

Wörter: 10551

# Masterarbeit

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

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 Visualization Group

Datum: 04.04.2024



**TECHNISCHE UNIVERSITÄT  
ILMENAU**

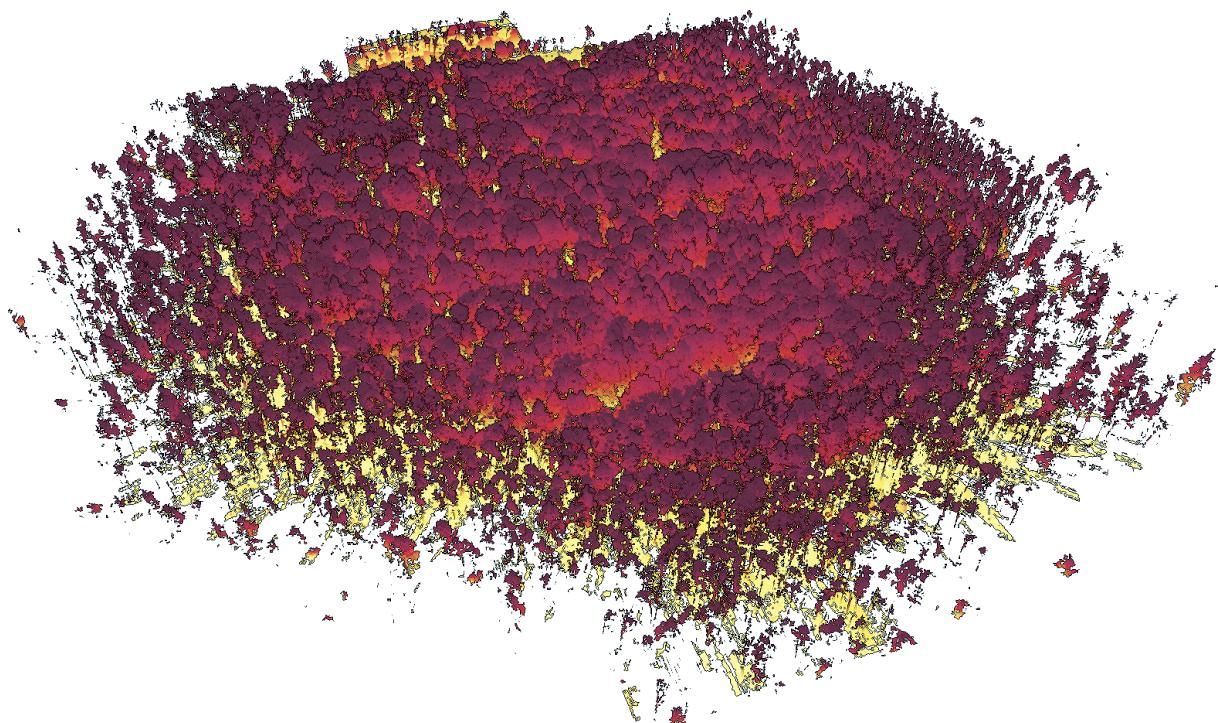
## **Abstrakt**

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

<b>1 Einleitung .....</b>	<b>1</b>
1.1 Motivation .....	1
1.2 Themenbeschreibung .....	1
1.3 Struktur der Arbeit .....	2
<b>2 Stand der Technik .....</b>	<b>4</b>
2.1 Punktdaten .....	4
2.2 Analyse von Bäumen .....	4
<b>3 Segmentierung von Waldstücken .....</b>	<b>5</b>
3.1 Ablauf .....	5
3.2 Bereiche bestimmen .....	5
3.3 Koordinaten bestimmen .....	7
3.4 Punkte zuordnen .....	8
<b>4 Visualisierung .....</b>	<b>9</b>
4.1 Punkte .....	9
4.1.1 Mögliche Polygone .....	9
4.1.2 Anzeigen im dreidimensionalen Raum .....	10
4.2 Detailstufen .....	11
4.2.1 Berechnung der Detailstufen .....	12
4.3 Eye-Dome Lighting .....	13
<b>5 Triangulierung .....</b>	<b>15</b>
5.1 Ziel .....	15
5.2 Ball-Pivoting Algorithmus .....	15
5.2.1 Überblick .....	15
5.2.2 $\alpha$ -Kugel für ein Dreieck .....	15
5.2.3 Ablauf .....	16
5.2.4 KD-Baum berechnen .....	16
5.2.5 Startdreieck bestimmen .....	16
5.2.6 Triangulierten Bereich erweitern .....	16
5.2.7 Komplettes Segment triangulieren .....	19
5.2.8 Vorauswahl .....	20
5.2.9 Auswahl von $\alpha$ .....	20
<b>6 Analyse von Bäumen .....</b>	<b>22</b>
6.1 Eingabedaten .....	22
6.2 Punkteigenschaften .....	22
6.2.1 Relative Höhe .....	23
6.2.2 Krümmung .....	23
6.2.3 Eigenschaften für die Visualisierung .....	24
6.3 Baumeigenschaften .....	25
6.3.1 Unterteilung in Scheiben .....	25
6.3.2 Bodenhöhe .....	26
6.3.3 Stammdurchmesser .....	26

6.3.4 Baumhöhe .....	27
6.3.5 Durchmesser von der Baumkrone .....	27
<b>7 Implementierung .....</b>	<b>28</b>
7.1 Technik .....	28
7.2 Benutzung .....	29
7.2.1 Installation .....	29
7.2.2 Ausführen .....	29
7.2.3 Import .....	29
7.2.4 Visualisierung .....	30
7.3 Struktur vom Quelltext .....	31
7.4 Format für eine Punktfolge .....	32
7.4.1 Daten .....	32
7.5 Import .....	33
7.5.1 Parallelisierung .....	34
7.6 Punkte .....	35
7.7 Segmente .....	36
7.7.1 Auswahl .....	36
7.7.2 Visualisierung .....	37
7.7.3 Exportieren .....	37
7.8 Detailstufen .....	37
<b>8 Auswertung .....</b>	<b>39</b>
8.1 Testdaten .....	39
8.2 Importgeschwindigkeit .....	39
8.3 Segmentierung von Waldstücken .....	40
8.4 Triangulierung .....	42
8.5 Visualisierung .....	43
8.6 Analyse von Bäumen .....	45
8.6.1 Punkteigenschaften .....	45
8.6.2 Baumeigenschaften .....	47
8.7 Fazit .....	49
8.8 Ausblick .....	50
<b>9 Appendix .....</b>	<b>51</b>
9.1 Systemeigenschaften .....	51
9.2 Messwerte vom Import .....	51
9.3 KD-Baum .....	53
9.3.1 Konstruktion .....	54
9.3.2 Suche mit festem Radius .....	55
9.3.3 Suche mit fester Anzahl .....	55
9.3.4 Schnelle Suche .....	55
9.4 Baum (Datenstruktur) .....	55
9.4.1 Konstruktion .....	55
9.4.2 Suchanfrage .....	56
<b>Glossar .....</b>	<b>57</b>
<b>Bibliographie .....</b>	<b>58</b>

# 1 Einleitung

## 1.1 Motivation

Größere Gebiete wie Teile von Wäldern können mit 3D-Scannern 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 inspiziert werden.

## 1.2 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.

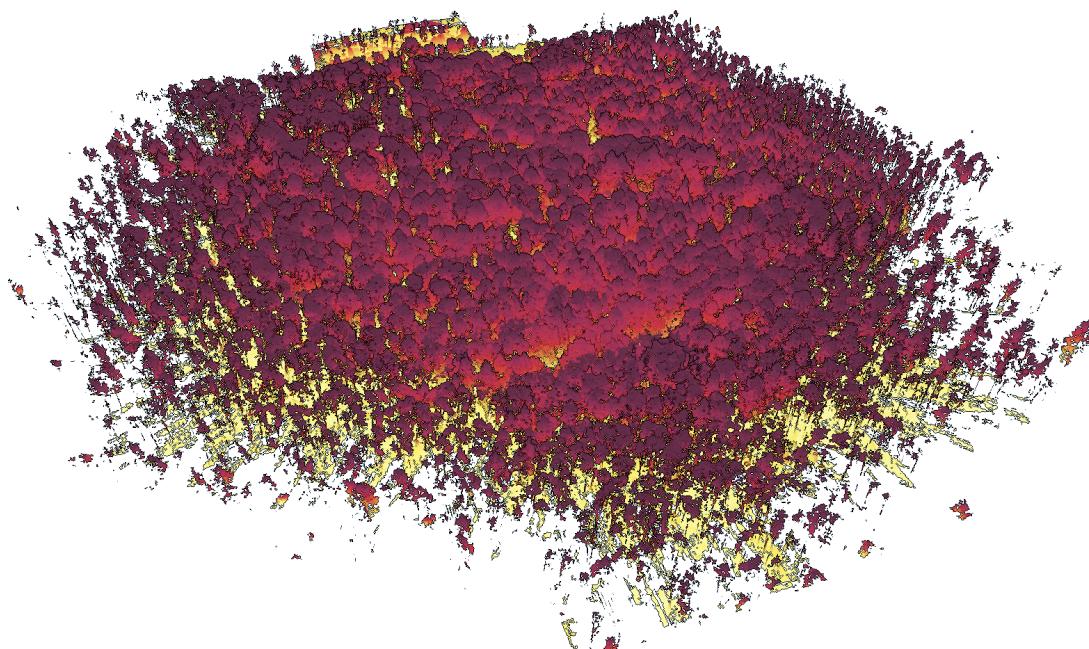


Abbildung 1: Waldstück mit Einfärbung der Punkte nach Höhe.

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.

## 1.3 Struktur der Arbeit

### Abschnitt 1: Einleitung

In der Einleitung wird die Motivation für die Arbeit erklärt, das Thema definiert und die Struktur der Arbeit vorgestellt.

### Abschnitt 2: Stand der Technik

Der Stand der Technik beschäftigt sich mit zugehörigen wissenschaftlichen und technischen Arbeiten. Dazu gehört die Aufnahme und Verarbeitung von Punktdaten und die Analyse von Bäumen mit den Daten.

### Abschnitt 3: Segmentierung von Waldstücken

Die Segmentierung erklärt den verwendeten Algorithmus für die Unterteilung von einer Punktfolge für ein Waldstück in mehrere Punktfolgen für jeweils einen Baum. Die Punktfolge kann dabei ungeordnet sein und für die einzelnen Punkte wird nur die Position vorausgesetzt.

### Abschnitt 4: Visualisierung

Im Abschnitt Visualisierung werden die Algorithmen erklärt, um die Punktfolgen mit allen gegebenen und berechneten Daten in Echtzeit zu rendern.

## **Abschnitt 5: Triangulierung**

Bei der Triangulierung wird für die Punktwolke von einem Baum ein geeignetes Dreiecksnetz gesucht. Mit dem Dreiecksnetz und den Punkten kann für den Baum ein dreidimensionales Mesh erstellt werden.

## **Abschnitt 6: Analyse von Segmenten**

Bei der Analyse wird mit der Punktwolke von einem Baum die charakteristischen Eigenschaften abgeschätzt. Dazu gehört der Durchmesser vom Stamm und der Krone und die Höhe vom gesamten Baum, Stamm und Krone.

## **Abschnitt 7: 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 8: 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.

## 2 Stand der Technik

### 2.1 Punktdaten

Punktwolken können mit unterschiedlichen Lidar Scanverfahren aufgenommen werden. Aufnahmen vom Boden oder aus der Luft bieten dabei verschiedene Vor- und Nachteile (Donager, Sánchez Meador, und Blackburn 2021). Bei einem Scan von Boden aus, kann nur eine kleinere Fläche mit hoher abgetastet werden, wodurch einzelne Bäume genau analysiert werden können (Disney 2019). Aus der Luft können größere Flächen mit konstanter Genauigkeit erfasst werden, wodurch Waldstücke aufgenommen werden können. Dafür ist die Punktanzahl pro Baum geringer (Suzuki u. a. 2021).

Ein häufiges Format für Lidar-Daten ist das LAS Dateiformat (Graham 2012). Bei diesem werden die Messpunkte als Liste 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 gespeichert (Isenburg 2013). Die Kompression ist verlustfrei und ermöglicht eine Kompressionsrate zwischen 5 und 15 je nach Eingabedaten.

*LAStools* ist eine Sammlung von Software für die allgemeine Verarbeitung von LAS Dateien (Hug, Krzystek, und Fuchs 2004). 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 Wäldern ausgelegt, wodurch Funktionalitäten wie Berechnungen von Baumeigenschaften mit zugehöriger Visualisierung nicht gegeben sind.

### 2.2 Analyse von Bäumen

Die Punktwolken gehören zu einzelnen Bäumen oder Waldstücken. Für Waldstücke wird die Punktwolke automatisch oder manuell in Segmente unterteilt (Burt, Disney, und Calders 2019; H. Weiser u. a. 2022), welche wie die einzelnen Bäume weiter analysiert werden können.

Mit der Punktwolke von einem Baum können charakteristische Informationen abgeschätzt werden. Dazu gehört die Berechnung von Baum-, Stamm- und Kronenhöhe (Suzuki, Shiozawa, Yamaba, und Amano 2021) oder der Durchmesser vom Stamm bei 1.3 m Höhe und der Krone (H. Weiser u. a. 2022).

In der Arbeit von Hackenberg u. a. (2015) beschäftigt sich mit der Berechnung vom Holzvolumen von einem Baum. Dafür wird der Baum mit Zylindern rekonstruiert, aus denen das totale Volumen berechnet wird. Für die Rekonstruktion können auch neuronale Ansätze verwendet werden (Liu u. a. 2021).

Eine Alternative zur Analyse von Bäumen mit Punktwolken ist die Verwendung von mehreren Fotografien als Datenquelle (Guo u. a. 2020). Für die Fotos werden die Tiefeinformationen geschätzt, womit eine dreidimensionale Rekonstruktion ermöglicht wird. Das Verfahren ist nur für die Rekonstruktion von einzelnen Bäumen geeignet, dafür sind Farbinformationen vorhanden, womit realistische Modelle erstellt werden können.

## 3 Segmentierung von Waldstücken

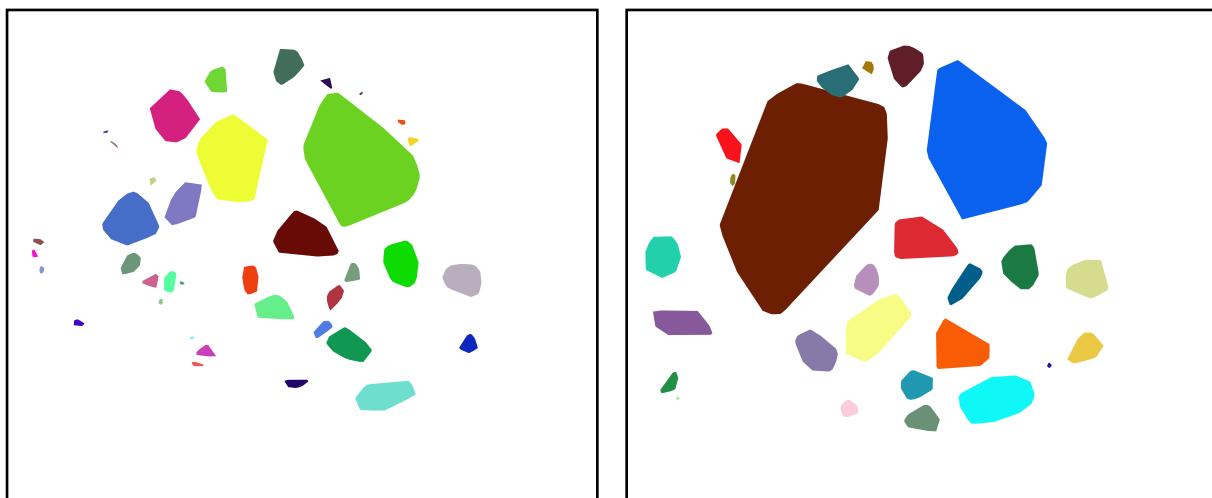
### 3.1 Ablauf

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.

