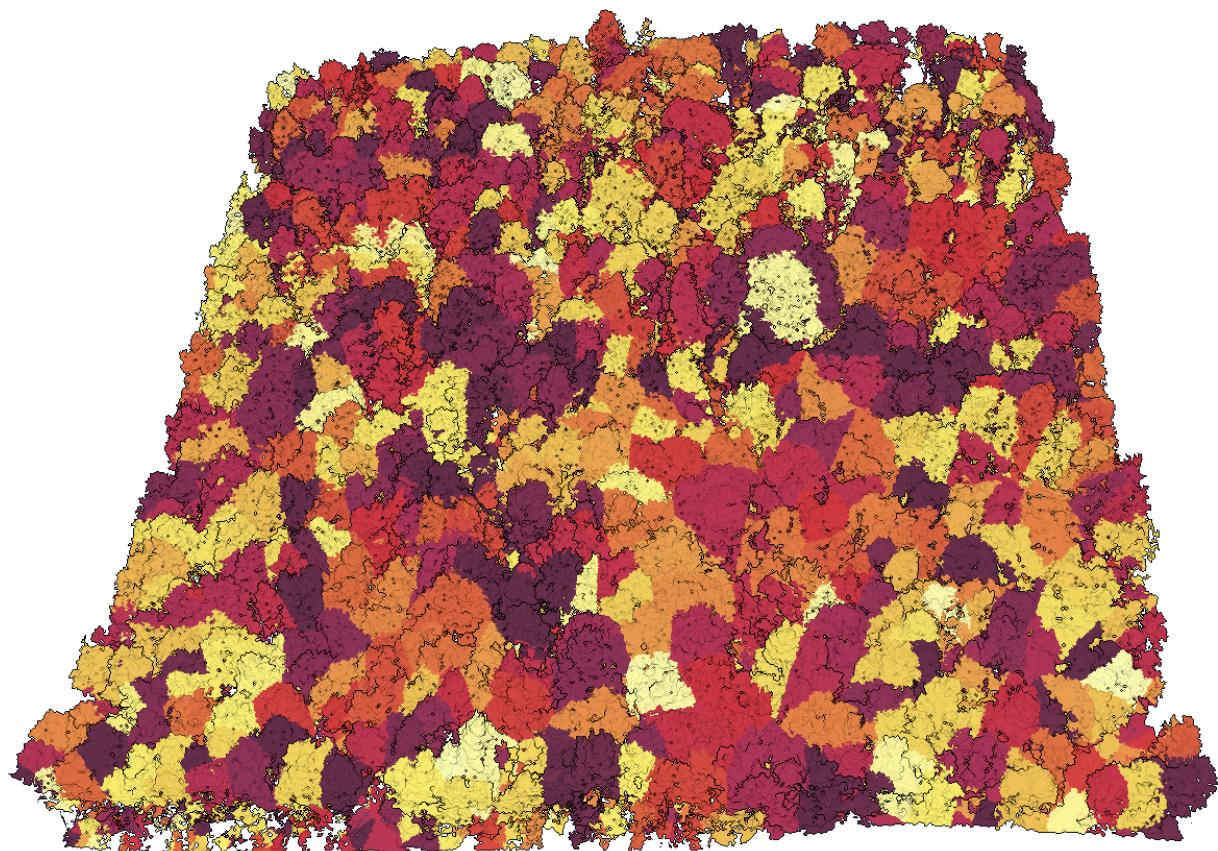


Abbildung 6: Segmentierung von einem Waldstück.

## 2. Analyse von Segmenten

## 2.1. Eigenschaften

Die Baumeigenschaften werden für jedes Segment einzeln berechnet. Dabei sind alle Punkte im Segment verfügbar.

### 2.1.1. Nachbarschaft

Um relevante Eigenschaften für einen Punkt zu bestimmen, werden die umliegenden Punkte benötigt. Dafür wird für alle Punkte ein **KD-Baum** erstellt. Mit diesem können effizient für einen Punkt die  $k$ -nächsten Punkte bestimmt werden.

### 2.1.2. Krümmung

Die Krümmung der Oberfläche wird für jeden Punkt geschätzt. Dafür werden die Positionen der Punkte in der Nachbarschaft betrachtet. Zuerst wird der geometrische Schwerpunkt bestimmt, dass die Positionen der Punkte um diesen verschoben werden können. Ohne die Verschiebung würde die globale Position der Punkte das Ergebnis verfälschen. Mit den Positionen der Punkte kann die Kovarianzmatrix bestimmt werden.

Die Eigenvektoren der Kovarianzmatrix bilden eine Orthonormalbasis und die Eigenwerte geben die Ausdehnung entlang des zugehörigen Basisvektors an. Der kleinste Eigenwert gehört zu Dimension mit der geringsten Ausdehnung. Je kleiner der Eigenwert, desto näher liegen die Punkte in der Nachbarschaft an der Ebene, aufgespannt durch die Eigenvektoren zugehörig zu den größeren Eigenwerten.

Wenn die Eigenwerte  $\lambda_i$  mit  $i \in \mathbb{N}_0^2$  absteigend nach größer sortiert sind, dann kann die Krümmung  $c$  mit  $c = \frac{3\lambda_2}{\lambda_0 + \lambda_1 + \lambda_2}$  berechnet werden.  $c$  liegt dabei im abgeschlossenen Bereich  $[0; 1]$ .



Abbildung 7: Punktwolke mit Krümmung markiert.

### 2.1.3. Punkthöhe

Für jeden Punkt wird die relative Höhe im Segment bestimmt. Dafür wird die Mindesthöhe  $y_{\min}$  und die Maximalhöhe  $y_{\max}$  im Segment benötigt. Die relative Höhe  $h$  für den Punkt mit der Höhe  $p_y$  kann mit  $h = \frac{p_y - y_{\min}}{y_{\max} - y_{\min}}$  berechnet werden. Die relative Höhe liegt dabei im Bereich  $[0; 1]$ .

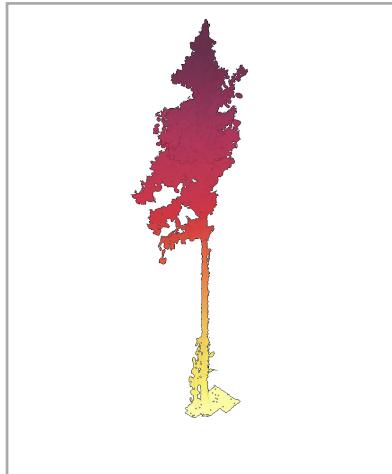
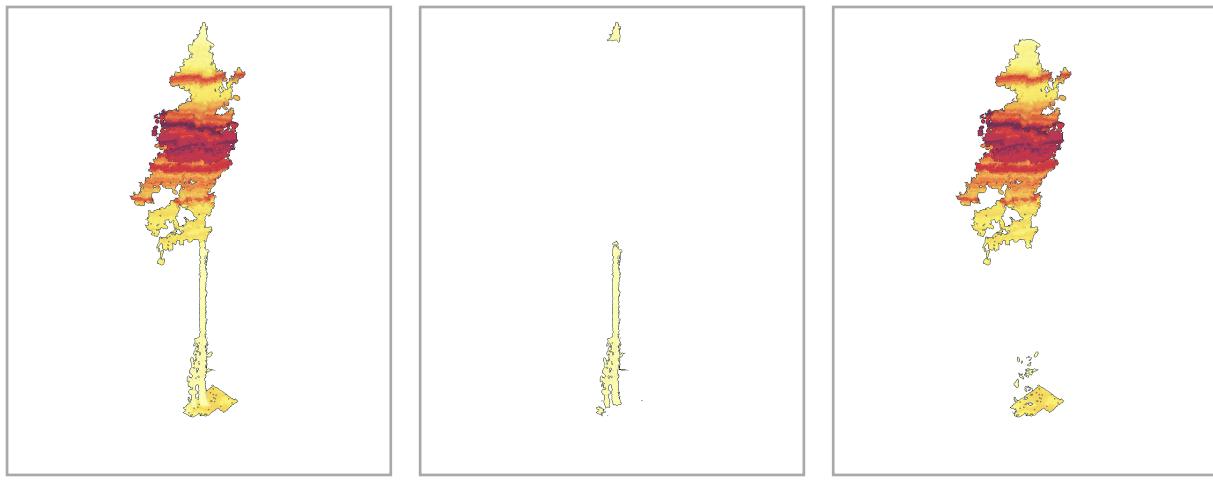


Abbildung 8: Punktwolke mit Höhe markiert.

#### 2.1.4. Ausdehnung

Der Baum wird entlang der Horizontalen in gleich hohe Scheiben unterteilt. Die Breite der Scheiben ist dabei einstellbar. Die Ausdehnung wird für jede Scheibe berechnet. Zuerst wird der geometrische Schwerpunkt der Positionen berechnet, womit die durchschnittliche Standardabweichung entlang der Horizontalen bestimmt wird.

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.



(a) Alle Punkte

(b) Geringe Ausdehnung

(c) Hohe Ausdehnung

Abbildung 9: Punktwolke mit Varianz markiert.

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.

To-do: Mehr Baumeigenschaften

## 2.2. Segmentierung von einem Baum

<https://besjournals.onlinelibrary.wiley.com/doi/10.1111/2041-210X.13144>

To-do: Baumeigenschaften + ? → Segmente

## 2.3. Eigenschaften für Visualisierung

### 2.3.1. Normale

Mit den Eigenvektoren aus [Abschnitt III-2.1.2](#) wird die Normale für die Umgebung bestimmt. Der Eigenvektor, welcher zum kleinsten Eigenwert gehört, ist orthogonal zur Ebene mit der größten Ausdehnung.

### 2.3.2. Punktgröße

Für die Punktgröße wird der durchschnittliche Abstand zu den umliegenden Punkten bestimmt.

## 2.4. Baumart

To-do: Segmente + Eigenschaften + ? → Klassifizierung?

- out of scope?
- neural?