### 3.2 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.



(a) Höhere Scheibe

(b) Tiefere Scheibe

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  $o = \frac{1}{|d|} \begin{pmatrix} d_x \\ d_y \end{pmatrix}$  ist orthogonal zu  $d$  und zeigt aus dem Bereich hinaus, so lange  $a$  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.

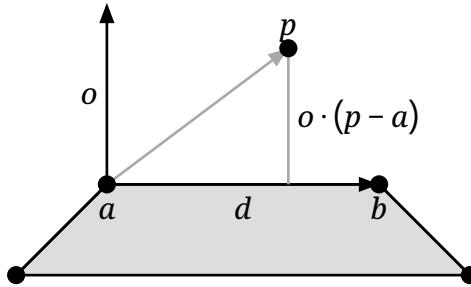


Abbildung 3: Berechnung vom Abstand vom Punkt  $p$  zur Kante zwischen  $a$  und  $b$ .

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.

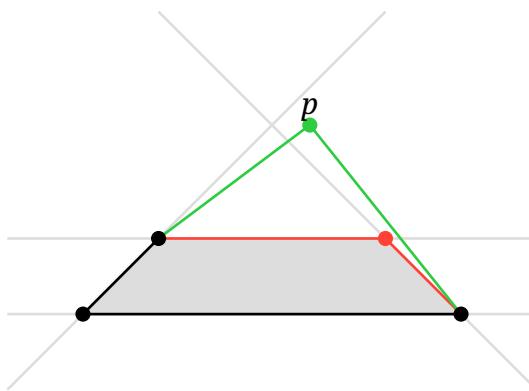


Abbildung 4: Hinzufügen vom Punkt  $p$  zum Bereich. Die Kanten in Rot werden entfernt und die Kanten in Grün werden hinzugefügt.

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.

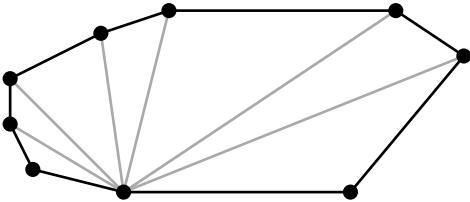


Abbildung 5: Unterteilung von einem Bereich in Dreiecke. Alle Dreiecke haben einen beliebigen Eckpunkt als ersten Punkt gemeinsam und die beiden anderen Punkte sind die Eckpunkte der Kanten ohne den ersten Punkt.

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 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.

### 3.3 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.

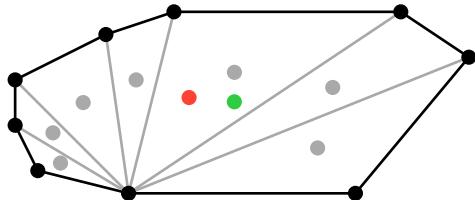


Abbildung 6: Berechnung vom Schwerpunkt von einem konvexen Bereich. Der korrekte Schwerpunkt in Grün ist die durchschnittliche Position der Schwerpunkte der Dreiecke nach der Fläche vom Dreieck gewichtet. In Rot ist die durchschnittliche Position der Eckpunkte.

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.

### 3.4 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.

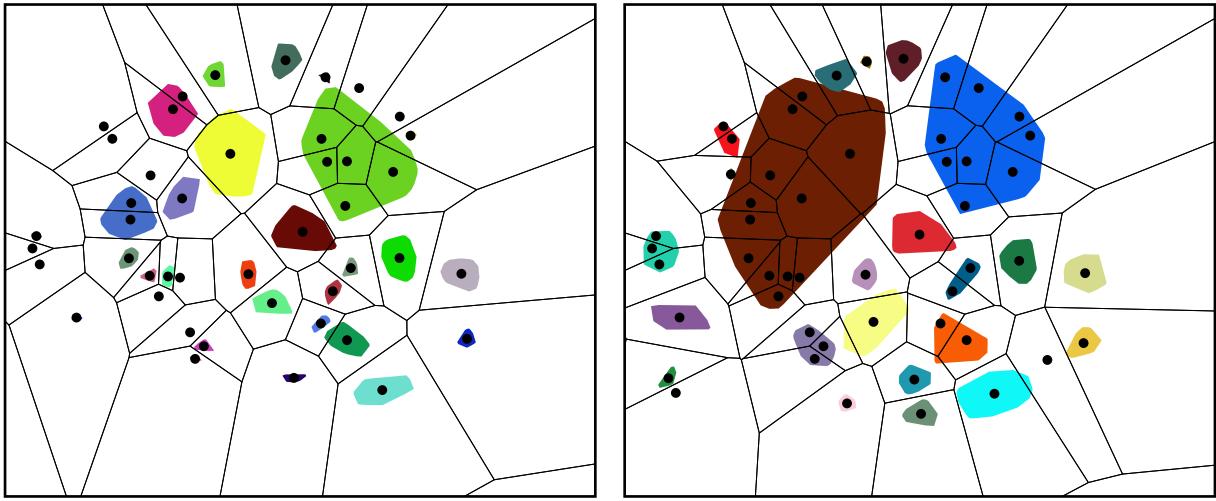


Abbildung 7: Berechnete Koordinaten für die Punkte mit zugehörigen Bereichen und Voronoi-Diagramm.

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.

# 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 mehreren 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 9 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  $\frac{w \cdot h}{2} = \frac{\tan(60^\circ) \cdot 3}{2} = \tan(60^\circ) \cdot 3 \approx 5.2$  benötigt.

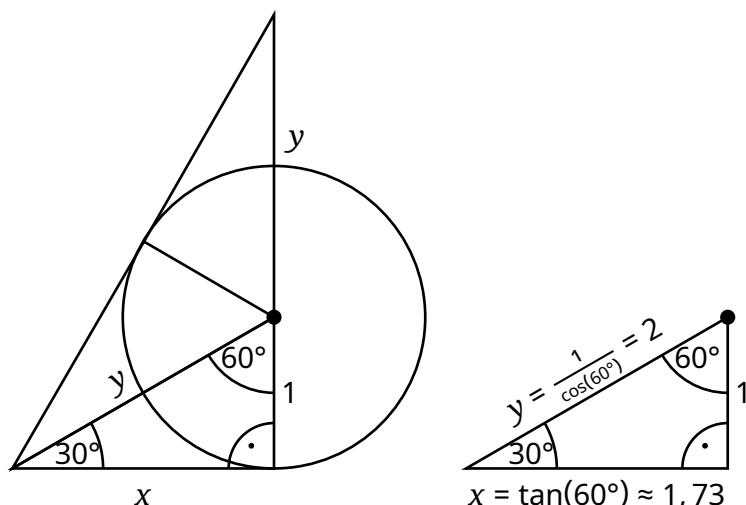


Abbildung 9: Seitenlänge für das kleinste gleichseitige Dreieck, welches den Einheitskreis enthält.

Das kleinste mögliche Viereck ist das Quadrat mit Seitenlänge 2. In Abbildung 10 ist die Konstruktion gegeben. Um diesen anzuseigen, werden zwei Dreiecke benötigt. Für die beiden Dreiecke werden dadurch sechs Ecken und eine Fläche von  $a^2 = 2^2 = 4$  benötigt.

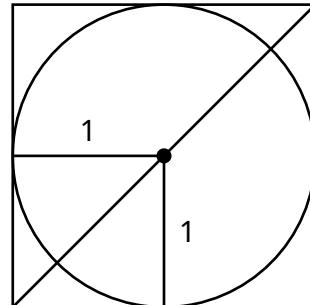


Abbildung 10: Seitenlänge für das kleinste Quadrat, welches den Einheitskreis enthält.

In Abbildung 11 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.

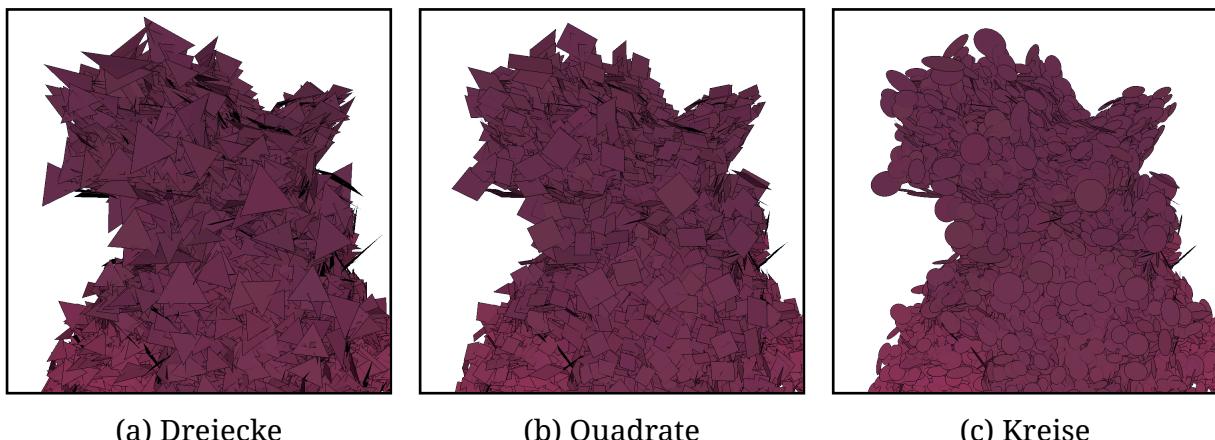


Abbildung 11: Die gleiche Punktfolge mit unterschiedlichen Polygonen und Kreisen für die Punkte.

#### 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 12 gegeben.

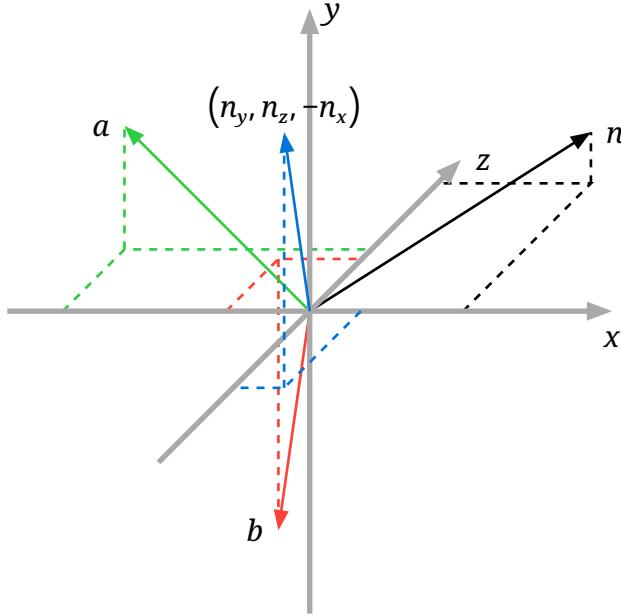


Abbildung 12: Beispiel für die Berechnung von  $a$  und  $b$  paarweise orthogonal zu  $n$ .

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 13 ist die Berechnung dargestellt.

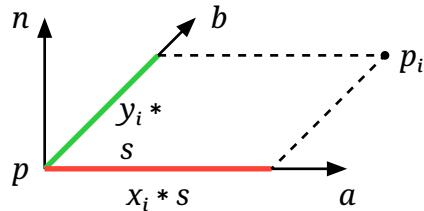


Abbildung 13: Berechnung von einem Eckpunkt.

## 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 14 sind die unterschiedlichen Stufen vom Octree mit zugehörigen Detailstufen visualisiert.

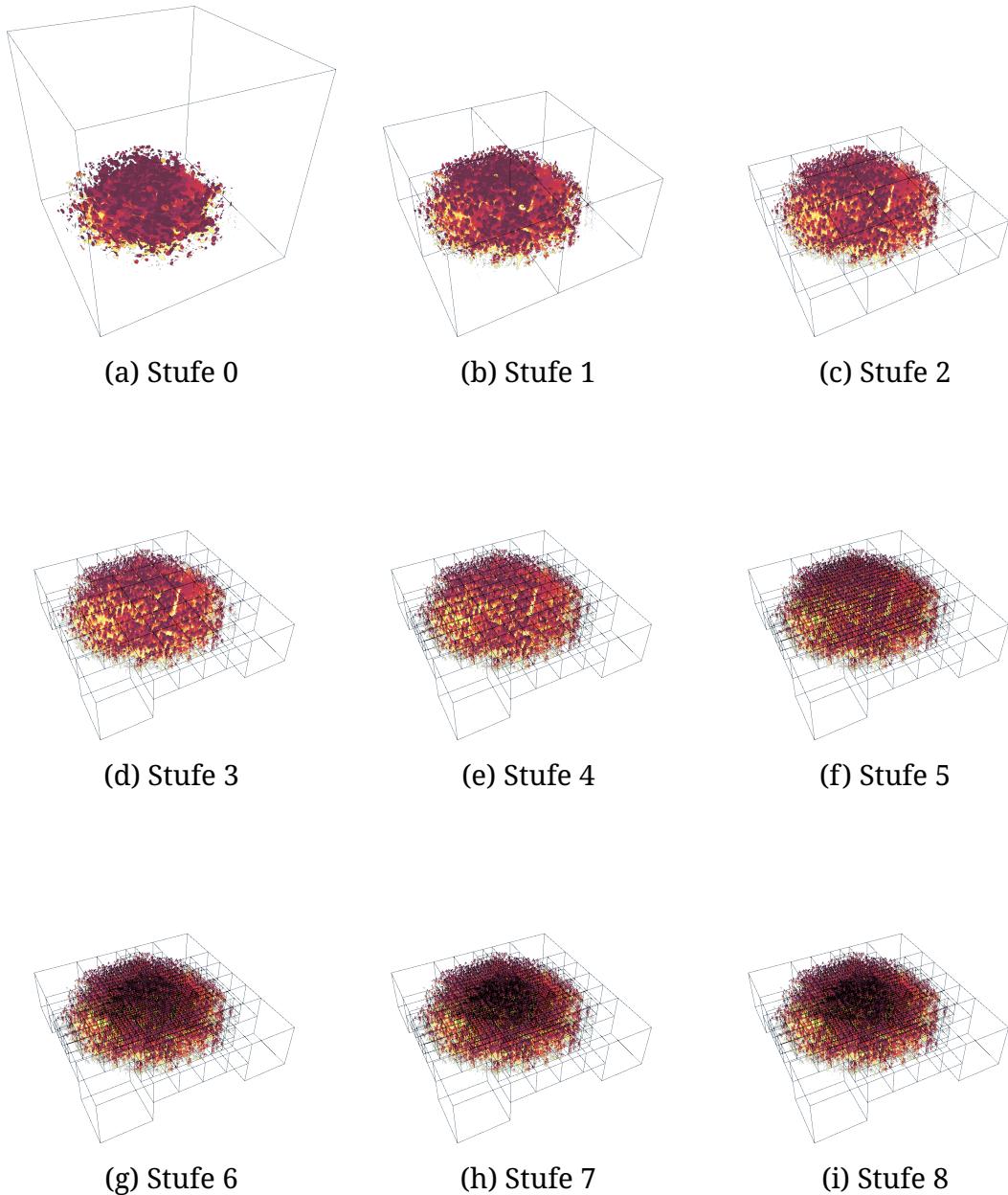


Abbildung 14: Unterschiedliche Detailstufen. Jeder Würfel enthält bis zu 32768 Punkte. In der höchsten Stufe werden alle Punkte im Datensatz angezeigt.

#### **4.2.1 Berechnung der Detailstufen**

Die Detailstufen werden wie bei „Fast Out-of-Core Octree Generation for Massive Point Clouds“ (Schütz, Ohrhallinger, und Wimmer 2020) 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 Polygonen das nähere 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 15 gegeben.

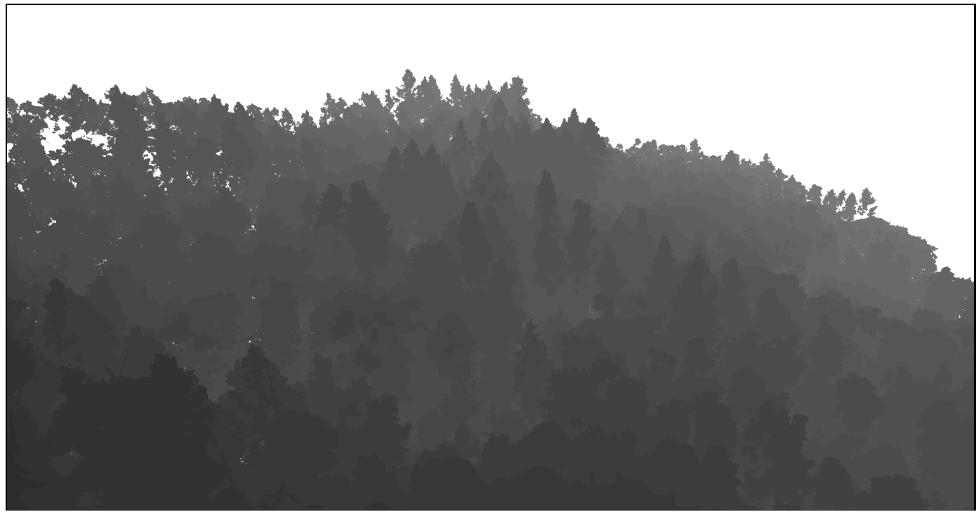


Abbildung 15: Tiefeninformationen nach dem Rendern der Szene. Je heller eine Position ist, desto weiter ist das Polygon zugehörig zur Koordinate von der Kamera entfernt.

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 16 gegeben.

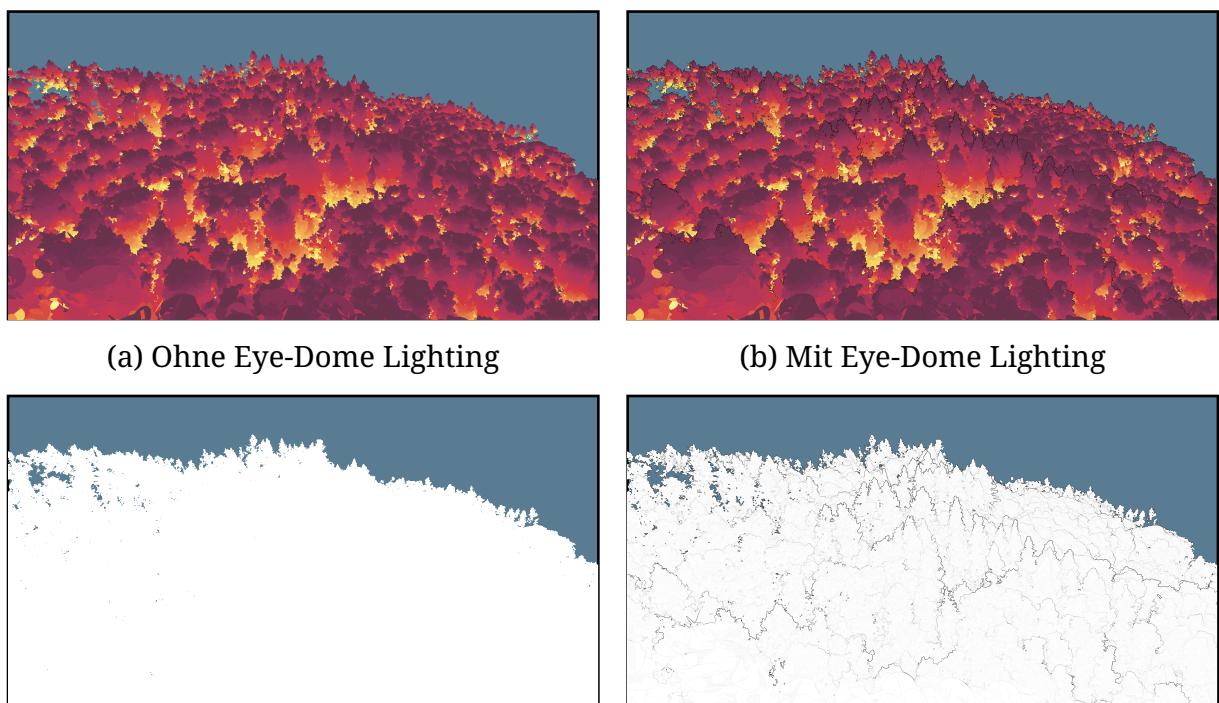


Abbildung 16: Waldstück mit und ohne Eye-Dome Lighting. Die Punkte sind zusätzlich in Weiß angezeigt, um den Effekt hervorzuheben.

# 5 Triangulierung

## 5.1 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.

## 5.2 Ball-Pivoting Algorithmus

### 5.2.1 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 alle anderen Punkte sind außerhalb der Kugel.

In Abbildung 17 ist ein Beispiel in 2D gegeben. Dabei werden die Linien gesucht, dass der zugehörige Kreis keine weiteren Punkte enthält.

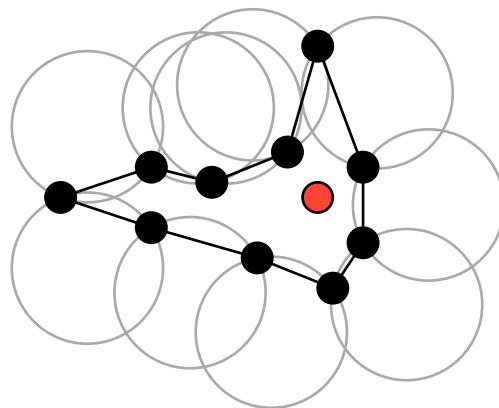


Abbildung 17: Ball-Pivoting Algorithmus in 2D. Für die äußeren Punkte in Schwarz wird eine Oberfläche gefunden. Für den inneren Punkt in Rot kann kein Nachbar gefunden werden, weil alle zugehörigen Kreise weitere Punkte enthalten würden.

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.

### 5.2.2 $\alpha$ -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.

### 5.2.3 Ablauf

#### 5.2.4 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. Die Konstruktion und Verwendung vom KD-Baum wird in Abschnitt 9.3 erklärt.

Mit dem KD-Baum kann auch überprüft werden, ob in einer  $\alpha$ -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.

#### 5.2.5 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 Kombination 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.

#### 5.2.6 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  $\alpha$ -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 18 ist ein Beispiel für eine Erweiterung in 2D gegeben. Im zweidimensionalen werden Kanten gesucht und Punkte werden als Pivot verwendet. Die vorherige Kante und der momentane Pivot-Punkt sind in Schwarz. Der  $\alpha$ -Kreis rollt entlang der markierten Richtung und berührt zuerst den grünen Punkt. Die weiteren Kandidaten sind in Rot und liegen weiter entlang der Rotation.

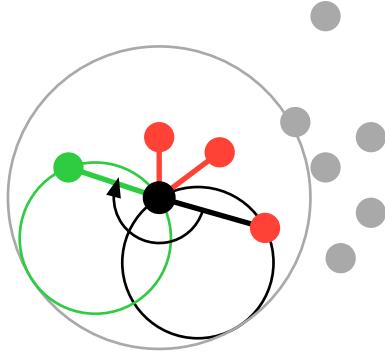


Abbildung 18: Erweiterung der gefundenen Oberfläche in 2D.

### 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  $\alpha$ , dadurch ist der Abstand vom Zentrum zum Mittelpunkt  $d_c = \sqrt{\alpha^2 - d^2}$ . In Abbildung 19 ist die Konstruktion veranschaulicht.

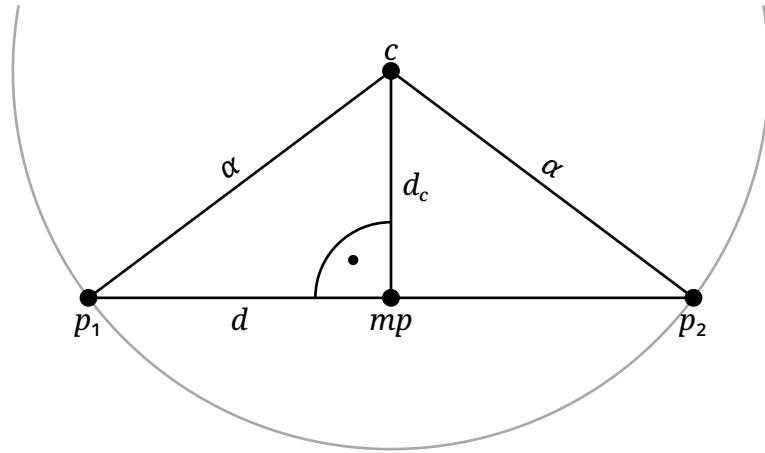


Abbildung 19: Konstruktion des Abstands der  $\alpha$ -Kugel vom Mittelpunkt der Kante

Die möglichen Punkte sind vom Zentrum der Kugel  $c$  maximal  $\alpha$  entfernt und  $c$  ist vom Mittelpunkt  $mp$  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  $\alpha$ -Kugel bestimmt, welche  $p_1$ ,  $p_2$  und  $p$  berührt. Die Kugel wird dabei wie in Abbildung 20 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 die restlichen Kandidaten wird der Winkel  $\varphi$  berechnet, wie weit um die Kante die Kugel gerollt wurde.

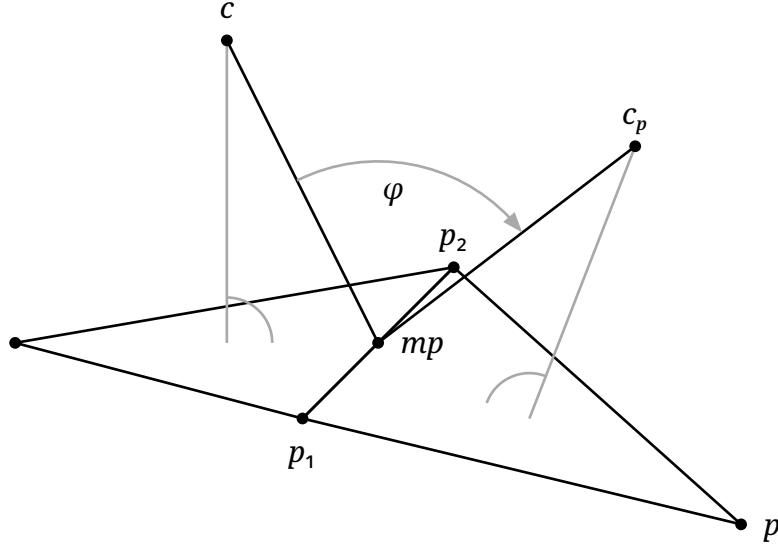


Abbildung 20: Berechnung von Zentrum der  $\alpha$ -Kugel und zugehöriger Winkel für einen Kandidatenpunkt

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, 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  $\varphi$  am kleinsten ist.

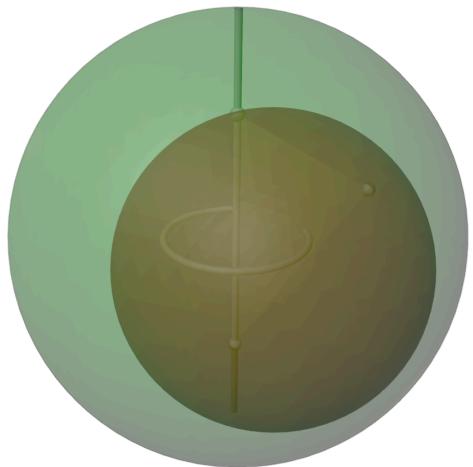
Es muss nicht kontrolliert werden, ob ein Punkt in der  $\alpha$ -Kugel von  $(p_1, p_2, p_3)$  liegt, weil diese immer leer ist. Würde ein weiterer Punkt in der Kugel liegen, so würde der zugehörige Winkel  $\varphi$  von diesem Punkt kleiner sein, weil der Punkt beim Rollen um die Kante früher von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

### Triangulierung erweitern

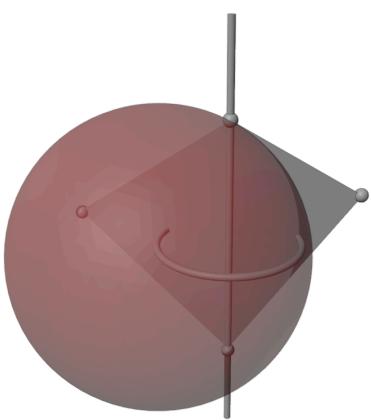
Das neu gefundene Dreieck mit den Eckpunkten  $(p_1, p_2, p_3)$  wird zur Triangulierung hinzugefügt. Die Kante  $(p_1, p_2)$  wird von den äußeren Kanten entfernt, dafür werden die neuen Kanten zwischen  $(p_1, p_3)$  und  $(p_3, p_2)$  hinzugefügt. Wenn eine der neuen Kante in den äußeren Kanten bereits 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 21 gegeben.



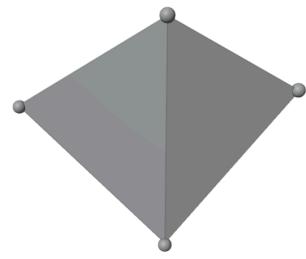
(a) Kante mit zugehörigem Dreieck,  $\alpha$ -Kugel und Ring mit Radius  $d_c$



(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 21: Erweiterung der Triangulierung in 3D.

### 5.2.7 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.

### 5.2.8 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.

### 5.2.9 Auswahl von $\alpha$

In Abbildung 22 wurde die Triangulation für die gleiche Punktwolke mit unterschiedlichen Werten für  $\alpha$  berechnet. Mit einem größerem  $\alpha$  wird das Ergebnis immer weiter vereinfacht. Bei einem kleinen Wert für  $\alpha$  können Lücken in der Triangulierung entstehen, wenn die Punkte weiter als  $2\alpha$  voneinander entfernt sind.

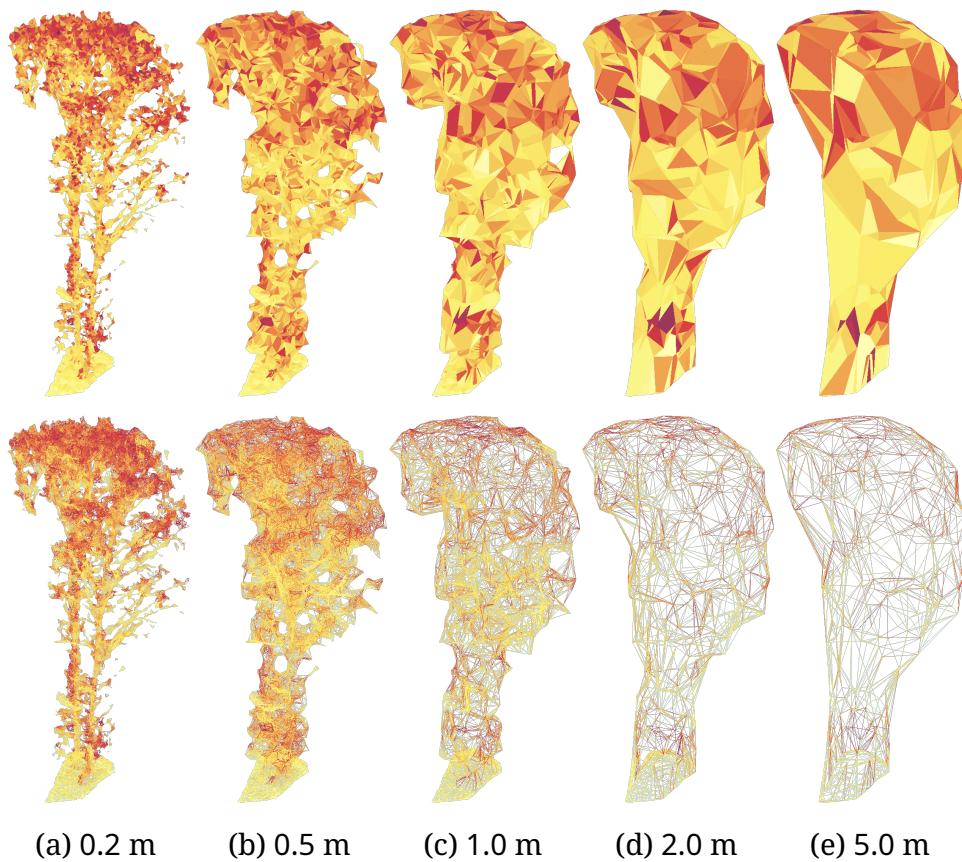


Abbildung 22: Triangulation für unterschiedliche  $\alpha$ . Im oberen Bild sind die Dreiecke ausgefüllt und im unteren Bild umrandet.