## 3. Triangulierung

### 3.1. Ziel

Eine Triangulierung ermöglicht eine Rekonstruktion der ursprünglichen Oberfläche vom eingescannten Bereich, welche weiterverarbeitet oder anzuzeigen werden kann. Die meisten Programme und Hardware sind auf das Anzeigen von Dreiecken spezialisiert, und können diese effizienter als Punkte darstellen. Die Triangulierung wird dabei für die Segmente getrennt 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  $\alpha$  ( $\alpha$ -Kugel) erreicht werden können. Dabei berührt die Kugel die drei Eckpunkte vom Dreieck und kein weiterer Punkt aus der Punktwolke liegt in der Kugel.

In Abbildung 10 ist ein Beispiel in 2D gegeben. Dabei werden die Linien gesucht, dass der zugehörige Kreis keine weiteren Punkte enthält.

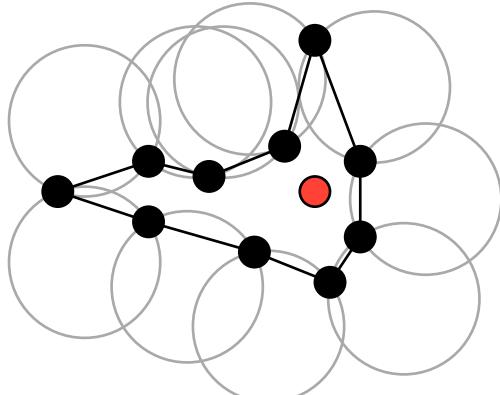


Abbildung 10: Ball-Pivoting-Algorithmus in 2D. Für die äußeren Punkte in Schwarz wird eine Oberfläche gefunden. Für den inneren Punkt in Rot kann kein Nachbar gefunden werden, weil alle zugehörigen Kreise weitere Punkte enthalten würden.

Die gefundenen Dreiecke bilden eine Hülle, welche alle Punkte beinhaltet. Je kleiner  $\alpha$  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. $\alpha$ -Kugel für ein Dreieck

Für ein Dreieck  $(p_1, p_2, P - 3)$  wird die Position der zugehörigen  $\alpha$ -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  $\alpha$ , so gibt es keine zugehörige  $\alpha$ -Kugel. Für Abstände kleiner gleich  $\alpha$  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  $\alpha$ -Kugel mit  $c + d \cdot \frac{o}{|o|}$  berechnet werden kann.

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

### 3.2.3. Ablauf

#### 3.2.4. Startdreieck bestimmen

Als Anfang wird ein Dreieck mit zugehöriger  $\alpha$ -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  $2\alpha$  oder weniger bestimmt. Für weiter entfernte Punkte gibt es keine  $\alpha$ -Kugel, welche beide Punkte berühren würde.

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

Wenn ein Dreieck mit zugehöriger  $\alpha$ -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.

### 3.2.5. 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  $\alpha$ -Kugel mit Zentrum  $c$  bekannt. Die Kante dient als Pivot, um welches die  $\alpha$ -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 11 ist ein Beispiel für die Erweiterung in 2D gegeben. Es werden Kanten als Oberfläche gesucht und Punkte werden als Pivot-Element verwendet.

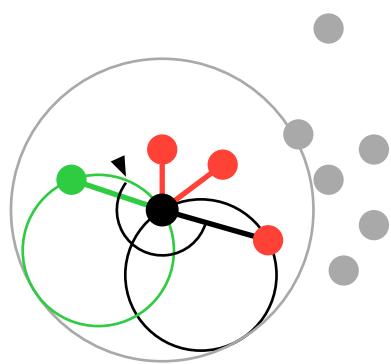


Abbildung 11: Erweiterung der gefundenen Oberfläche in 2D. Die vorherige Kante und der momentane Pivot-Punkt sind in Schwarz. Der  $\alpha$ -Kreis rollt entlang der markierten Richtung. In Grün ist der erste Punkt und zugehörigen Kreis, welche berührt werden. Die weiteren Punkte in Rot sind Kandidaten, liegen aber weiter in der Rotation, weshalb die grüne Kante zur Oberfläche hinzugefügt wird.

#### 3.2.5.1. 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  $d = |p_{1,2} - mp|$  berechnet. Der Abstand zwischen dem Zentrum der Kugel und den Endpunkten von der Kante ist immer  $\alpha$ , dadurch ist der Abstand vom Zentrum zum Mittelpunkt  $x = \sqrt{\alpha^2 - d^2}$ . In Abbildung 12 ist die Berechnung veranschaulicht.

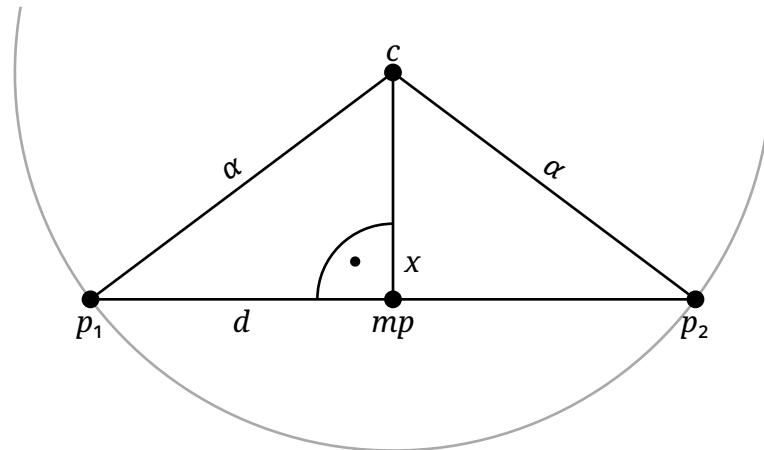


Abbildung 12: Berechnung des Abstands der  $\alpha$ -Kugel vom Mittelpunkt der Kante

Die möglichen Punkte sind vom Zentrum der Kugel  $c$  maximal  $\alpha$  entfernt und  $c$  ist vom Mittelpunkt  $mp$   $x$  entfernt. Deshalb werden die Punkte in der Kugel mit Zentrum  $mp$  und Radius  $\alpha + x$  bestimmt.

### 3.2.5.2. 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  $\alpha$ -Kugel bestimmt, welche  $p_1$ ,  $p_2$  und  $p$  berührt. Die Kugel wird dabei wie in Abbildung 13 bestimmt, dass die Kugel auf der korrekten Seite vom potenziellen Dreieck liegt.  $p$  kann so liegen, dass es keine zugehörige  $\alpha$ -Kugel gibt, in diesem Fall wird  $p$  nicht weiter betrachtet. Für jeden Kandidaten wird der Winkel  $\varphi$  berechnet, wie weit um die Kante die Kugel gerollt wurde.

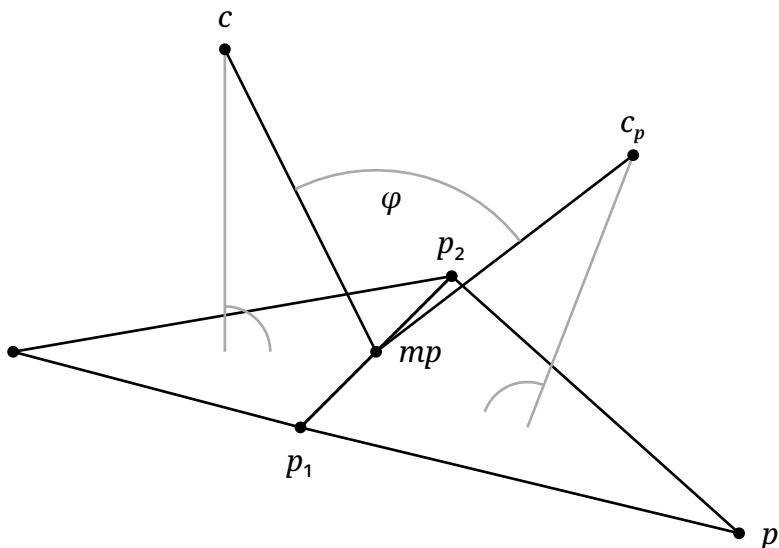


Abbildung 13: Berechnung von Zentrum der  $\alpha$ -Kugel und zugehöriger Winkel für einen Kandidatenpunkt

Mit  $mp$ ,  $c$  und  $c_p$  wird der Winkel  $\varphi$  bestimmt. Dafür werden die Vektoren  $a = c - mp$  und  $b = c_p - mp$  bestimmt und normalisiert. Der Kosinus von  $\varphi$  ist dabei das Skalarprodukt  $s = a \cdot b$ . Zusätzlich wird das Kreuzprodukt  $k = a \times b$  bestimmt, womit

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

berechnet wird. Von allen Kandidaten wird der Punkt  $p_3$  ausgewählt, für den  $\varphi$  am kleinsten ist.

Es muss nicht kontrolliert werden, ob ein Punkt in der  $\alpha$ -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  $\varphi$  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, kann das nicht sein. Dies gilt aber nur, wenn die Kugel zum Start bereits leer ist.

### 3.2.5.3. 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 Kanten zwischen  $(p_1, p_3)$  und  $(p_3, p_2)$  hinzugefügt. Wenn eine neue Kante in den äußeren Kanten bereits vorhanden ist, wird diese nicht hinzugefügt, sondern entfernt, weil das zugehörige zweite Dreieck bereits gefunden wurde. Ein Veranschaulichung ist in Abbildung 14 gegeben.