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

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

## 6 Analyse von Bäumen

Die charakteristischen Eigenschaften werden für jeden Baum einzeln berechnet. Für jeden Punkt im Baum wird zuerst die relative Höhe, lokale Krümmung und zugehörige horizontale Ausdehnung bestimmt. Mit diesen Daten wird dann eine Unterteilung in Boden, Stamm und Krone durchgeführt. Ein Beispiel für die Ergebnisse sind in Abbildung 23 zu sehen.

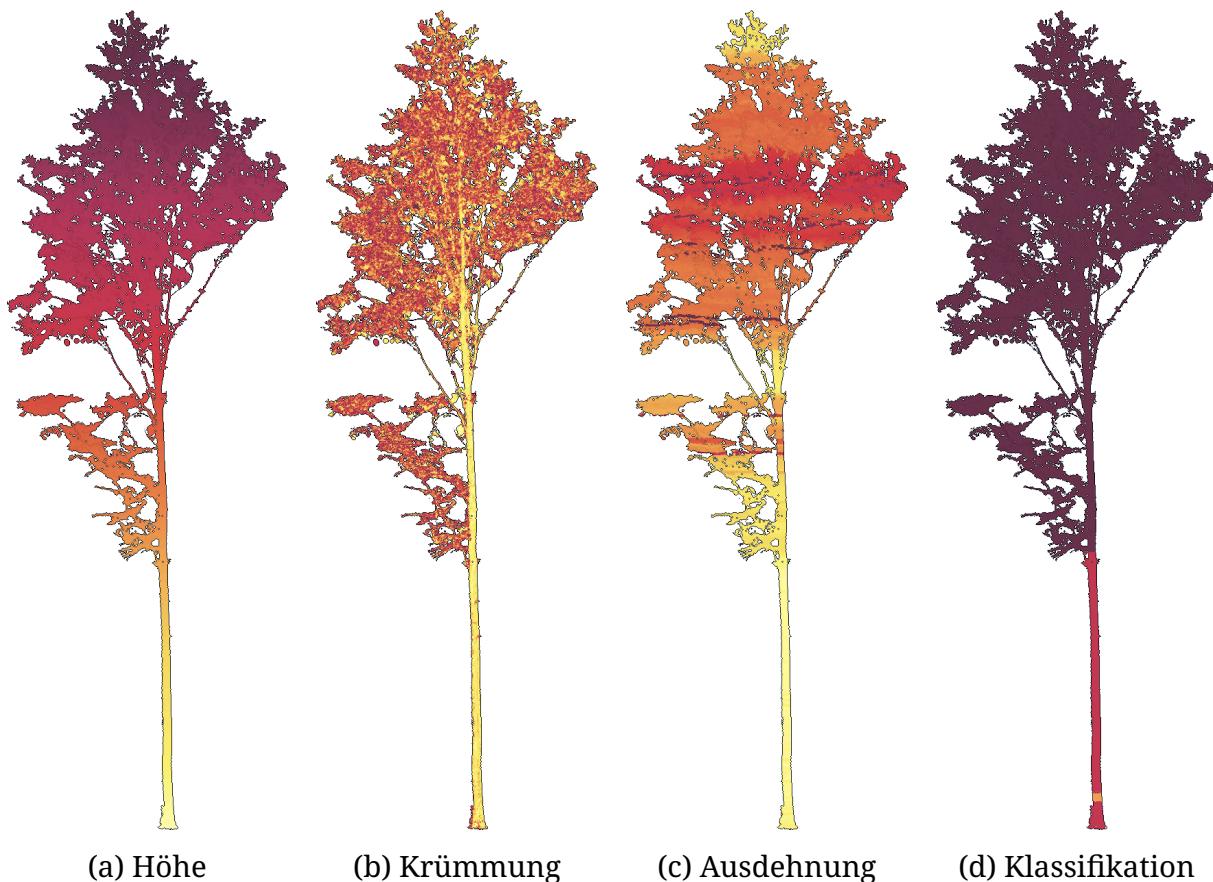


Abbildung 23: Segment basierend auf den berechneten Eigenschaften eingefärbt.

Mit der Unterteilung wird für den Stamm und die Krone die Höhe und der Durchmesser abgeschätzt.

## 6.1 Eingabedaten

Die Punkte sind als Liste der Länge  $n$  gegeben. Für den Punkt  $i \in \mathbb{N}_0^{n-1}$  ist nur die globale Position  $p_i = (p_{ix}, p_{iy}, p_{iz})$  bekannt. Die Punkte sind dabei ungeordnet, wodurch aufeinanderfolgende Punkte in der Liste weit voneinander entfernte Positionen haben können.

## 6.2 Punkteigenschaften

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 mit den Punkten 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 Abschnitt 9.3 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.

### 6.2.1 Relative Höhe

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

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

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.

### 6.2.2 Krümmung

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

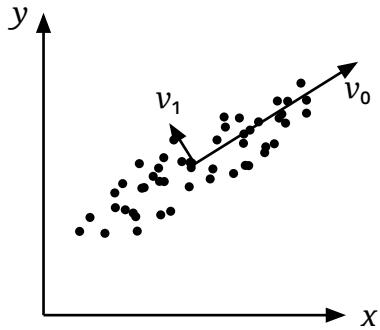
$$s_i = \frac{1}{k} \cdot \sum_{j \in N_i} p_j$$

Mit dem Schwerpunkte kann die Kovarianzmatrix  $C_i$  bestimmt werden.

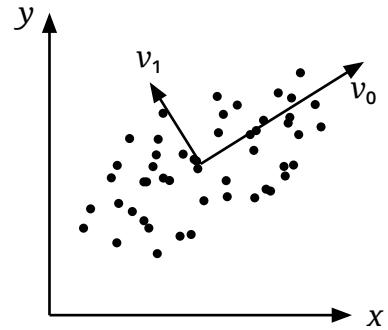
$$C_i = \frac{1}{k} \cdot \sum_{j \in N_i} \begin{pmatrix} (p_{jx} - s_{ix})^2 & (p_{jx} - s_{ix}) \cdot (p_{jy} - s_{iy}) & (p_{jx} - s_{ix}) \cdot (p_{jz} - s_{iz}) \\ (p_{jy} - s_{iy}) \cdot (p_{jx} - s_{ix}) & (p_{jy} - s_{iy})^2 & (p_{jy} - s_{iy}) \cdot (p_{jz} - s_{iz}) \\ (p_{jz} - s_{iz}) \cdot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

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  $v_{i\alpha}$  mit  $\alpha \in \{0, 1, 2\}$  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 zweidimensionale Eigenvektoren ist in Abbildung 24 gegeben.



(a)  $\lambda_0 = 4\lambda_1$



(b)  $\lambda_0 = 2\lambda_1$

Abbildung 24: Eigenwerte und zugehörige Eigenvektoren der Kovarianzmatrix für eine Punktmenge. Die Länge der Vektoren ist abhängig 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.

### 6.2.3 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. In Abbildung 25 ist ein Beispiel gegeben.

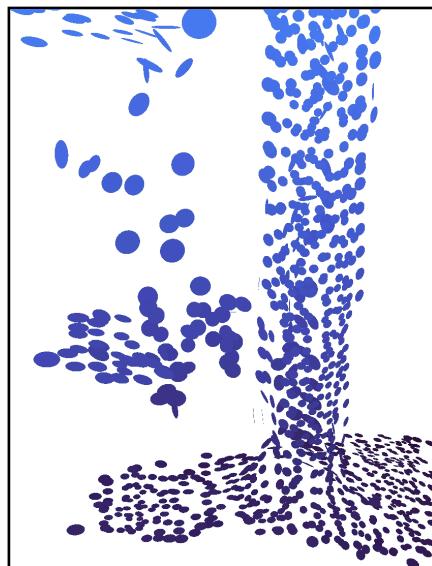


Abbildung 25: Ausschnitt von einer Punktwolke. Die dichteren Punkte beim Stamm sind kleiner als die umliegenden Punkte und die Orientierung ändert sich beim Übergang vom Stamm zum Boden.

Für die Orientierung wird die Normale bestimmt, welche orthogonal zur geschätzten zugehörigen Oberfläche vom Punkt ist. Dafür werden die Eigenvektoren aus Abschnitt 6.2.2 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.

### 6.3 Baumeigenschaften

Für den Baum wird die Gesamthöhe und die Höhe und Durchmesser vom Stamm und der Krone gesucht. Dafür werden die Punkte in Scheiben unterteilt, diese dem Boden, Stamm oder Krone zugeordnet und dann wird mit den Scheiben die gesuchten Werte berechnet. Die einstellbaren Parameter dafür sind in Tabelle 1 gelistet.

Name	Funktion
$w$	Breite von einer Scheibe
$h_g$	Maximale Suchhöhe für den Boden
$s_g$	Skalierungsfaktor für die Mindestfläche vom Boden
$h_t$	Höhe für die Berechnung vom Stammdurchmesser
$r_t$	Bereich für die Berechnung vom Stammdurchmesser
$d_c$	Differenz zwischen den Stammdurchmesser und dem Mindestdurchmesser der Krone

Tabelle 1: Parameter für die Klassifikation.

#### 6.3.1 Unterteilung in Scheiben

Zuerst werden die Punkte in gleich breite Scheiben unterteilt. Dafür wird mit der Breite  $w$ ,  $h_{\min}$  und  $h_{\max}$  die Anzahl der benötigten Scheiben  $c$  und für jeden Punkt  $i$  der Index  $s_i$  der zugehörigen Scheibe berechnet.

$$c = \left\lceil \frac{h_{\max} - h_{\min}}{w} \right\rceil + 1 \quad s_i = \left\lfloor \frac{p_{iy} - h_{\min}}{w} \right\rfloor$$

Für jede Scheibe wird der horizontale konvexe Bereich bestimmt, der alle Punkte in der Scheibe beinhaltet. Dafür werden die Methoden wie bei der Segmentierung in Abschnitt 3.2 verwendet. Mit den Bereichen wird die Fläche bestimmt, welche von den Punkten in der Scheibe belegt wird. In Abbildung 26 ist ein Beispiel gegeben.

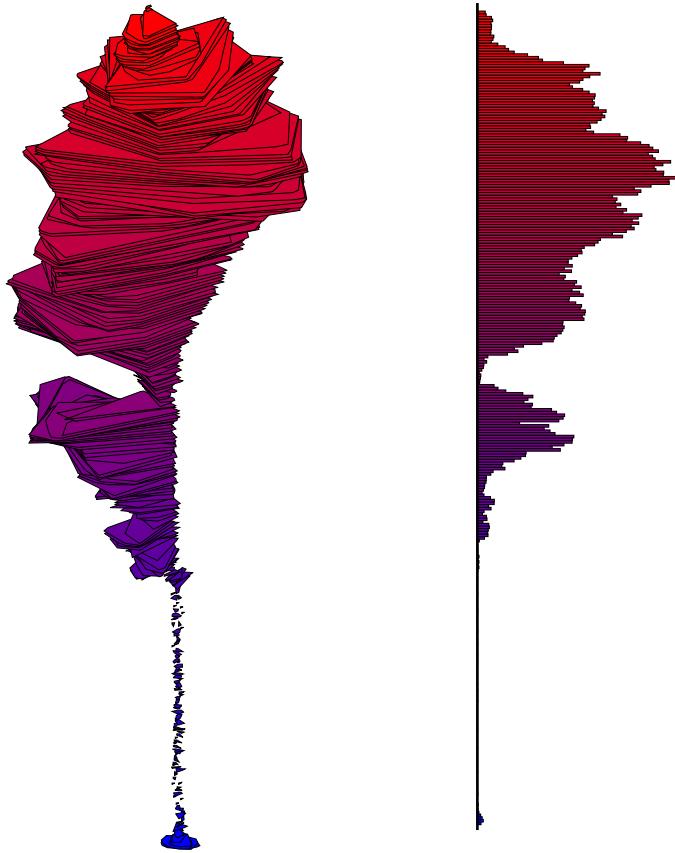


Abbildung 26: Baum in Scheiben unterteilt mit der zugehörigen Fläche für jede Scheibe.

### 6.3.2 Bodenhöhe

Mit den Flächen wird zuerst die kleinste Fläche  $a_{\min}$  und die größte Fläche  $a_{\max}$  bestimmt.

Danach wird die Bodenhöhe  $g$  gesucht. Dafür werden die Suchhöhe  $h_g$  und der Skalierungsfaktor  $s_g$  verwendet. Von unten wird die erste Scheibe gesucht, welche eine größere Fläche als  $s_g \cdot a_{\min}$  hat. Ist die Scheibe höher als  $h_g$ , wird die Höhe der untersten Scheibe als Boden verwendet. Ist die Scheibe niedriger als  $h_g$ , wird die nächste Scheibe gesucht, dessen Fläche kleiner als  $s_g \cdot a_{\min}$  ist, und die Höhe von dieser Scheibe wird als  $g$  verwendet.

### 6.3.3 Stammdurchmesser

Mit der berechneten Bodenhöhe  $g$ , der Messhöhe  $h_t$  und dem Bereich  $r_t$  wird der Stammdurchmesser  $d_t$  bestimmt. Zuerst werden alle Punkte bestimmt, deren Höhen im Bereich  $[g + h_t - \frac{r_t}{2}, g + h_t + \frac{r_t}{2}]$  liegen. Mit dem MSAC-Algorithmus<sup>1</sup> wird der Kreis gesucht, welcher am besten die Punkte beschreibt.

Beim MSAC-Algorithmus werden wiederholt genug zufällige Datenpunkte ausgewählt, dass aus diesen der gewünschte Wert eindeutig berechnet werden kann. Danach wird für jeden Datenpunkt die Abweichung zum Wert bestimmt. Die Abweichung wird auf einen Maximalwert beschränkt, um den Einfluss von weit entfernten Datenpunkten zu

---

<sup>1</sup>m-estimator sample consensus

verringern. Die Summe von allen Abweichungen ist der Fehler für den momentanen Wert. Der Wert mit dem geringsten Fehler wird als das finale Ergebnis verwendet (Torr und Zisserman 2000).

Am Anfang wird als Durchmesser der Standardwert 0,5 m und als bester Fehler der größtmögliche Wert verwendet. Dann werden wiederholt drei zufällige Punkte ausgewählt, mit denen das Zentrum und der Durchmesser vom zugehörige Kreis eindeutig berechnet werden kann. Mit allen Punkten wird der Fehler für den Kreis berechnet. Um den Effekt von Datenpunkten zu verringern, welche nicht zum Stamm gehören wird die Abweichung auf 0,2 m beschränkt. Ist der momentane Fehler kleiner als der bisher bester Fehler, wird der Durchmesser als neuen besten Wert verwendet und der beste Fehler wird aktualisiert.

Nach allen Iterationen wird der Durchmesser vom besten Kreis als der geschätzte Stammdurchmesser verwendet.

### 6.3.4 Baumhöhe

Mit dem Stammdurchmesser  $d_t$  und der Mindestdifferenz  $d_c$  wird die Höhe vom Anfang der Krone bestimmt. Dafür wird die Scheibe gesucht, dessen Durchmesser  $d_c$  größer als  $d_t + d_c$  ist. Die zugehörige geschätzte Mindestfläche  $a_m$  kann abhängig vom Durchmesser geschätzt werden.

$$a_m = \pi \cdot \left( \frac{d_t + d_c}{2} \right)^2$$

Danach wird die erste Scheibe höher als der Boden gesucht, dessen Fläche größer als  $a_m$  ist. Der Baum geht vom Boden bis zur höchsten Scheibe, der Stamm vom Boden bis zu dieser Scheibe und die Krone von dieser Scheibe bis zur höchsten Scheibe.

### 6.3.5 Durchmesser von der Baumkrone

Für den Durchmesser von der Baumkrone wird für alle Scheiben in der Krone die Scheibe mit der größten Fläche  $a_c$  gesucht. Mit der Fläche wird der zugehörige Durchmesser  $d_c$  geschätzt.

$$d_c = 2 \cdot \sqrt{\frac{a_c}{\pi}}$$

# 7 Implementierung

## 7.1 Technik

Das Softwareprojekt ist auf [GitHub](#)<sup>2</sup> 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 2 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 2: Benutzte Bibliotheken

<sup>2</sup><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, solange eine Rust-Bibliothek existiert, welche das Format einlesen kann.

## 7.2 Benutzung

### 7.2.1 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-Release](#)<sup>3</sup> heruntergeladen werden. Die Schritte zum selber kompilieren sind im [Readme](#)<sup>4</sup> verfügbar.

### 7.2.2 Ausführen

In Tabelle 3 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 3: Mögliche Befehle für das Programm.

### 7.2.3 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 4 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.

---

<sup>3</sup><https://github.com/antonWetzel/treeee/releases>

<sup>4</sup><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 m	Breite der horizontalen Scheiben für die Segmentierung
--segmenting-max-distance	1 m	Mindestabstand zwischen Bereichen für die Segmentierung
--calculations-slice-width	0,1 m	Breite der horizontalen Scheiben für die Analyse von einem Baum
--ground-min-area-scale	1,5	Mindestgröße vom Boden im Vergleich zu der kleinsten Scheibe
--ground-max-search-height	1 m	Maximale Suchhöhe für den Anfang vom Boden
--trunk-diameter-height	1,3 m	Höhe für den Durchmesser vom Stamm
--trunk-diameter-range	0,2 m	Bereich für den Durchmesser vom Stamm
--crown-diameter-difference	1 m	Unterschied vom Durchmesser zwischen dem Stamm und dem Anfang von der Krone
--neighbors-count	31	Maximale Anzahl der Punkte in der Nachbarschaft
--neighbors-max-distance	1 m	Maximale Distanz vom Punkt zu den Punkten in der Nachbarschaft
--lod-size-scale	0,95	Skalierungsfaktor für die Fläche der kombinierten Punkte für die Detailstufen
--proj-location	+proj=utm +ellps=GRS80 +zone=32	Region für die geographischen Koordinaten

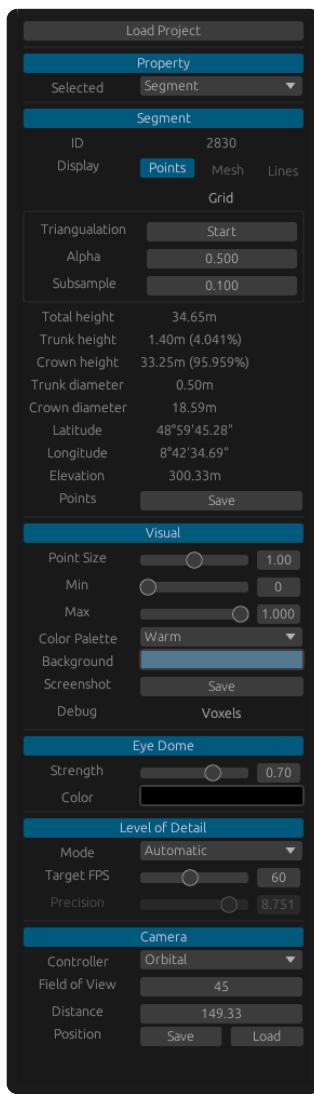
Tabelle 4: Optionen für den Import.

#### 7.2.4 Visualisierung

Um eine Punktwolke 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 Punktwolken 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 27 erklärt.



- **Load Project**
  - Die geladene Punktwolke ändern
- **Property**
  - Die angezeigte Eigenschaft ändern
- **Segment**
  - Punkte, Linien oder Dreiecke anzeigen
  - Triangulation starten
  - Informationen über das ausgewählte Segment
  - Segment speichern
- **Visual**
  - Punktgröße ändern
  - Punkte basierend auf der ausgewählten Eigenschaft filtern
  - Farbpalette wechseln
  - 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 27: Benutzerinterface mit den verfügbaren Optionen und Informationen.

### 7.3 Struktur vom Quelltext

Das Softwareprojekt ist in mehrere Module unterteilt, um den Quelltext zu strukturieren. In Tabelle 5 und Abbildung 28 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 Punktwolke und zugehörige Daten
k-nearest	Nachbarschaftssuche mit KD-Bäumen
render	Rendern von Punktwolken, Linien und Dreiecken mit wgpu
viewer	Visualisierung von Punktwolken
triangulation	Triangulation von Punktwolken
importer	Import von Punktwolken
treee	Gemeinsames Interface für importer und viewer

Tabelle 5: Module vom Projekt mit zugehöriger Funktionalität.

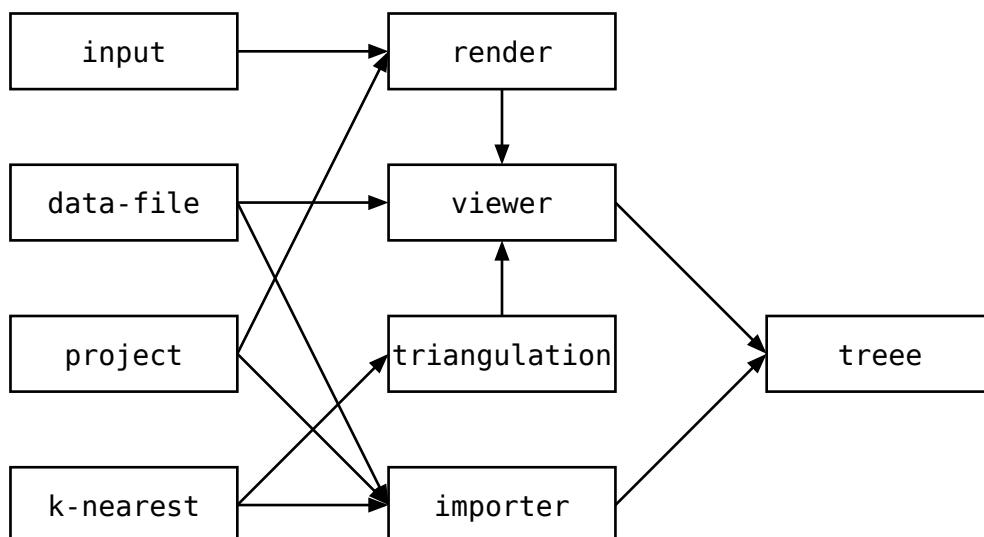


Abbildung 28: Abhängigkeiten der Module untereinander.

## 7.4 Format für eine Punktwolke

Die Struktur von einer Punktwolke 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](#)<sup>5</sup> definiert.

### 7.4.1 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.

<sup>5</sup><https://github.com/antonWetzel/treee/blob/main/project/src/lib.rs>

Die Struktur ist in Abbildung 29 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  $\pi$ .

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 29: 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.

## 7.5 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 30 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 Punktwolke wird abgespeichert.

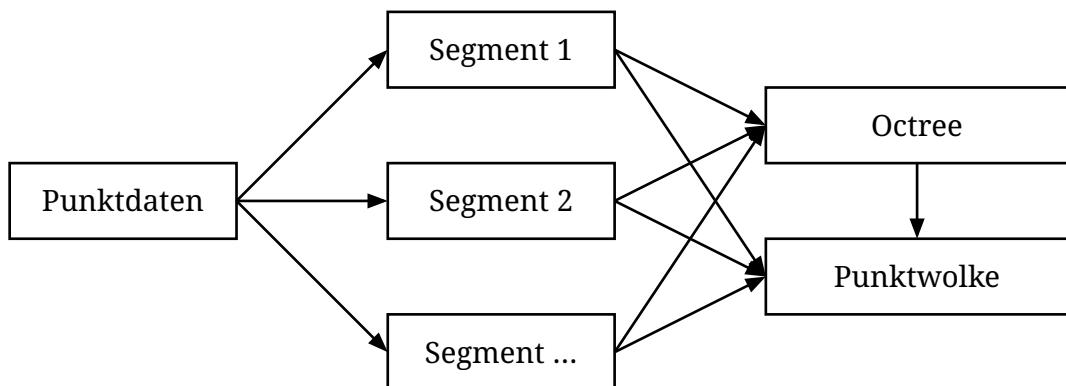
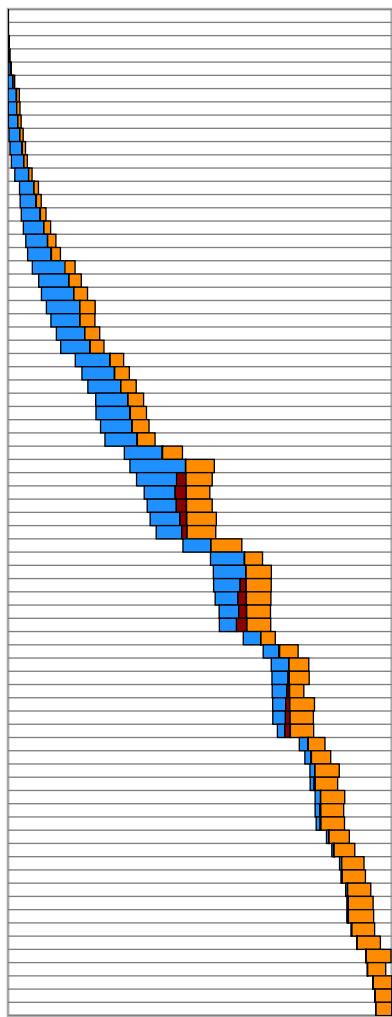


Abbildung 30: Datenfluss beim Import.

### 7.5.1 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 31 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 31: 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.

## 7.6 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 wieder verwendet.

Für die Grafikpipeline wird das Polygon in Dreiecke zerlegt. In Abbildung 32 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.

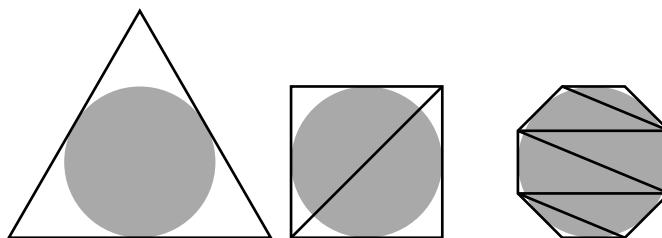


Abbildung 32: Zerlegung von unterschiedlichen Polygonen in Dreiecke.

In Abbildung 33 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.

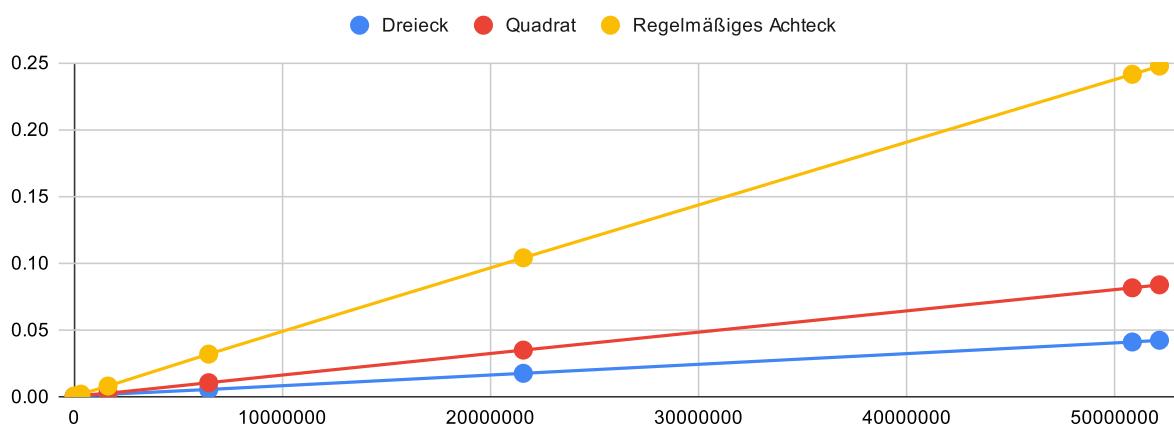


Abbildung 33: Renderzeit bei unterschiedlichen Polygonen als Basis in Sekunden abhängig von der Anzahl der Punkte.

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 34 zu sehen.



Abbildung 34: Verfügbare Farbpaletten.

## 7.7 Segmente

### 7.7.1 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 35 überprüft werden, ob der Strahl den Voxel berührt (Majercik u. a. 2018).

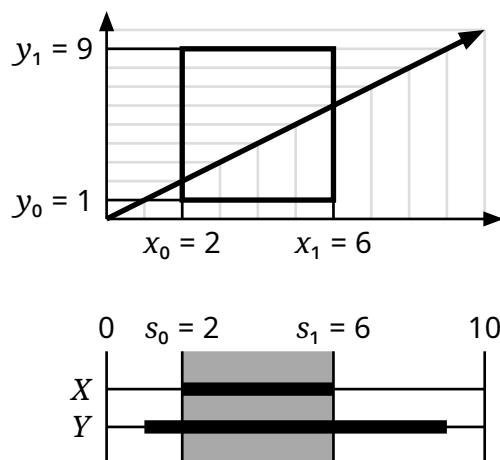


Abbildung 35: Schnittmenge von Strahl und Quadrat in 2D. Zuerst wird für jede Achse der Bereich bestimmt, für den der Strahl im Quadrat liegen kann. Die Schnittmenge ist die Überschneidung von den Bereichen für alle Achsen.

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.

### 7.7.2 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.

### 7.7.3 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.

## 7.8 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 36 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.

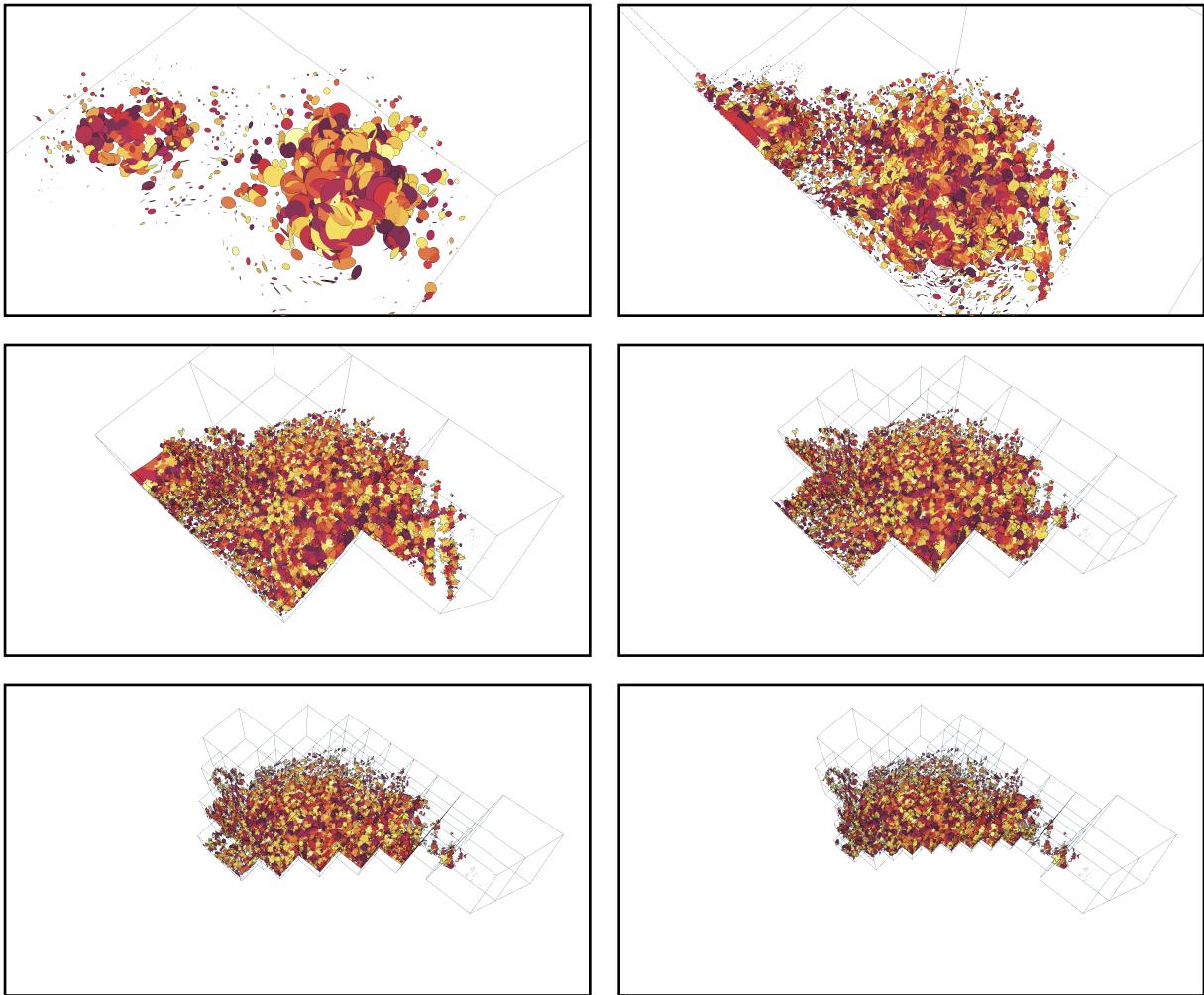


Abbildung 36: Sichtbare Knoten für unterschiedliche Detailstufen. Ein Knoten wird gerendert, solange ein Teil vom Knoten im Sichtfeld der Kamera liegt.