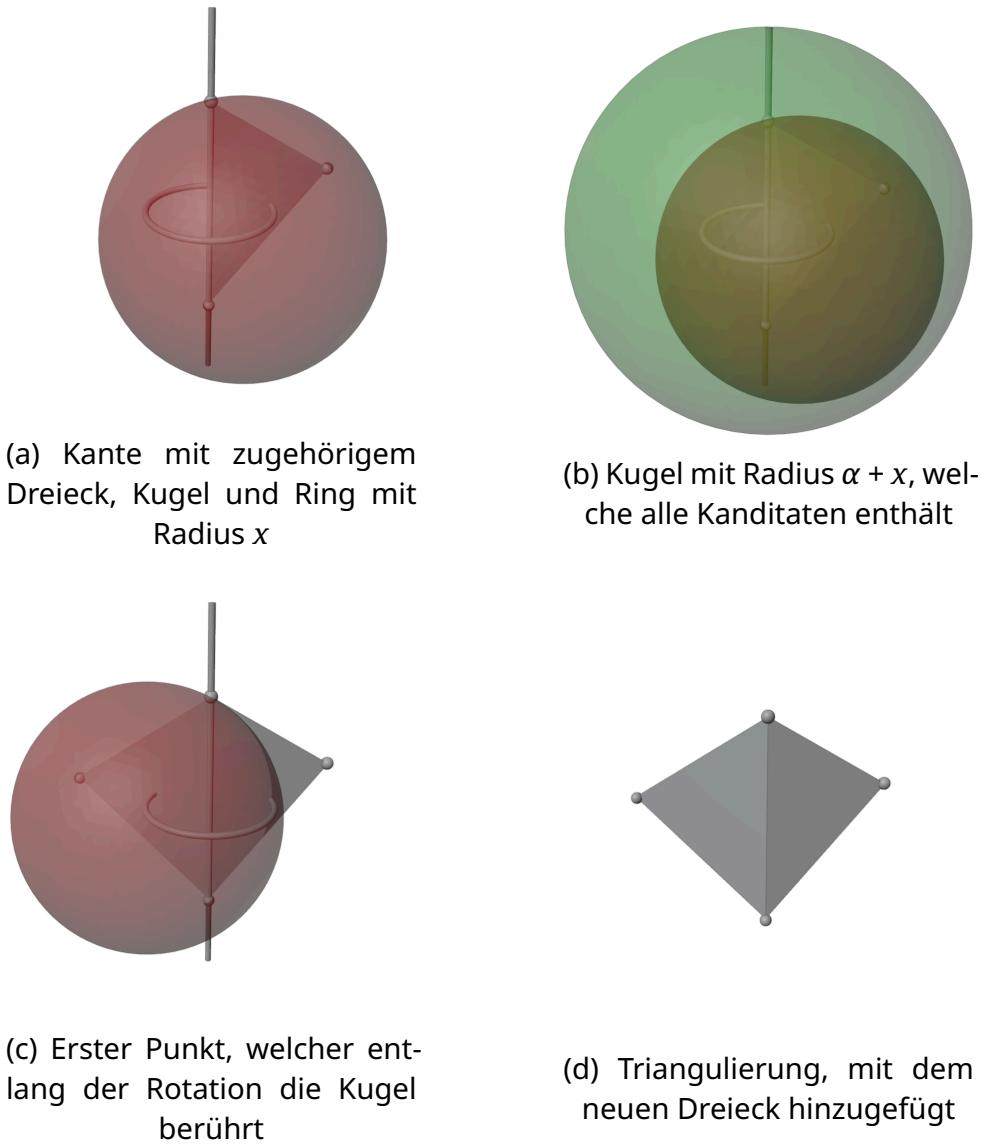


Abbildung 14: Erweiterung der Triangulierung in 3D.

### 3.2.6. 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 malformierten Daten 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 die-

se noch nicht in den inneren Kanten vorhanden sind. Bei der Erweiterung wird die ausgewählte 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.

### 3.2.7. Vorauswahl

Vor der Triangulierung wird mit dem Mindestabstand  $d$  die Anzahl der Punkte und deformierte Dreiecke verringert. Dafür wird ein Subset 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 zum Set hinzugefügt und danach werden alle Punkte im Set 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 dem Set entfernt und der nächste Punkt, der noch im Set ist, wird betrachtet.

### 3.2.8. Auswahl von $\alpha$

- Groß genug für keine Lücken
- Klein genug für gute Laufzeit

To-do: Auswahl  $\alpha$

## 4. Visualisierung

### 4.1. Punkte

Grafikpipelines können Punkte, Linien oder Dreiecke rendern. Ein Punkt wird dabei mit genau einem Pixel angezeigt, dadurch können Datenpunkte unterschiedlich weit von der Kamera entfernt nicht unterschiedlich groß angezeigt werden. Um einen Datenpunkt anzuzeigen, wird deshalb ein ausgefüllter Kreis verwendet.

Um einen Kreis zu rendern, kann ein beliebiges Polygon gerendert werden, solange der Einheitskreis mit Zentrum  $(0, 0)$  vollständig enthalten ist. Die Bereiche, 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

Das kleinste passende Dreieck ist ein gleichseitiges Dreieck. In Abbildung 15 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.

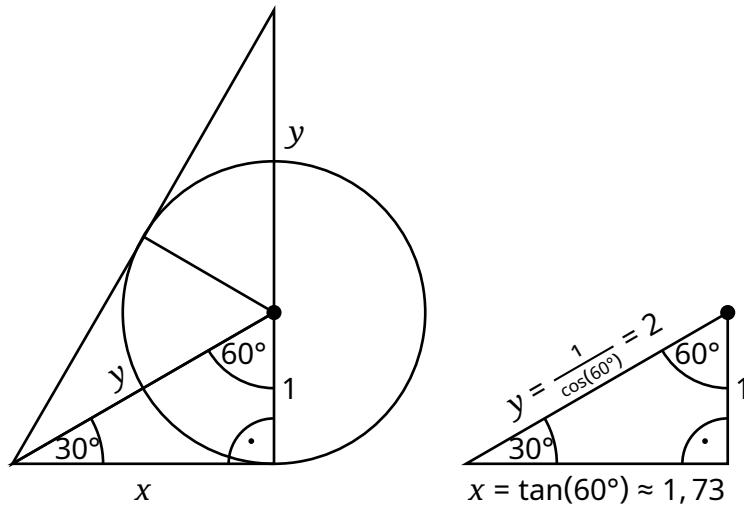


Abbildung 15: Seitenlänge für das kleinste gleichseitige Dreieck, welches den Einheitskreis enthält.

Das kleinste mögliche Viereck ist das Quadrat mit Seitenlänge 2. In Abbildung 16 ist die Konstruktion gegeben. Um diesen anzulegen, werden zwei Dreiecke benötigt. Für die beiden Dreiecke werden dadurch sechs Ecken und eine Fläche von  $a^2 = 2^2 = 4$  benötigt.

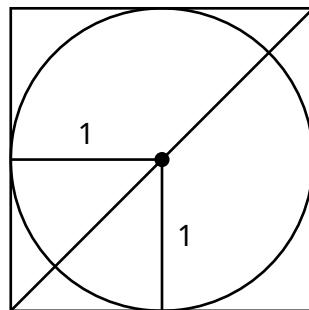


Abbildung 16: Seitenlänge für das kleinste Quadrat, welches den Einheitskreis enthält.

Für Polygone mit mehr Ecken, wird der benötigte Bereich kleiner, es werden aber auch mehr Ecken benötigt.

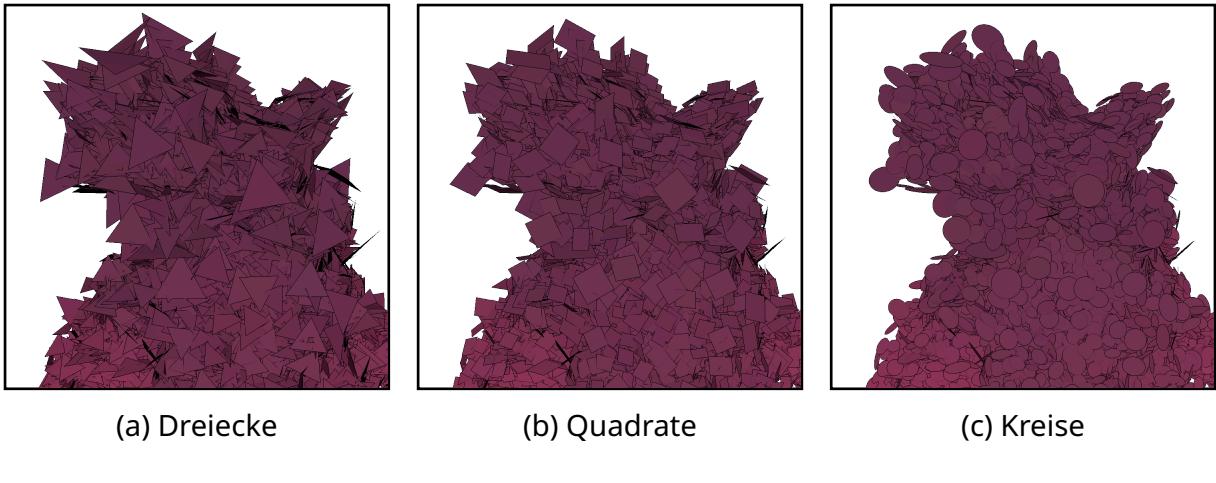


Abbildung 17: Unterschiedliche Polygone und Kreise für das Anzeigen der Punkte.

#### 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 18 gegeben.

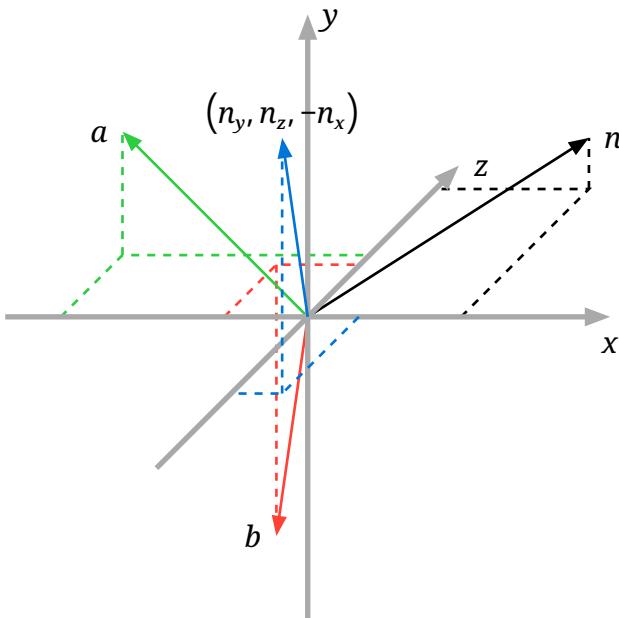


Abbildung 18: Beispiel für die Berechnung von  $a$  und  $b$  paarweise orthogonal zu  $n$ .

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 19 ist die Berechnung dargestellt.