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.

# 8 Auswertung

## 8.1 Testdaten

Der benutzte Datensatz (Hannah Weiser u. a. 2022) beinhaltet 12 Hektar Waldfläche in Deutschland, Baden-Württemberg. Die Daten sind mit Laserscans aufgenommen, wobei die Scans von Flugzeugen (ALS<sup>6</sup>), Drohnen (ULS<sup>7</sup>) und vom Boden (TLS<sup>8</sup>) aus durchgeführt wurden. Dabei entstehen 3D-Punktwolken, welche im komprimiert LAS Dateiformat gegeben sind.

Für die meisten Waldstücke existieren ALS-, ULS- und TLS-Daten. Dabei enthalten die ALS-Daten die wenigsten und die TLS-Daten die meisten Punkte. Bei den ALS- und ULS-Daten sind die Punkte gleichmäßig über das gescannte Gebiet verteilt. Bei den TLS-Daten wird von einem zentralen Punkt aus gescannt, wodurch die Punktedichte nach außen immer weiter abnimmt.

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.

## 8.2 Importgeschwindigkeit

Für den Import sind in Abbildung 37 die Importgeschwindigkeiten in Punkte pro Sekunde gegeben. Die genauen Messwerte sind in Abschnitt 9.2 und die Eigenschaften vom verwendeten System sind in Abschnitt 9.1 gelistet. Für jedes Waldstück wurde ein zufälliger Datensatz von den ALS-, ULS- und TLS-Daten verwendet.

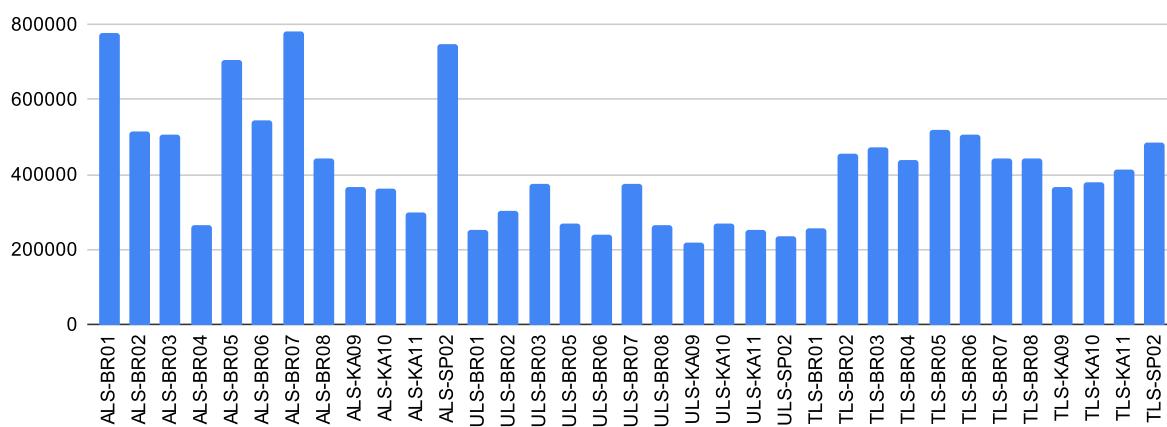


Abbildung 37: Geschwindigkeit vom Import in Punkte pro Sekunde.

Bei den ALS-Daten wird für die meisten Datensätze eine Importgeschwindigkeit von 500 000 erreicht. Durch die kleinen Datenmengen schwankt aber die Importgeschwindigkeit.

<sup>6</sup>aircraft laser scan

<sup>7</sup>uncrewed aerial vehicle laser scan

<sup>8</sup>terrestrial laser scan

digkeit stark. Für die ULS-Daten wird eine Importgeschwindigkeit von 200 000 und für die TLS-Daten von 400 000 erreicht.

In Abbildung 38 ist die Dauer für die einzelnen Phasen vom Import aufgeschlüsselt. Am längsten wird für die Analyse der Segmente benötigt, weil für jeden Punkt die benötigten Eigenschaften berechnet werden. Bei den ALS-KAxx-Daten ist der Boden sehr eben, wodurch die Baumkronen ähnliche Höhen haben. Dadurch haben viele Punkte die gleiche Höhe und die Segmentierung dauert länger.

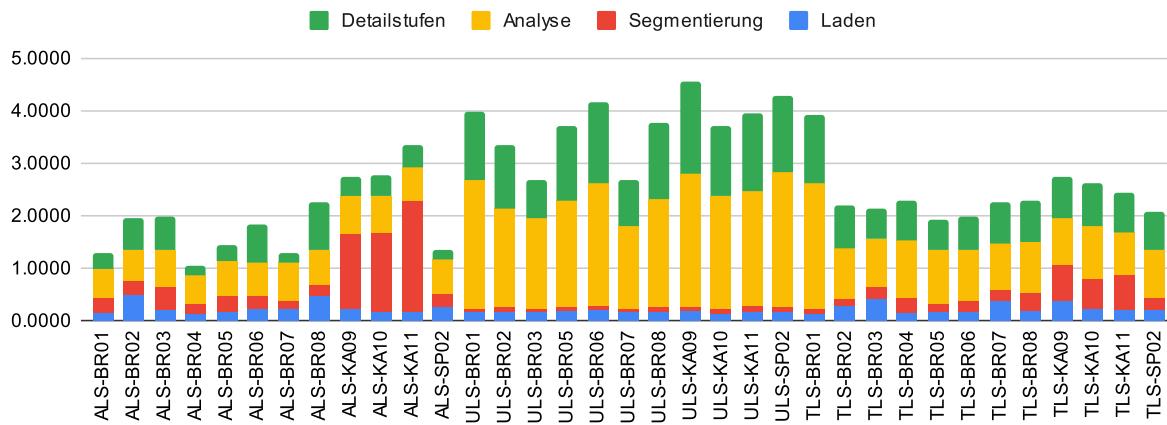


Abbildung 38: Dauer für die einzelnen Importphasen in  $\mu\text{s}$  pro Punkt.

### 8.3 Segmentierung von Waldstücken

Ein Beispiel für eine Segmentierung ist in Abbildung 39 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.

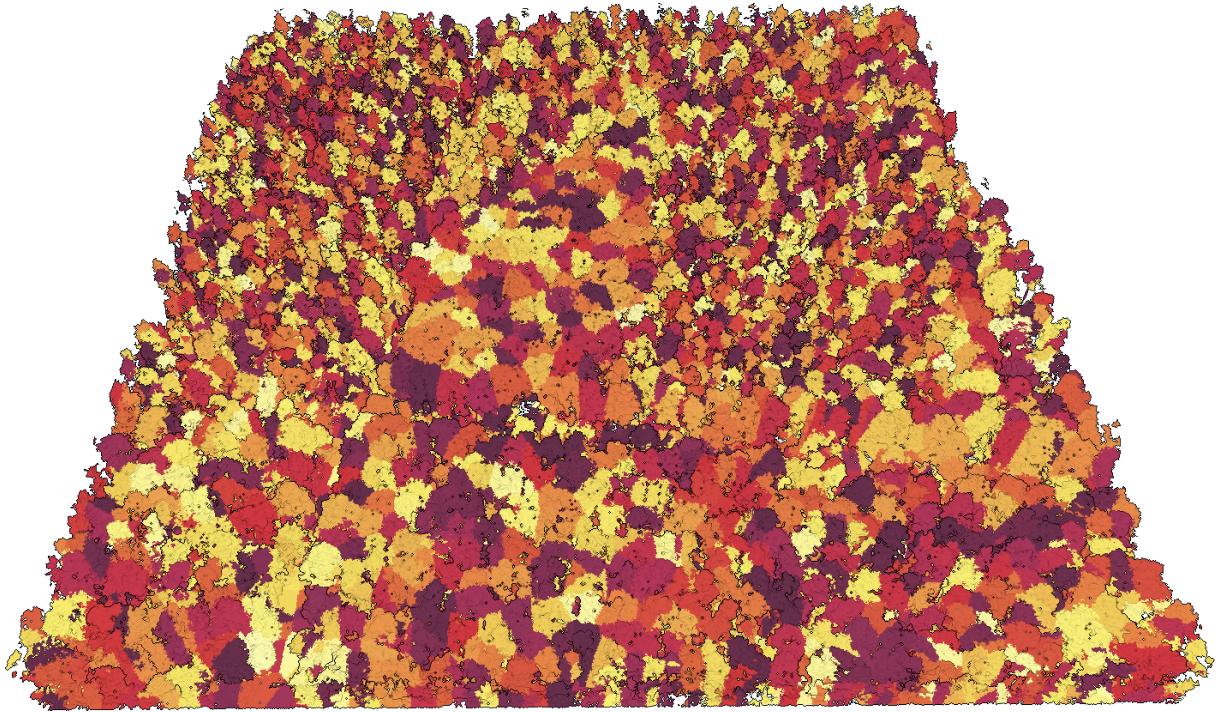


Abbildung 39: Segmentierung von einer Punktfolge.

Punkte, welche zu keinem Baum gehören, werden trotzdem zu den Segmenten zugeordnet. Bei Bereichen ohne Bäume entstehen dadurch Segmente wie in Abbildung 40. 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.

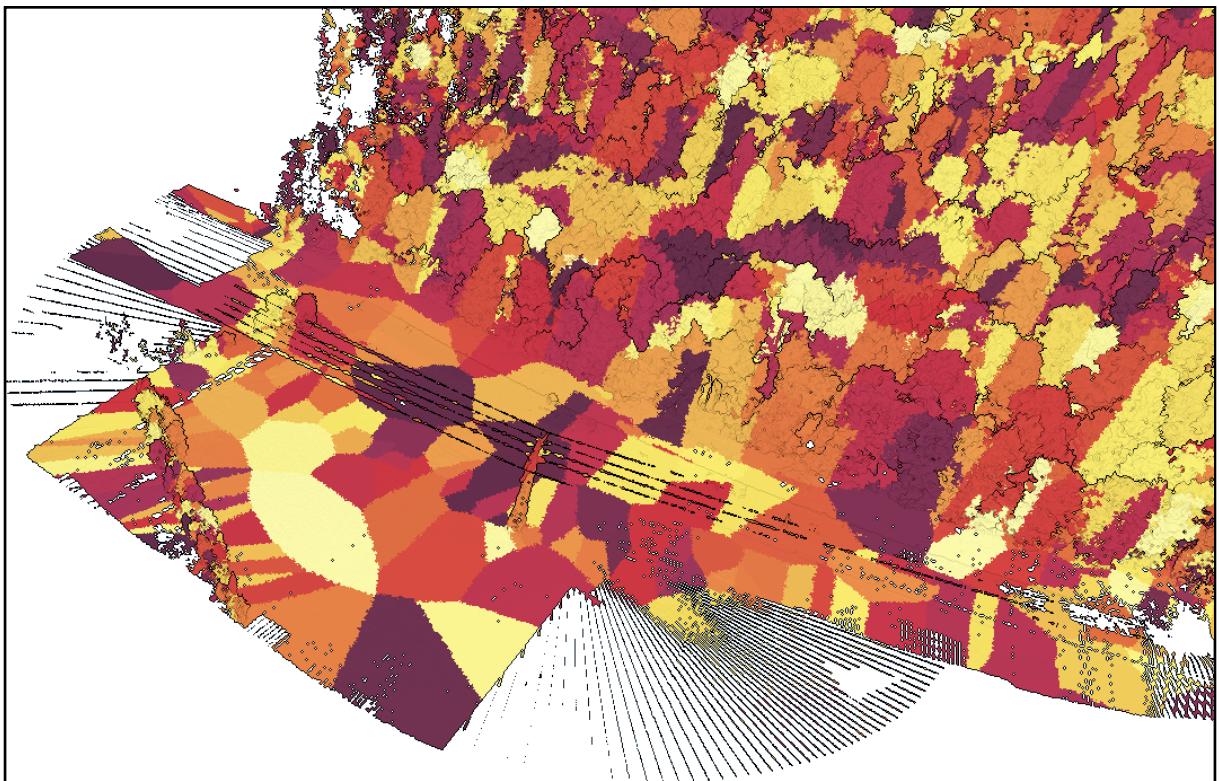


Abbildung 40: Segmente bei Gebieten ohne Bäume.

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 41 zu sehen.



(a) Alle Segmente

(b) Segment einzeln

Abbildung 41: Segmentierungsfehler bei Baumspitzen.

## 8.4 Triangulierung

Ein Beispiel für die Triangulation ist in Abbildung 42 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.

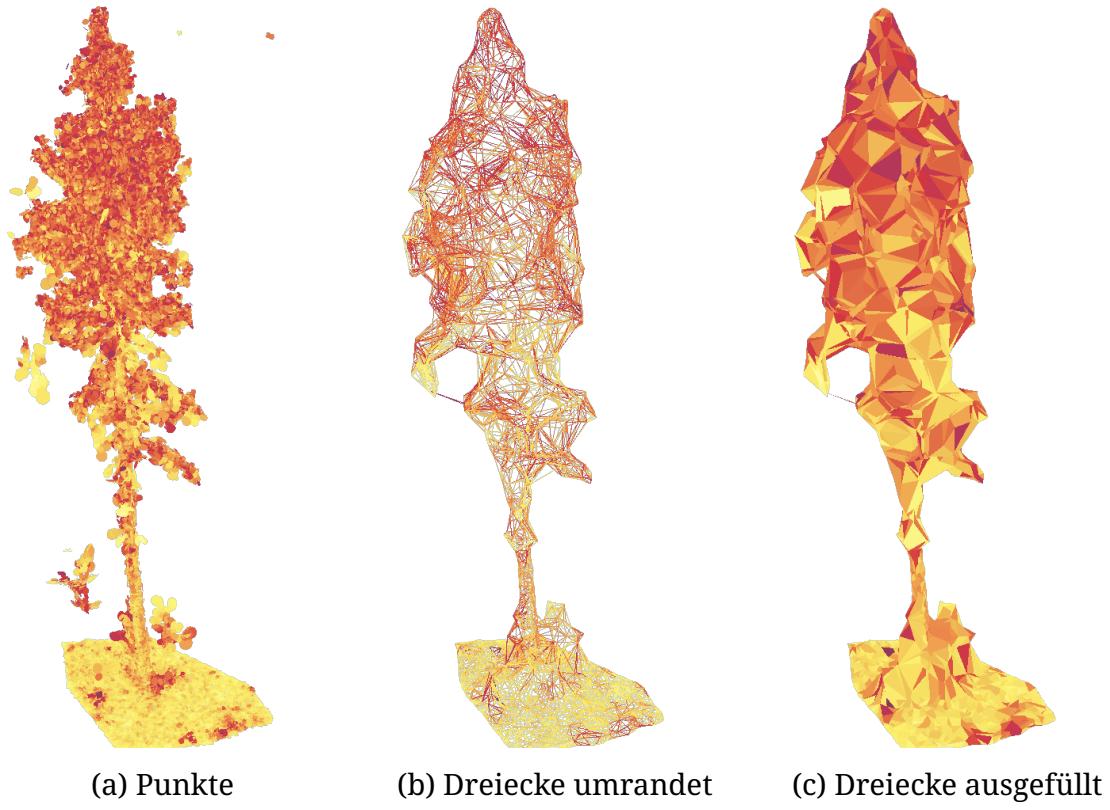


Abbildung 42: Beispiel für eine Triangulierung von einem Baum mit  $\alpha = 1m$ .

## 8.5 Visualisierung

In Abbildung 43 ist die benötigte Renderzeit für die Beispiele aus Abbildung 44 gegeben.

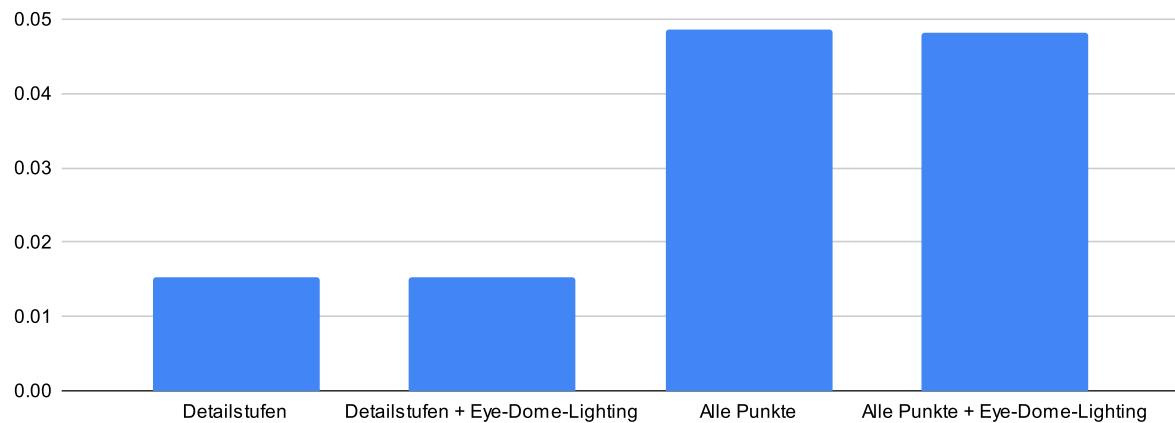


Abbildung 43: Renderzeit in Sekunden für einen Datensatz mit 59 967 504 Punkten.

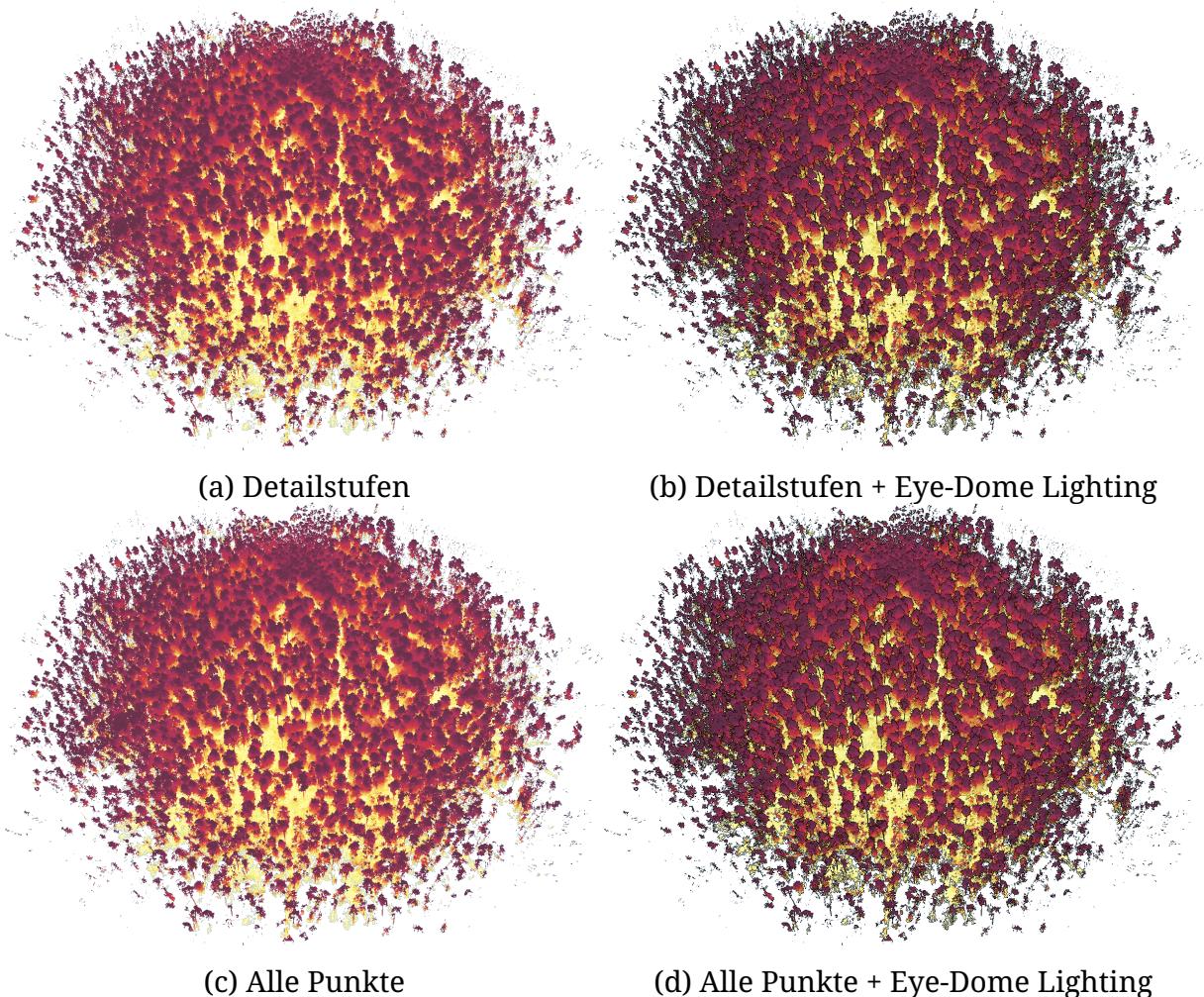


Abbildung 44: Visualisierung von einem Datensatz mit 59 967 504 Punkten.

Die Detailstufen ermöglichen eine Visualisierung vom kompletten Datensatz in Echtzeit ohne einen sichtbaren Detailverlust. Der zusätzliche Speicherbedarf für die Detailstufen ist in Abbildung 45 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.

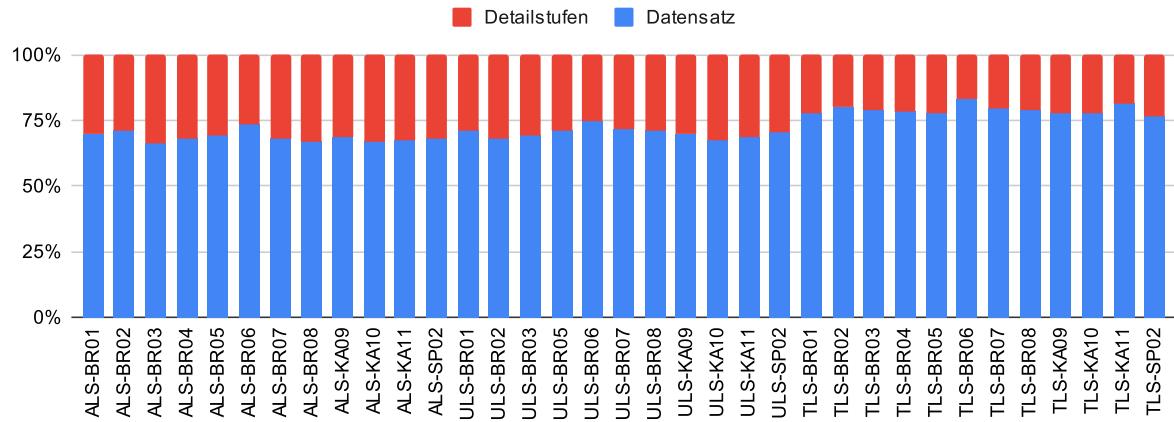


Abbildung 45: Speicherbedarf für die Punkte vom Datensatz und der Detailstufen.

## 8.6 Analyse von Bäumen

### 8.6.1 Punkteigenschaften

In Abbildung 46 und Abbildung 47 ist ein Segment basierend auf den berechneten Eigenschaften 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.

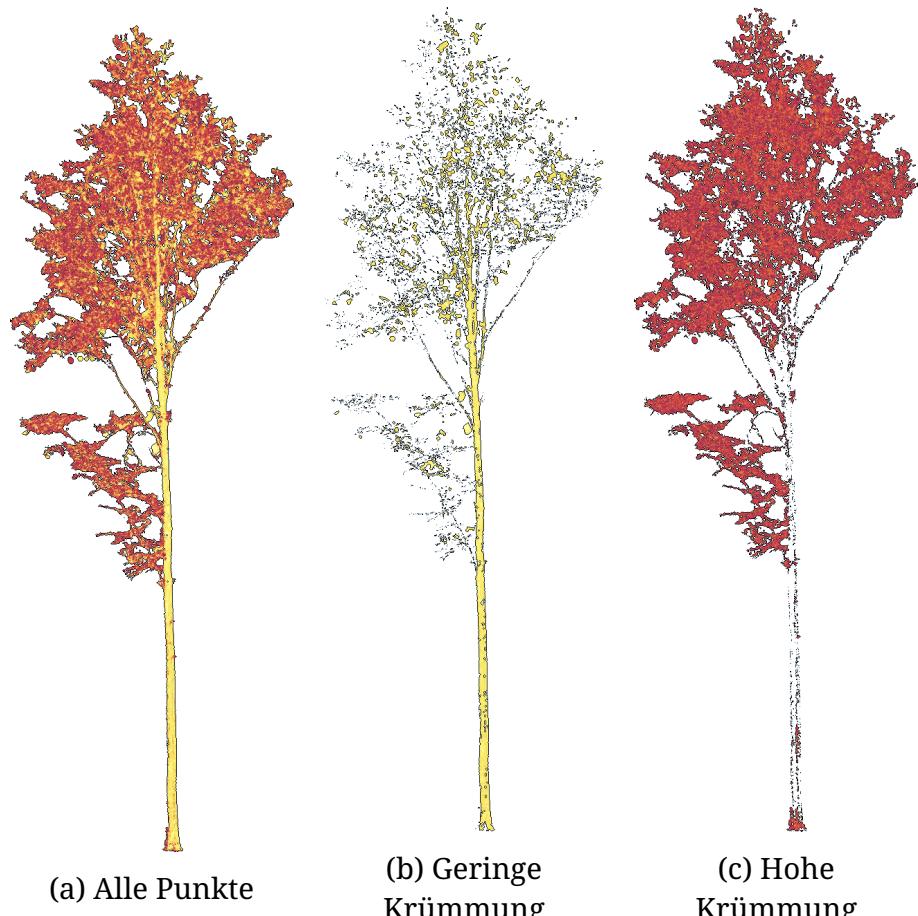


Abbildung 46: Punktfolke basierend auf der Krümmung eingefärbt.

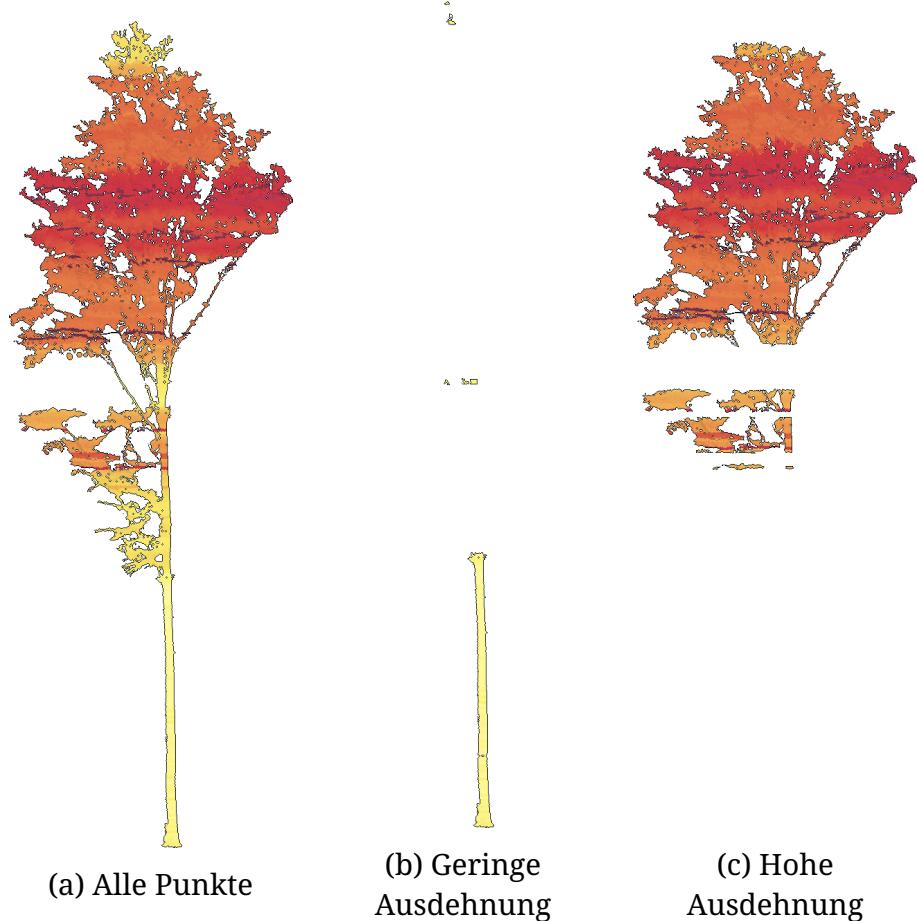


Abbildung 47: Punktwolke basierend auf der Ausdehnung eingefärbt.

### 8.6.2 Baumeigenschaften

In Abbildung 48 werden die korrekten gemessenen Werte für die Baumeigenschaften mit den aus den Punktwolken algorithmisch berechneten Werten verglichen. Entlang der x-Achse sind die gemessenen Werte und entlang der y-Achse sind die berechneten Werte.

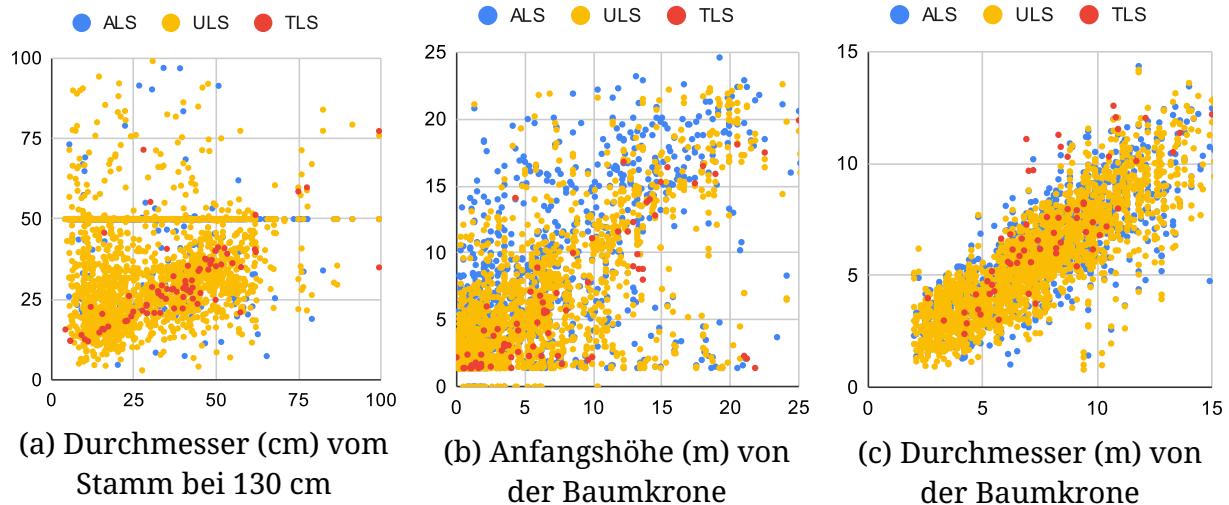


Abbildung 48: Vergleich zwischen den gemessenen Baumeigenschaften und den berechneten Werten.