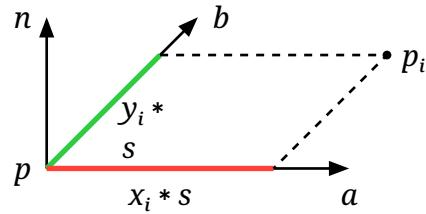


Abbildung 19: Berechnung von einem Eckpunkt.

#### 4.2. Detailstufen

Je nach Scannertechnologie 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 entfernte Punkte ist es nicht notwendig, alle Punkte genau wiederzugeben. Deshalb wird für weit entfernte Punkte eine vereinfachte Version angezeigt. Diese besteht aus weniger Punkten und benötigt dadurch weniger Ressourcen, bietet aber eine gute Approximation der ursprünglichen Daten.

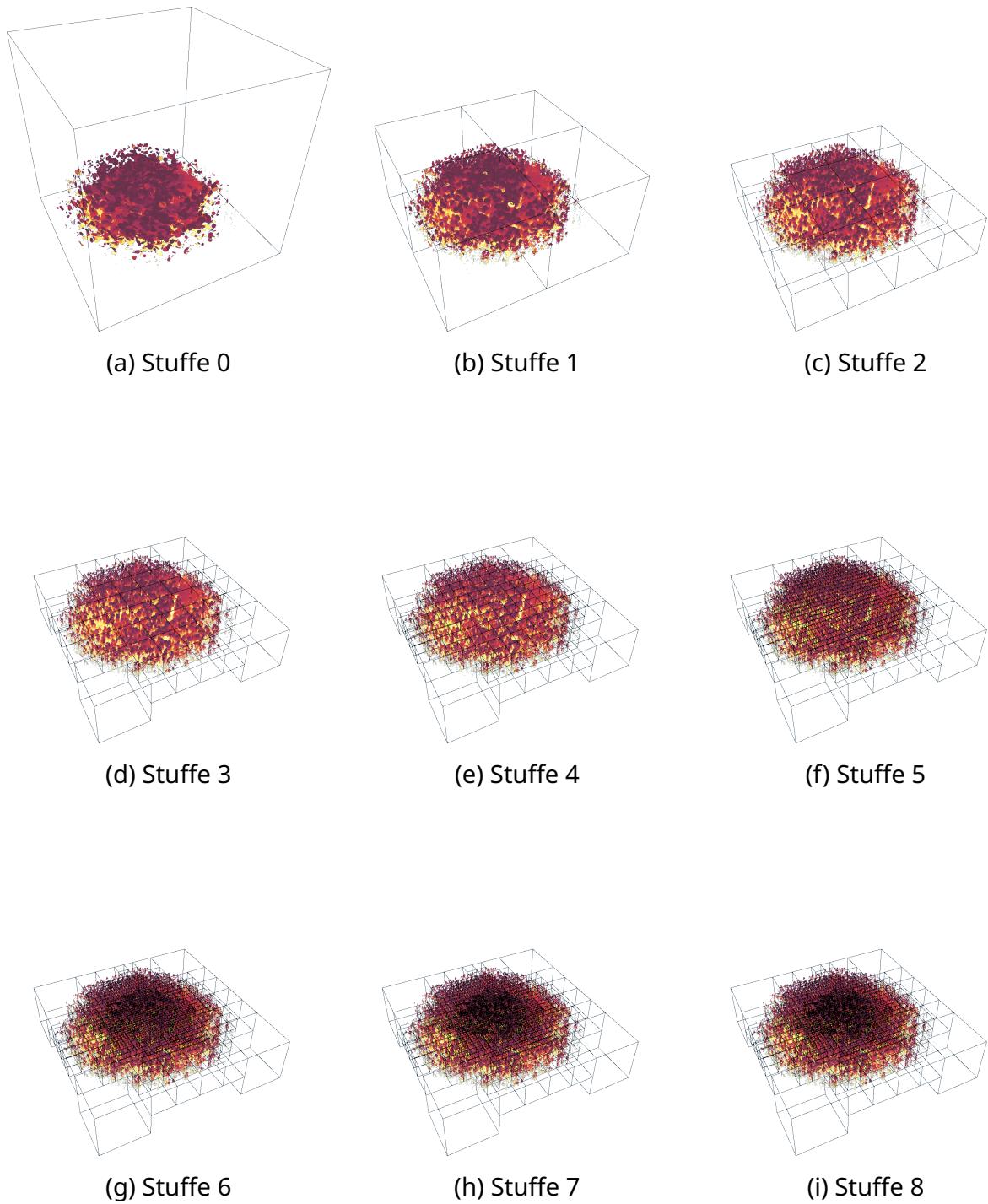


Abbildung 20: Unterschiedliche Stufen der Unterteilung. Jeder Würfel enthält bis zu 32768 Punkte. In der höchsten Stufe werden alle Punkte im Datensatz angezeigt.

Für die gesamte Punktewolke wird ein Octree mit den Punkten erstellt. Der zugehörige Voxel vom Root-Knoten wird so gewählt, dass alle Punkte im Voxel liegen. Rekursiv wird der Voxel in acht gleich große Voxel geteilt, solange in einem Voxel noch zu viele Punkte liegen. Nach der Unterteilung gehört jeder Punkt im Datensatz zu genau einem Leaf-Knoten.

Für jeden Branch-Knoten wird eine Punktwolke berechnet, welche als Vereinfachung der Punkte der zugehörigen Kinderknoten verwendet werden kann.

#### 4.2.1. Berechnung der Detailstufen

Die Detailstufen werden wie bei „Fast Out-of-Core Octree Generation for Massive Point Clouds“ [8] von den Blättern des Baumes bis zur Wurzel 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 nach oben beschränkt. 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 wird ein repräsentativer Punkt bestimmt. Dafür werden zuerst alle Punkte aus den Kinderknoten bestimmt, welche im Teilvoxel liegen. Liegt kein Punkt im Teilvoxel, so wird dieser übersprungen. Aus den Punkten wird ein repräsentativer Punkt für den ganzen Voxel bestimmt. Dafür werden Position, Normale und Größe gemittelt und die Eigenschaften von einem der Punkte übernommen.

Bei der nächsten größeren Detailstufe sind auch die zugehörigen Voxel von den Knoten größer. Durch die feste Anzahl der Teilvoxel sind diese deshalb größer, 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 Tiefenunterschied zu den umliegenden Pixeln bestimmt. Das Tiefenbild für die Veranschaulichung ist in Abbildung 21 gegeben.

Je größer der Unterschied ist, desto stärker wird der Pixel im Ergebnisbild eingefärbt. Dadurch werden Kanten hervorgehoben, je nachdem wie groß der Tiefenunterschied ist.

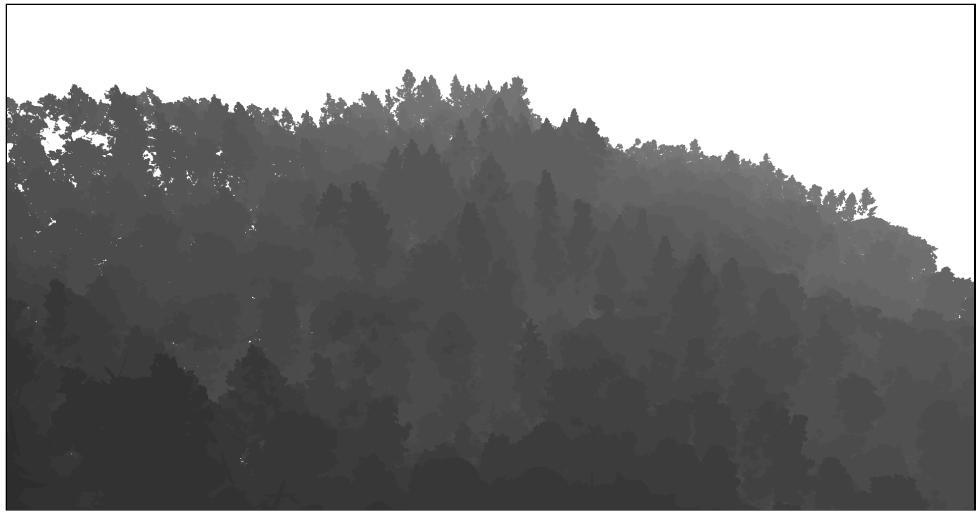


Abbildung 21: Tiefenbild nach dem Rendern der Szene. Je heller eine Position ist, desto weiter ist das Polygon zugehörig zur Koordinate von der Kamera entfernt.

Der Effekt entsteht dadurch, dass für jedes Pixel der maximale Tiefenunterschied zu den umliegenden Pixeln bestimmt wird. Je größer der Unterschied, desto mehr wird das zugehörige Pixel verdunkelt. Ein Veranschaulichung ist in Abbildung 22 gegeben.

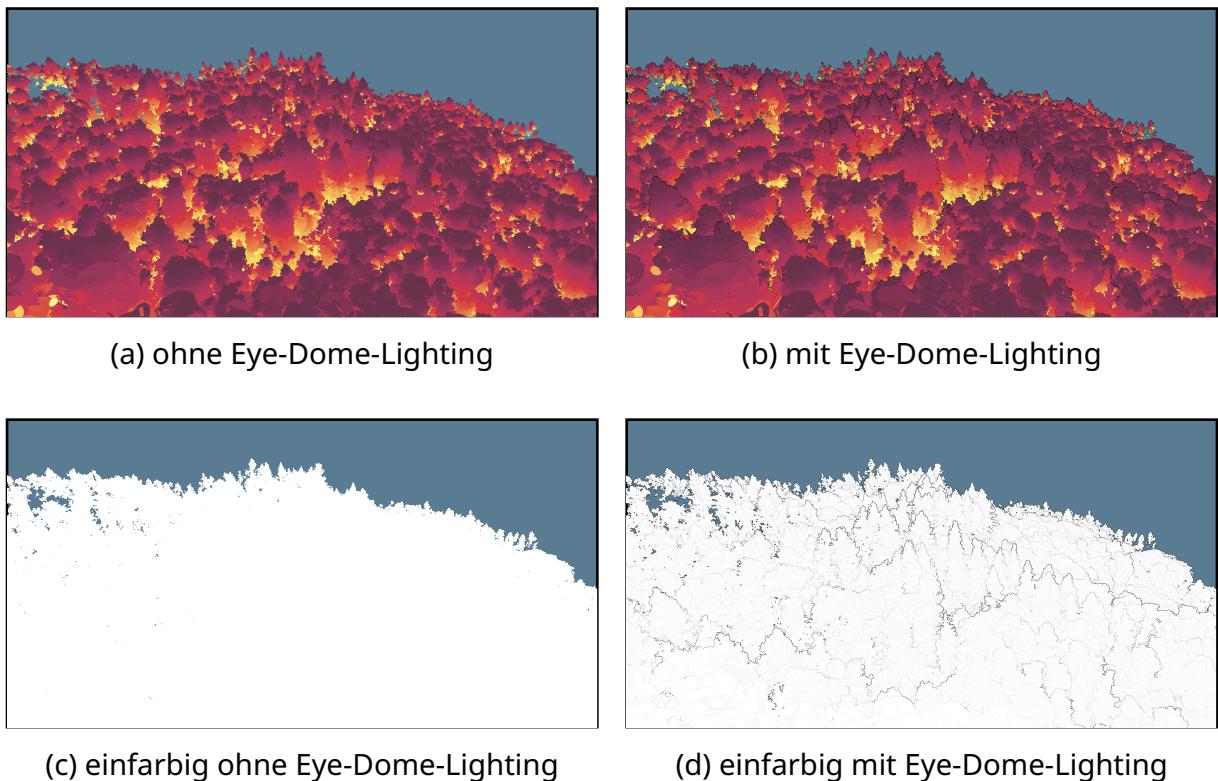


Abbildung 22: Waldstück mit und ohne Eye-Dome-Lighting. Die Punkte sind zusätzlich in Weiß angezeigt, um den Effekt hervorzuheben.

## **IV. Implementierung**

### **1. Technik**

Das Projekt ist unter <https://github.com/antonWetzel/treee> 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
pollster	0.3	Auf asynchrone Ergebnisse warten
rfd	0.13	Dialogfenster zum Öffnen und Speichern von Dateien
crossbeam	0.8	Synchronisierung zwischen Threads
log	0.4	Logs erzeugen
simple_logger	4.3	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
bincode	1.3.3	Serialisierung als Binary
serde_json	1.0.113	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

Tabelle 1: Benutzte Bibliotheken

Als Datensätze werden Dateien im LASzip-Format verwendet. Dieses Format wird häufig für Punktwolken verwendet. Weitere Formate können einfach eingebunden werden, solange 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 kön-

nen von [To-do: GitHub-Release](#) heruntergeladen werden. Die Schritte zum selber kompilieren sind im [Readme](#)<sup>1</sup> verfügbar.

## 2.2. Import

To-do: Benutzung aus dem Readme hier?

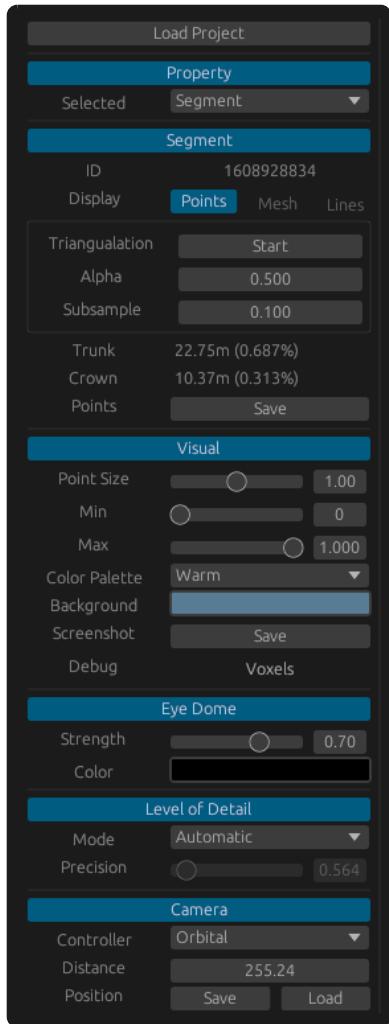
## 2.3. Visualisierung

Bei der Visualisierung wird eine importierte Punktwolke geöffnet. Die Punktwolke besteht dabei aus der Struktur vom Octree, Informationen über die Segmente und die Punktdaten. Die Daten für die einzelnen Punkte werden zuerst nicht geladen. In Abbildung 23 ist das Benutzerinterface erklärt.

Je nach Position der Kamera werden die benötigten Punkte geladen, welche momentan sichtbar sind. Dadurch können auch Punktwolken angezeigt werden, die mehr Punkte enthalten als gleichzeitig interaktiv anzeigbar. Auch bei den Segmenten wird nur das Segment geladen, welches ausgewählt wurde.

---

<sup>1</sup><https://github.com/antonWetzel/treee?tab=readme-ov-file#treee>



- **Load Project**
  - Die geladene Punktwolke ändern
- **Property**
  - Eigenschaft zum Anzeigen ändern
- **Segment**
  - Informationen über das ausgewählte Segment
  - Triangulation starten und anzeigen
  - Punkte speichern
- **Visual**
  - Punktegröße ändern
  - Punkte basierend auf der ausgewählten Eigenschaft filtern
  - Farbpalette und Hintergrund ändern
  - Screenshot speichern
  - Knoten für Detailstufen anzeigen
- **Eye Dome**
  - Farbe und Stärke vom Eye-Dome-Lighting ändern
- **Level of Detail**
  - Algorithmus und Qualität der Detailstufen anpassen
- **Camera**
  - Bewegung der Kamera ändern
  - Kameraposition speichern

Abbildung 23: Benutzerinterface mit allen Optionen.

### 3. 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 die Ergebnisse speichern und zum Octree hinzufügen
4. Detailstufen bestimmten und Octree speichern

Der zugehörige Datenfluss ist in Abbildung 24 zu sehen. Nach der ersten Phase sind die Segmente bekannt, nach der zweiten Phase analysiert und zum Octree hinzugefügt. Die Struktur von der Punktwolke ist bereits bekannt. Nach der dritten Phase sind auch die Detailstufen vom Octree erstellt.

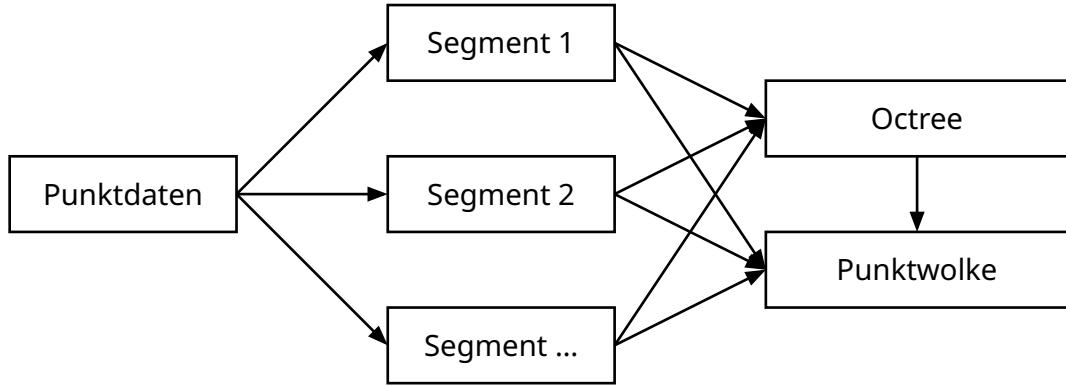
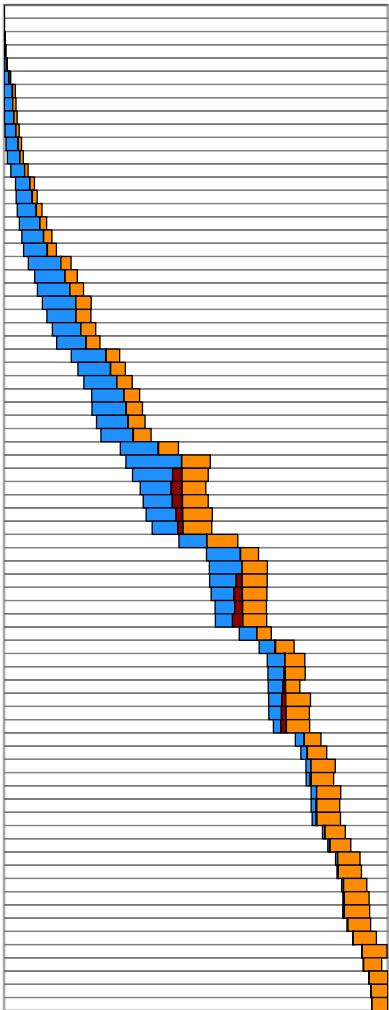


Abbildung 24: Datenfluss für den Import.

### 3.1. 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 25 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 der Bäume berechnen
3. Jeder Punkte zur nächsten Koordinate zugeordnet

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 diese 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 gestartet werden, wenn von der vorherigen Scheibe die zweite Stufe beendet ist. Wenn die erste Stufe länger dauert, müssen deshalb andere Threads warten.

Abbildung 25: 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 alle Kinderknoten parallel verarbeitet werden.

## 4. 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 26 sind die getesteten Varianten gegeben. Die Dreiecke werden dann 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 untransformierten Koordinaten 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.

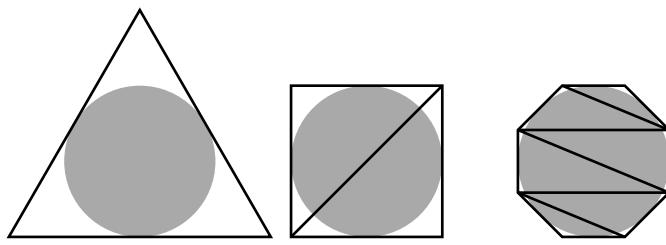


Abbildung 26: Zerlegung von unterschiedlichen Polygone in Dreiecke.

In Abbildung 27 sind die Renderzeiten für unterschiedliche Polygone 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.

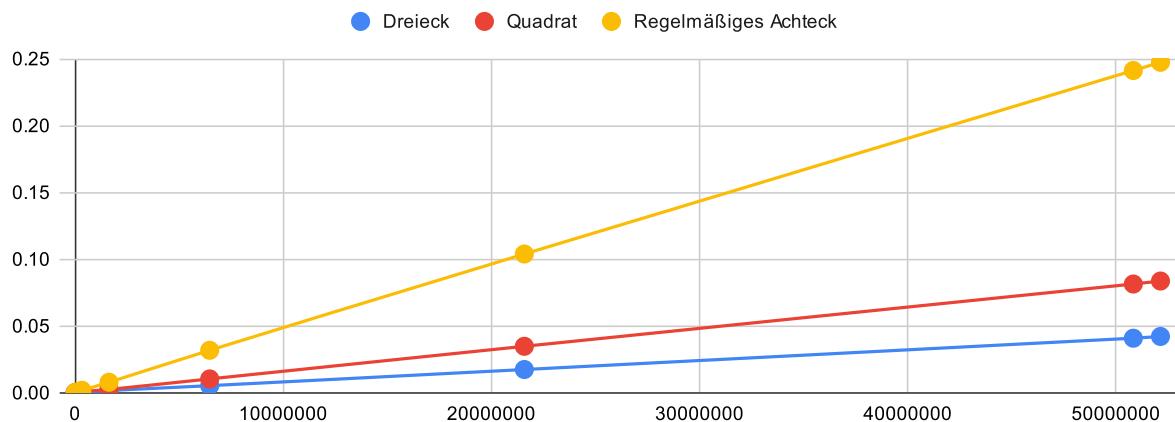


Abbildung 27: Renderzeit bei unterschiedlichen Polygone als Basis in Sekunden abhängig von der Anzahl der Punkte.

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. Die Eigenschaften sind separat als Wert zwischen 0 und  $2^{32} - 1$  gespeichert und werden mit einer Farbpalette in einen Farbverlauf umgewandelt. Dabei kann die Farbpalette unabhängig von den Daten ausgewählt werden. Die verfügbaren Farbpaletten sind in Abbildung 28 zu sehen.

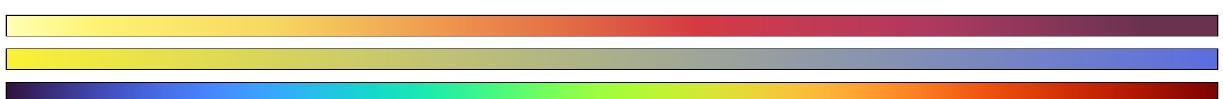


Abbildung 28: Verfügbare Farbpaletten.

## 5. Segmentierung

## 5.1. Auswahl

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

Im Octree wird vom Root-Knoten aus die Leaf-Knoten gefunden, welche den Strahl enthalten. Dabei werden die Knoten näher an der Position der Kamera bevorzugt. Für den Leaf-Knoten sind die Segmente bekannt, welche Punkte in diesem Knoten haben. Für jedes mögliche Segment wird für jeden Punkt überprüft, ob er entlang des Strahls liegt.

Sobald ein Punkt gefunden ist, müssen nur noch Knoten überprüft werden, die näher an der Kamera liegen, weil alle Punkte in weiter entfernten Knoten weiter als der momentan beste gefundene Punkt liegen.

To-do: Voxel Distance? | Point on ray?

## 5.2. Visualisierung

Im Octree kann zu den Punkten in einem Leaf-Knoten mehrere Segmente gehören. Um die Segmente einzeln anzuzeigen, wird jedes Segment separat abgespeichert. Sobald ein einzelnes 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 zu ermöglichen.

## 5.3. Exportieren

To-do: Implementierung Segment export

## 6. Detailstufen

Beim Anzeigen wird vom Root-Knoten aus zuerst geprüft, ob der momentane Knoten von der Kamera aus sichtbar ist. In Abbildung 29 ist ein Beispiel für das Filtern bei unterschiedlichen Detailstufen gegeben.

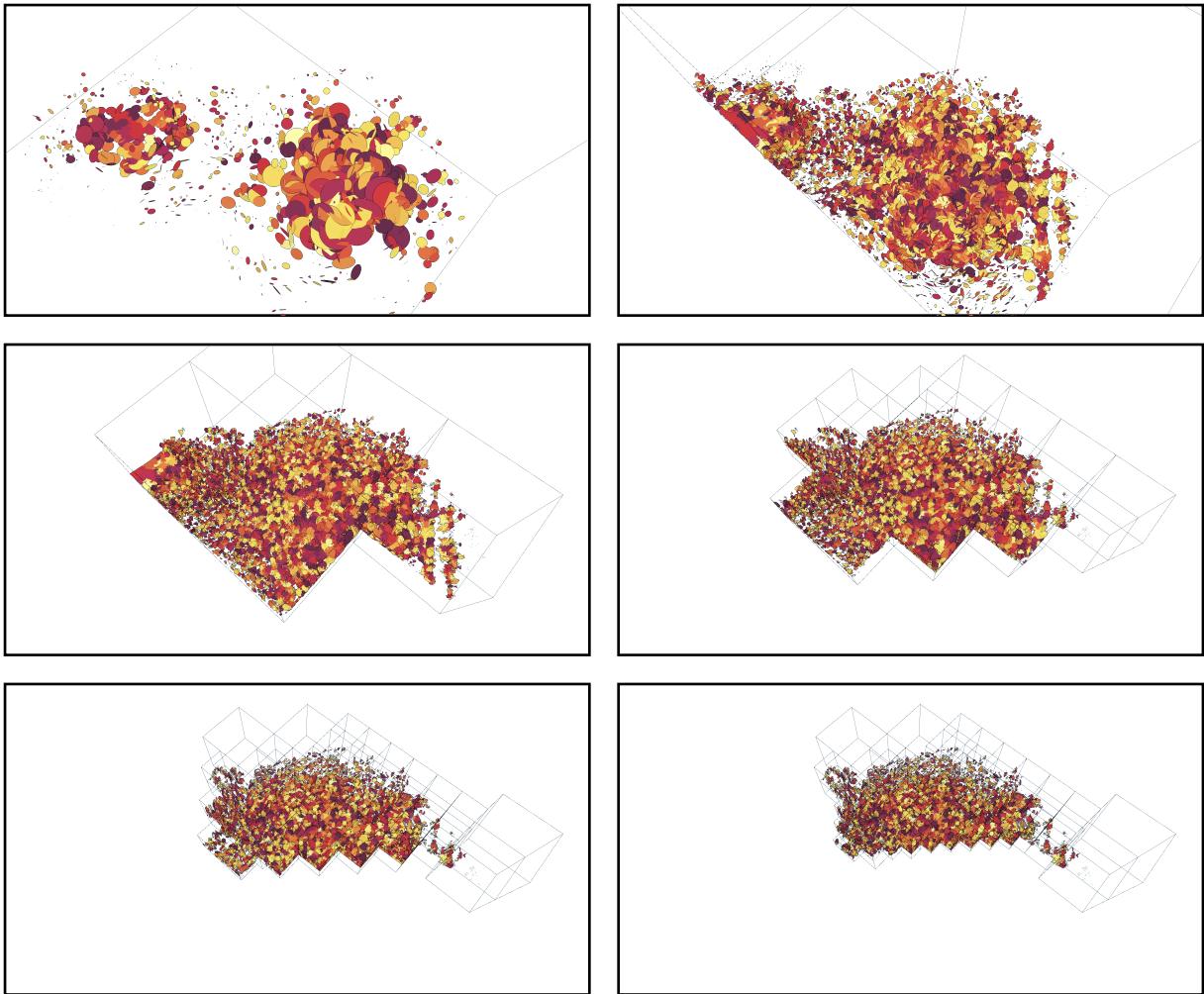


Abbildung 29: Sichtbare Knoten für unterschiedliche Detailstufen. Ein Knoten wird gerendert, solange ein Teil vom Knoten im Kamerafrustrum enthalten ist.

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

## V. Auswertung

To-do: Ergebnisse

### 1. Testdaten

Der benutzte Datensatz [9] beinhaltet 12 Hektar Waldfläche in Deutschland, Baden-Württemberg. Die Daten sind mit Laserscans aufgenommen, wobei die Scans von Flugzeugen, Drohnen und vom Boden aus durchgeführt wurden. Dabei entstehen 3D-Punktwolken, welche im komprimiert LAS Dateiformat gegeben sind.

Der Datensatz ist bereits in einzelne Bäume unterteilt. Zusätzlich wurden für 6 Hektar die Baumart, Höhe, Stammdurchmesser auf Brusthöhe und Anfangshöhe und Durchmesser der Krone gemessen. Mit den bereits bestimmten Eigenschaften können automatisch berechnete Ergebnisse validiert werden.

To-do: Was ALS, ULS und TLS Daten

To-do: Kurze Namen für Tabellen

To-do: TLS Daten

### 2. Import

Für den Import sind in [Abschnitt VI-3](#) die Messdaten für unterschiedliche Datensätze gegeben. In Abbildung 30 ist der Durchsatz beim Import angegeben. Dabei wird eine Importgeschwindigkeit von circa 400 000 Punkte pro Sekunde für die meisten Datensätze erreicht.

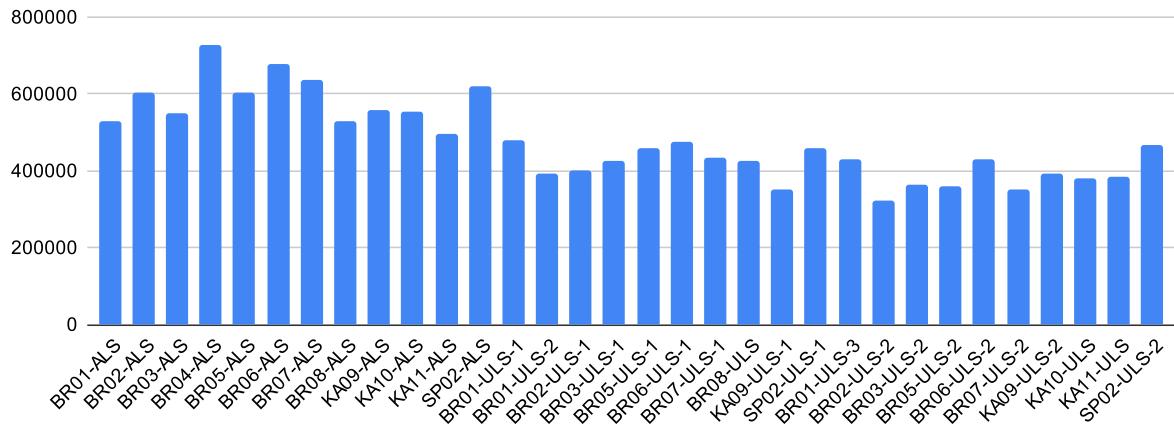


Abbildung 30: Geschwindigkeit vom Import in Punkte pro Sekunde.

In Abbildung 31 ist die Dauer für die einzelnen Phasen vom Import aufgeschlüsselt.

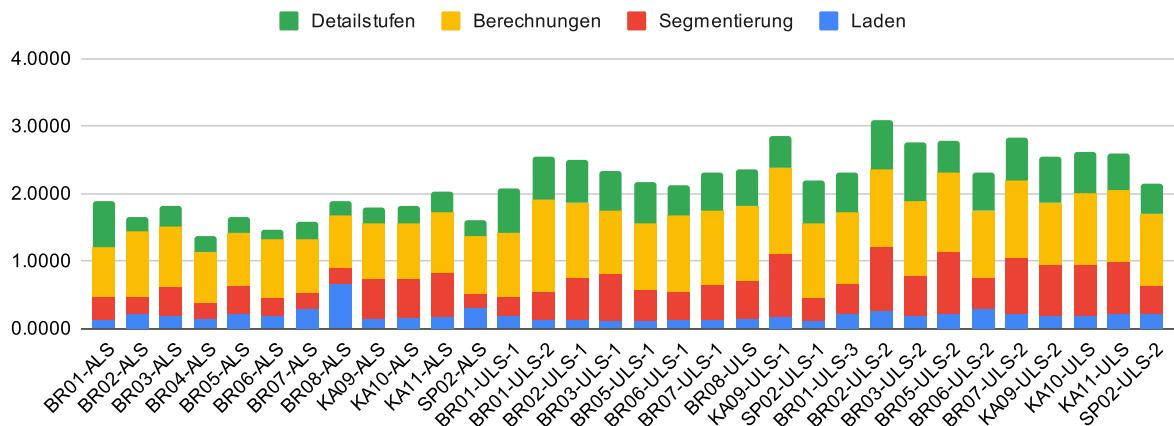


Abbildung 31: Dauer für die einzelnen Importphasen in  $\mu$ s pro Punkt.

### 3. Segmentierung in Bäume

Ein Beispiel für eine Segmentierung ist in Abbildung 32 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.

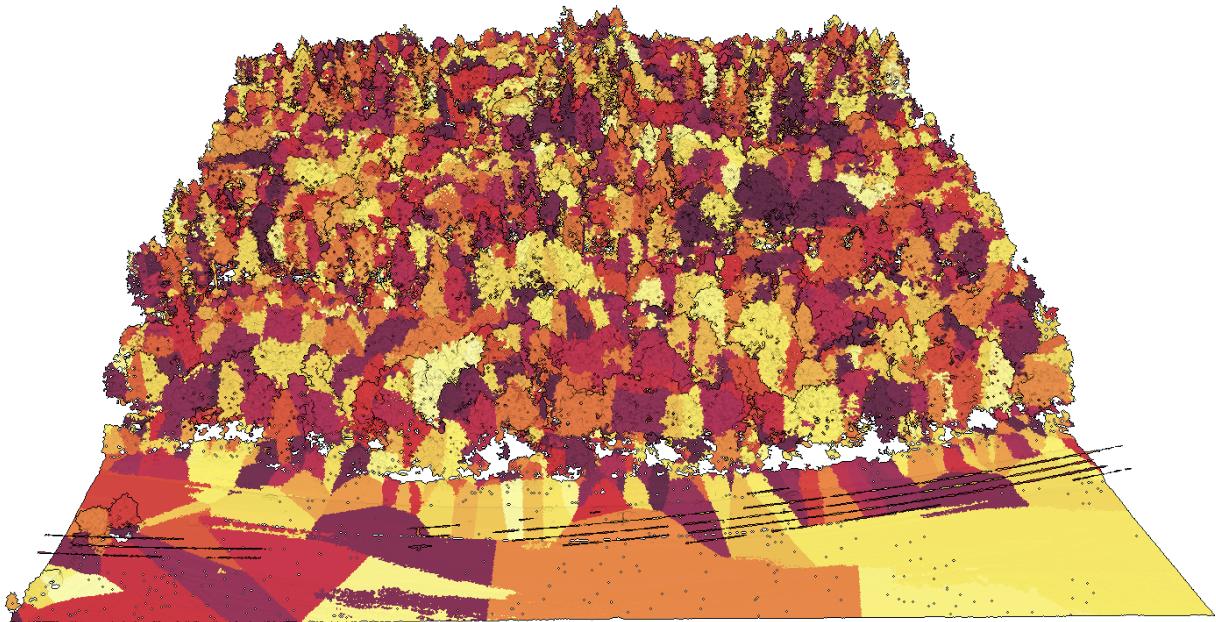


Abbildung 32: Segmentierung von einer Punktfolge.

To-do: neues Bild

Punkte, welche zu keinem Baum gehören, werden trotzdem zu den Segmenten zugeordnet. Bei frei stehenden Flächen entstehen separate Segmente und unter Bäumen werden die Punkte zum Baum zugeordnet.

To-do: Fehler wenn spitze gerade so in scheibe

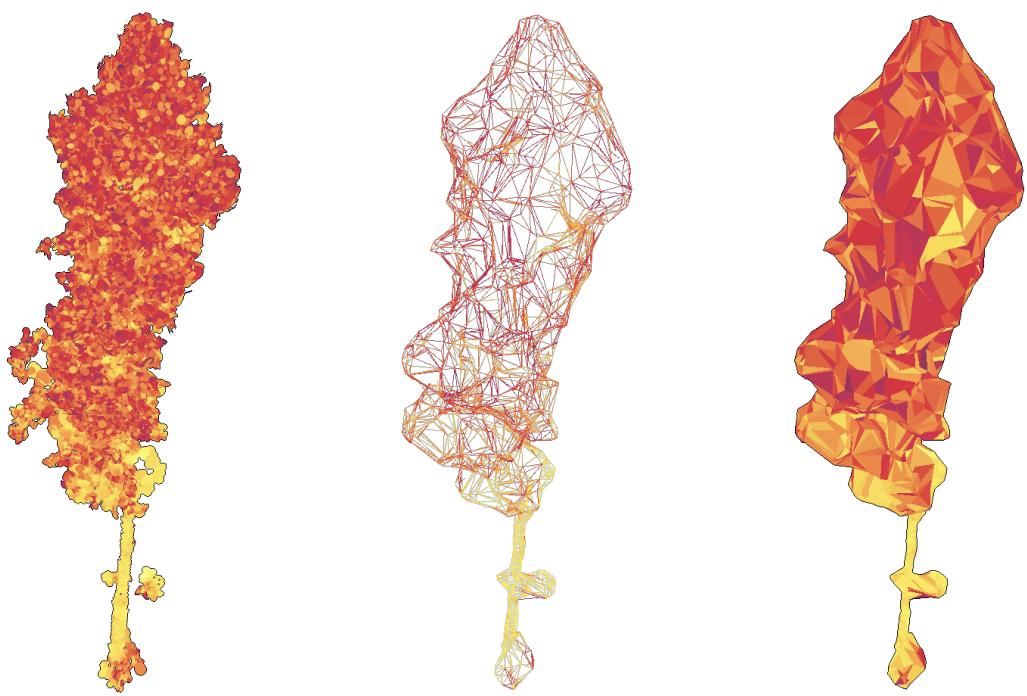
#### 4. Analyse von Segmenten

To-do: Berechnung Ergebnisse

#### 5. Triangulierung

To-do: Triangulierung Ergebnisse

To-do: Vergleich  $\alpha$



(a) Punkte

(b) Dreiecke umrandet

(c) Dreiecke ausgefüllt

Abbildung 33: Beispiel für eine Triangulierung von einem Baum.

## 6. Visualisierung

To-do: Visualisierung Ergebnisse

## 7. Fazit

To-do: Fazit

## 8. Diskussion

## 9. Ausblick

## VI. Appendix

To-do: Sinnvolle Ordnung

### 1. 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 34 gegeben.

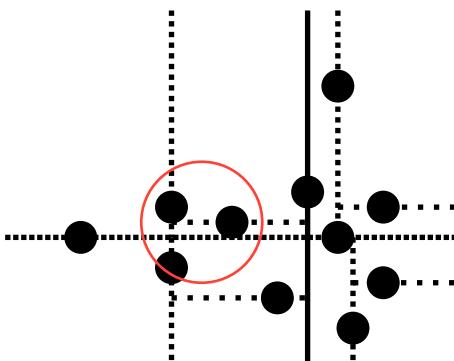


Abbildung 34: KD-Baum für Punkte in 2D. Für jede Unterteilung ist die Trennlinie gepunkteter gezeichnet. Weil der rote Kreis vollständig auf einer Seite der ersten Unterteilung ist, müssen die Punkte auf der anderen Seite nicht betrachtet werden.

#### 1.1. 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 [10] 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.

### **1.2. 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.

### **1.3. 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  $\infty$  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.

### **1.4. 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.

## 2. 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.

### 2.1. 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 Knoten 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 35 und Abbildung 36 sind Beispiele in 2D und 3D gegeben.

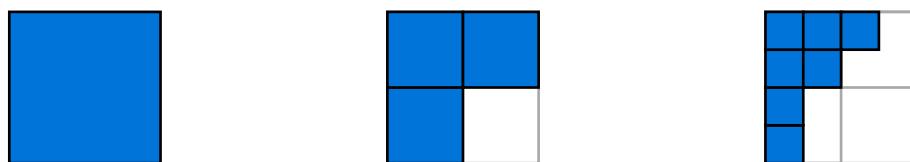


Abbildung 35: Unterschiedliche Stufen von einem Quadtree.

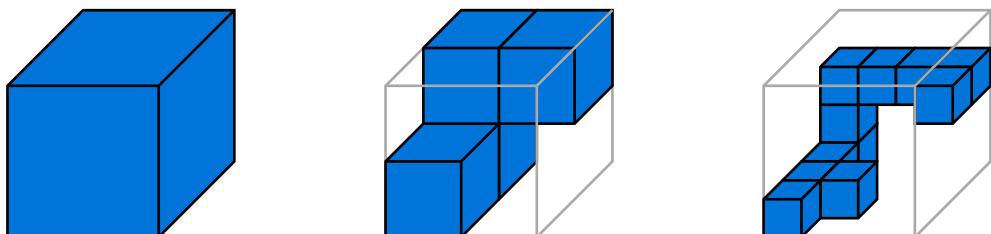


Abbildung 36: Unterschiedliche Stufen von einem Octree.

### 2.2. 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.

### 3. Messwerte vom Import

Datensatz	Datei	Daten Punkte	Segment Punkte	Detailstufen Punkte
BR01-ALS	ALS-on_BR01_2019-07-05_300m	12531758	12488283	5353797
BR02-ALS	ALS-on_BR02_2019-07-05_200m	5469363	5447584	2243313
BR03-ALS	ALS-on_BR03_2019-07-05_300m	12284905	12216643	6253200
BR04-ALS	ALS-on_BR04_2019-07-05_140m	2638216	2624587	1206212
BR05-ALS	ALS-on_BR05_2019-07-05_300m	13401630	13334884	5880173
BR06-ALS	ALS-on_BR06_2019-07-05_200m	6269955	6256270	2263182
BR07-ALS	ALS-on_BR07_2019-07-05_140m	3196739	3173027	1480571
BR08-ALS	ALS-on_BR08_2019-07-05_140m	2871871	2855507	1434091
KA09-ALS	ALS-on_KA09_2019-07-05_300m	14536939	14525577	6569850
KA10-ALS	ALS-on_KA10_2019-07-05_300m	13413120	13385318	6602555
KA11-ALS	ALS-on_KA11_2019-07-05_300m	14840932	14824682	7190141
SP02-ALS	ALS-on_SP02_2019-07-05_140m	2810265	2795664	1324312
BR01-ULS-1	ULS-off_BR01_2019-12-04	52179533	52141348	21454026
BR01-ULS-2	ULS-off_BR01_2020-03-27	62548947	62499492	23141070
BR02-ULS-1	ULS-off_BR02_2020-04-01	76639411	76556162	29514917
BR03-ULS-1	ULS-off_BR03_2020-03-31	63391431	63294976	28178811
BR05-ULS-1	ULS-off_BR05_2020-03-26	52169435	52114640	21481113
BR06-ULS-1	ULS-off_BR06_2020-03-27	60182129	60105599	20474140
BR07-ULS-1	ULS-off_BR07_2020-03-26	57159079	57086130	26184831
BR08-ULS	ULS-off_BR08_2020-03-31	66483369	66428152	26871724
KA09-ULS-1	ULS-off_KA09_2019-12-10	59967504	59852112	22659958
SP02-ULS-1	ULS-off_SP02_2020-04-01	64818104	64785079	23433217
BR01-ULS-3	ULS-on_BR01_2019-09-12	39420629	39314952	18774203
BR02-ULS-2	ULS-on_BR02_2019-08-24	41555427	41414918	19721750
BR03-ULS-2	ULS-on_BR0308_2019-08-24	83344946	83161982	37376830
BR05-ULS-2	ULS-on_BR05_2019-09-12	50247658	49949870	22923194
BR06-ULS-2	ULS-on_BR06_2019-09-12	48369613	48141254	19665260
BR07-ULS-2	ULS-on_BR07_2019-08-24	46833772	46686154	18260902
KA09-ULS-2	ULS-on_KA09_2019-09-13	50863340	50698058	22017351
KA10-ULS	ULS-on_KA10_2019-09-13	48167924	48006360	23057550
KA11-ULS	ULS-on_KA11_2019-09-06	58549518	58436609	26660630
SP02-ULS-2	ULS-on_SP02_2019-09-03	30811866	30743937	12817107

Tabelle 2: Messwerte (1).

Datensatz	Segmente	Punkte Laden	Segmentie- rung	Berechnun- gen	Detailstufen
BR01-ALS	6212	11.115789	40.577644	46.192574	19.51855
BR02-ALS	3236	5.4452295	11.7842	25.948767	1.8532621
BR03-ALS	7073	19.639217	36.14765	63.09323	11.374255
BR04-ALS	1826	3.2958136	4.4293747	14.067333	3.5932872
BR05-ALS	6642	19.979202	48.589592	68.87244	10.736564
BR06-ALS	2825	7.6129136	12.281778	26.213926	3.7624378
BR07-ALS	1679	7.2605247	5.4417944	13.093069	3.3996882
BR08-ALS	1670	4.8975563	4.946368	11.859719	3.8055406
KA09-ALS	5501	20.253628	44.4294	61.32498	9.591162
KA10-ALS	5589	13.374739	38.808857	62.368313	6.12319
KA11-ALS	6165	16.866133	43.835712	66.84172	17.142763
SP02-ALS	1717	2.8485336	4.3217897	13.497919	4.336844
BR01-ULS-1	6136	40.94356	204.46216	111.2325	39.32135
BR01-ULS-2	8010	54.895798	318.0728	132.72551	49.453686
BR02-ULS-1	12422	73.04987	593.49713	194.3728	86.028145
BR03-ULS-1	12513	56.934093	397.49368	170.23253	46.710426
BR05-ULS-1	7877	45.77166	305.79926	120.24389	44.79977
BR06-ULS-1	10442	62.7678	322.28363	138.45145	57.480915
BR07-ULS-1	10150	57.6198	371.6204	131.45961	41.95662
BR08-ULS	9626	69.43228	399.7619	136.86928	59.247993
KA09-ULS-1	10394	49.95081	337.27933	152.90706	49.862293
SP02-ULS-1	6111	59.225224	232.41098	122.057175	43.76792
BR01-ULS-3	8591	34.03373	197.88805	101.18946	34.713375
BR02-ULS-2	10661	38.96671	309.30426	120.889015	32.81025
BR03-ULS-2	12818	84.163345	472.56805	173.87895	76.906
BR05-ULS-2	18757	42.122265	656.2163	158.64189	34.002045
BR06-ULS-2	16668	38.630997	271.57794	175.99106	33.591534
BR07-ULS-2	11495	37.85239	329.6834	99.852005	36.01624
KA09-ULS-2	11882	39.857323	271.342	139.40224	44.893433
KA10-ULS	11919	37.97888	225.9031	147.53793	48.19509
KA11-ULS	9908	50.777466	262.81305	147.84149	47.006306
SP02-ULS-2	5955	30.434433	100.24474	73.175735	19.962479

Tabelle 3: Messwerte (2).

## 4. Punktwolkenformat

Die Struktur von einer Punktwolke ist in der `project.epc` Datei gespeichert. Dazu gehören die verfügbaren Eigenschaften und der Octree. Der benötigten Daten für `project.epc` werden in `project/src/lib.rs`<sup>2</sup> definiert.

### 4.1. Daten

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

Die Struktur ist in Abbildung 37 gegeben. Am Anfang der Datei wird für jeden Eintrag die Startposition  $s_i$  und die Länge  $l_i$  vom zugehörigen Datensegment gespeichert. Danach folgen die Datensegmente  $d_{\pi(i)}$  in beliebiger Reihenfolge  $\pi$ .

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 37: 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  $d_i$  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 gelesen.

---

<sup>2</sup><https://github.com/antonWetzel/treee/blob/main/project/src/lib.rs>

## 5. Quelltextstruktur

Name	Funktionalität
math	Funktionen für Vektoren, Matrizes, Projektionen und Winkel
project	Format für eine Punktfolge und Daten
k-nearest	Nachbarschaftssuche mit KD-Bäumen
input	Maus- und Tastatureingaben verarbeiten
triangulation	Triangulation von Punktfolgen
render	Rendern von Punktfolgen, Linien und Meshes mit wgpu
importer	Import von Punktfolgen
viewer	Visualisierung von Punktfolgen
treee	Gemeinsames Interface für importer und viewer

Tabelle 4: **To-do: Beschreibung**

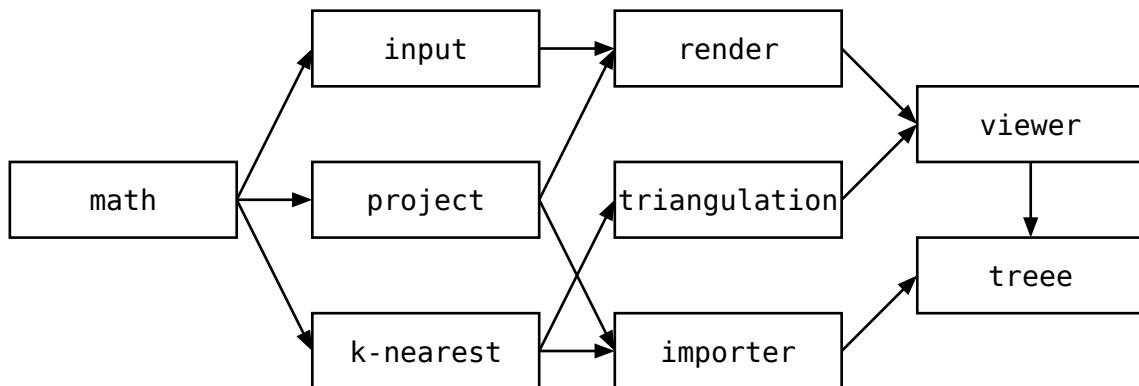


Abbildung 38: **To-do: Beschreibung**

## 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, 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] 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>
- [10] C. A. R. Hoare, „Algorithm 65: Find”, *Commun. ACM*, Bd. 4, Nr. 7, S. 321, Juli 1961, doi: [10.1145/366622.366647](https://doi.org/10.1145/366622.366647).