Für den Vergleich wurden die einzelnen Bäume aus dem Datensatz verwenden. Die Positionen der Bäume wurde mit einer Kombination von den ALS-, ULS- und TLS-Daten und einer manuellen Unterteilung der Punkte berechnet. Danach wurden alle Punktewolken der Waldstücke mit den Baumpositionen unterteilt, wodurch besonders für die ALS- und ULS-Daten für manche Bäume nur wenig Punkte bekannt sind (H. Weiser u. a. 2022). Eine Visualisierung vom Unterschied ist in Abbildung 49 gegeben.

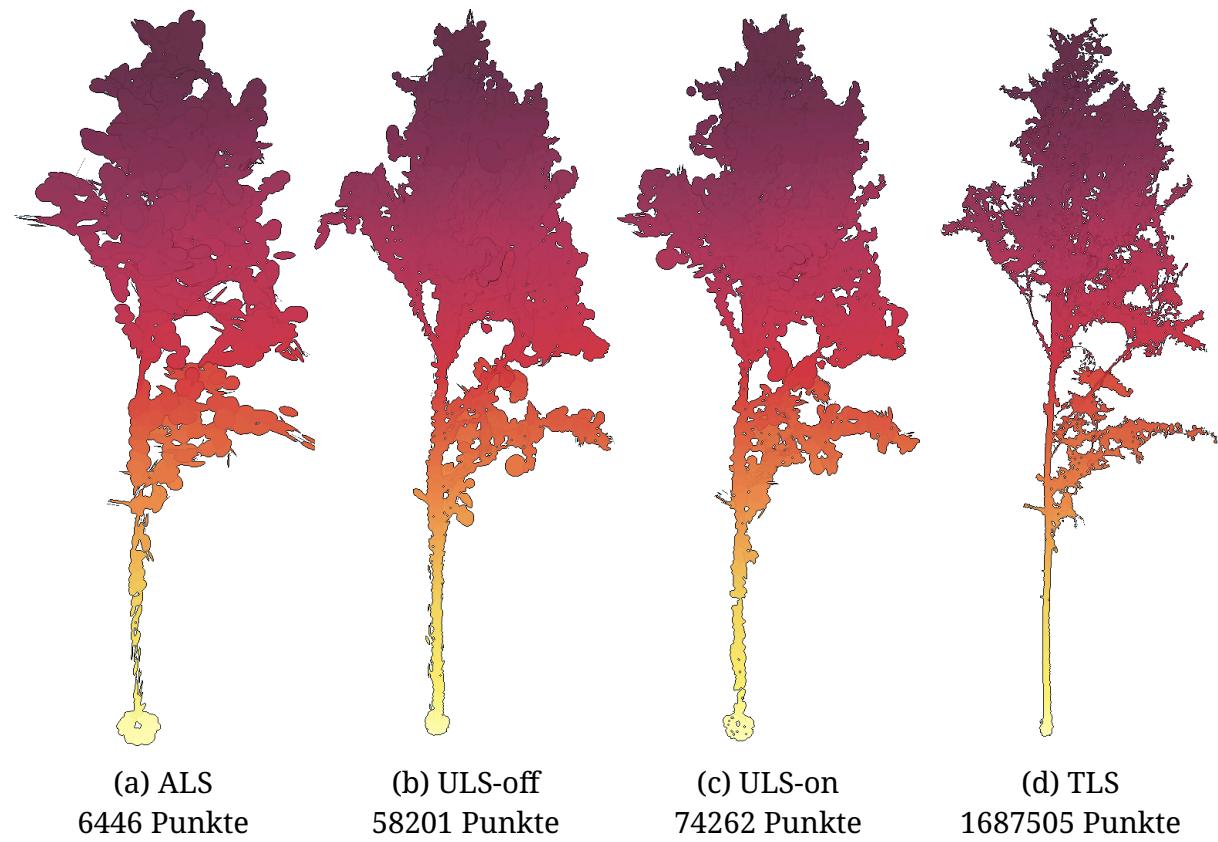


Abbildung 49: Vergleich zwischen den unterschiedlichen Daten für den gleichen Baum.

Die Berechnung vom Stammdurchmesser funktioniert für die TLS-Daten am besten. Der Stamm wird als Kreis approximiert, wodurch die berechneten Werte kleiner als die gemessenen Werte sind. Bei den ULS Daten ist eine Korrelation zu sehen, aber die Ergebnisse weichen stark von den gemessenen Werten ab. Mit den ALS-Daten kann der Stammdurchmesser nicht berechnet werden. Bei vielen Punktwolken waren zu wenig Datenpunkte im verwendeten Bereich für die Berechnung, wodurch der Standardwert von 50 cm verwendet wurde.

Auch die Berechnung von der Anfangshöhe der Baumkrone funktioniert mit den TLS-Daten am besten. Mit den ALS- und den ULS-Daten ist eine Approximation möglich, wodurch die Ergebnisse bei den ALS Daten weiter von den gemessenen Werten schwanken.

Bei der Berechnung vom Durchmesser der Baumkrone sind die ALS-, ULS- und TLS-Daten geeignet. Durch die Approximation der Baumform als Kreis ist der berechnete Wert kleiner als der gemessene Wert. Bei den gemessenen Werten würde der Durchschnitt zwischen der größten Ausdehnung und zugehörigen orthogonalen Ausdehnung bestimmt.

## 8.7 Fazit

Die Software ermöglicht den Übergang von den Punktdaten ohne weitere Informationen zu einer interaktiven Visualisierung vom Waldstück. Dadurch ist ein Überblick über das gescannte Waldstücke und einzelne Bäume möglich. 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 ist.

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 vom Durchmesser funktioniert für die meisten Bereiche vom Baum, wird aber durch Punkte, welche nicht zum Baum gehören stark beeinflusst.

Die charakteristischen Eigenschaften vom Baum können mit den genaueren TLS-Daten abgeschätzt werden, haben aber noch systematische Fehler, welche das Ergebnis verfälschen. Mit den ALS- und ULS-Daten werden größere Gebiete abgedeckt, dafür sind die berechneten Eigenschaften ungenauer. Besonders die Berechnung vom Durchmesser vom Stamm ist nur mit den TLS-Daten möglich.

Die Triangulierung berechnet ein Mesh für die Segmente, welches von der Visualisierung angezeigt werden kann. Die Visualisierung kann die berechneten Daten ohne Probleme visualisieren. Durch die Detailstufen können auch größere Datenmengen interaktiv angezeigt werden.

## **8.8 Ausblick**

Momentan werden die ermittelten Daten nur für die Visualisierung verwendet. In der `project.json` sind die charakteristischen Eigenschaften für die Segmente gespeichert, aber auch Daten für die Visualisierung. Durch eine Trennung wird eine automatische Weiterverarbeitung der berechneten Eigenschaften erleichtert.

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 und die Zeit verringert, ab der die ersten Ergebnisse sichtbar sind.

Der Importer unterstützt momentan nur Dateien im LASzip-Format, wodurch Daten in anderen Formaten nicht verwendet werden können oder zuerst konvertiert werden müssen. Durch andere Importformate kann die Verwendung erweitert werden.

Die Visualisierung kann Punktwolken auch aus anderen Quellen als Waldstücke anzeigen, ist aber stark an diesen Verwendungszweck angepasst. Durch einen zusätzlicher Importer, welcher für beliebige Datensätze geeignet ist kann die Visualisierung vielseitiger benutzt werden.

## 9 Appendix

### 9.1 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 6: 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 7: Prozessoreigenschaften

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

Tabelle 8: Grafikkarteneigenschaften

Größe	2 × 8 GiB
Taktrate	2133 MHz

Tabelle 9: 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 10: Sequenzielle Lese- und Schreibgeschwindigkeit der Festplatte mit 4 KiB Blöcken für unterschiedliche Dateigrößen.

## 9.2 Messwerte vom Import

Datensatz	Datei	Daten (Punkte)	Segmente (Punkte)	Detailstufen (Punkte)
ALS-BR01	ALS-on_BR01_2019-07-05_300m	12531758	12508612	5383527
ALS-BR02	ALS-on_BR02_2019-07-05_200m	5469363	5457090	2248915
ALS-BR03	ALS-on_BR03_2019-07-05_300m	12284905	12235917	6268300
ALS-BR04	ALS-on_BR04_2019-07-05_140m	2638216	2629231	1225097
ALS-BR05	ALS-on_BR05_2019-07-05_300m	13401630	13362033	5904834
ALS-BR06	ALS-on_BR06_2019-07-05_200m	6269955	6260421	2264327
ALS-BR07	ALS-on_BR07_2019-07-05_140m	3196739	3183756	1494056
ALS-BR08	ALS-on_BR08_2019-07-05_140m	2871871	2860173	1437912
ALS-KA09	ALS-on_KA09_2019-07-05_300m	14536939	14531527	6574613
ALS-KA10	ALS-on_KA10_2019-07-05_300m	13413120	13400728	6616077
ALS-KA11	ALS-on_KA11_2019-07-05_300m	14840932	14834281	7197589
ALS-SP02	ALS-on_SP02_2019-07-05_140m	2810265	2803582	1330103
ULS-BR01	ULS-off_BR01_2019-12-04	52179533	52149914	21465668
ULS-BR02	ULS-on_BR02_2019-08-24	41555427	41431426	19737224
ULS-BR03	ULS-on_BR0308_2019-08-24	83344946	83193255	37410691
ULS-BR05	ULS-off_BR05_2020-03-26	52169435	52118747	21484710
ULS-BR06	ULS-off_BR06_2020-03-27	60182129	60117592	20480701
ULS-BR07	ULS-on_BR07_2019-08-24	46833772	46708265	18287517
ULS-BR08	ULS-off_BR08_2020-03-31	66483369	66432982	26880086
ULS-KA09	ULS-on_KA09_2019-09-13	50863340	50723815	22030809
ULS-KA10	ULS-on_KA10_2019-09-13	48167924	48030861	23080707
ULS-KA11	ULS-on_KA11_2019-09-06	58549518	58459850	26676462
ULS-SP02	ULS-on_SP02_2019-09-03	30811866	30761609	12827262
TLS-BR01	TLS-on_BR01_2019-07-03	349902555	349880861	101402285
TLS-BR02	TLS-on_BR02_2019-06-13	206306919	206277122	50951743
TLS-BR03	TLS-on_BR03_2019-06-04_B	71067156	71056700	18813597
TLS-BR04	TLS-on_BR04_2019-07-04	326741799	326718896	91369208
TLS-BR05	TLS-on_BR05_2019-07-10	418794802	418764606	119672624
TLS-BR06	TLS-on_BR06_2019-07-22_B	124075064	124048349	25323813
TLS-BR07	TLS-on_BR07_2019-07-19_B	308965997	308937292	80092920
TLS-BR08	TLS-on_BR08_2019-07-23	417738648	417716948	111295554
TLS-KA09	TLS-on_KA09_2019-08-20_A	261723907	261705346	74651193
TLS-KA10	TLS-on_KA10_2019-07-30_B	283197972	283176980	81713068
TLS-KA11	TLS-on_KA11_2019-09-03_B	273795041	273770543	61803916
TLS-SP02	TLS-on_SP02_2019-08-23_B	252267714	252246056	77190719

Tabelle 11: Messwerte (Verwendete Datei und Punktanzahl).

Datensatz	Segmente (Anzahl)	Laden (Sekunden)	Segmentierung (Sekunden)	Analyse (Sekunden)	Detailstufen (Sekunden)
ALS-BR01	4390	1.861	3.446	7.069	3.796
ALS-BR02	2634	2.758	1.387	3.223	3.244
ALS-BR03	5740	2.389	5.456	8.657	7.830
ALS-BR04	1596	0.395	0.452	1.439	0.493
ALS-BR05	4911	2.237	4.222	8.848	3.744
ALS-BR06	2274	1.389	1.489	3.975	4.679
ALS-BR07	1294	0.760	0.462	2.341	0.536
ALS-BR08	1275	1.340	0.644	1.846	2.656
ALS-KA09	4585	3.414	20.675	10.224	5.572
ALS-KA10	4499	2.287	20.146	9.478	5.303
ALS-KA11	5835	2.534	31.228	9.595	6.172
ALS-SP02	1456	0.698	0.695	1.842	0.532
ULS-BR01	2712	14.805	6.842	50.851	42.392
ULS-BR02	5248	16.693	9.850	37.769	23.975
ULS-BR03	7266	11.356	24.106	92.917	61.733
ULS-BR05	4170	9.181	8.120	52.270	31.355
ULS-BR06	5909	9.932	13.340	57.183	38.787
ULS-BR07	6227	17.608	10.173	41.496	36.861
ULS-BR08	6036	12.952	21.606	64.969	51.375
ULS-KA09	6007	19.923	34.091	44.833	39.613
ULS-KA10	5638	10.368547	28.78788	46.987415	40.57933
ULS-KA11	5834	11.913675	39.09122	47.891563	42.77556
ULS-SP02	2936	6.447726	7.2568164	27.668728	22.448013
TLS-BR01	1122	56.536	25.621	850.015	454.944
TLS-BR02	2955	37.027	18.922	381.628	248.833
TLS-BR03	714	11.495	5.243	121.244	52.736
TLS-BR04	1813	59.840	23.269	660.753	469.446
TLS-BR05	2967	81.245	38.454	970.322	657.404
TLS-BR06	1898	18.813	8.927	194.271	109.483
TLS-BR07	2627	51.777	25.080	635.522	452.381
TLS-BR08	1937	78.780	33.562	1,060.721	734.373
TLS-KA09	1261	38.988	22.069	562.831	346.010
TLS-KA10	1380	47.884	33.696	613.597	422.592
TLS-KA11	1824	43.516	26.665	706.204	393.059
TLS-SP02	1558	36.764	19.562	605.249	327.259

Tabelle 12: Messwerte (Anzahl Segmente und Importgeschwindigkeit).

## 9.3 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 50 gegeben.

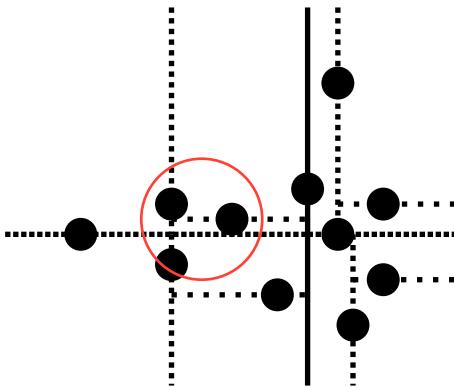


Abbildung 50: KD-Baum für Punkte in 2D. Für jede Unterteilung ist die Trenn-gerade gepunkteter gezeichnet. Weil der rote Kreis vollständig auf einer Seite der ersten Unterteilung ist, müssen die Punkte auf der anderen Seite nicht betrachtet werden.

### 9.3.1 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 (Hoare 1961) 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.

### **9.3.2 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.

### **9.3.3 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  $\infty$  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.

### **9.3.4 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.

## **9.4 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.

### **9.4.1 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 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 51 und Abbildung 52 sind Beispiele in 2D und 3D gegeben.

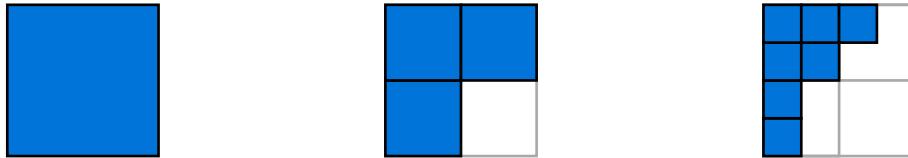


Abbildung 51: Unterschiedliche Stufen von einem Quadtree.

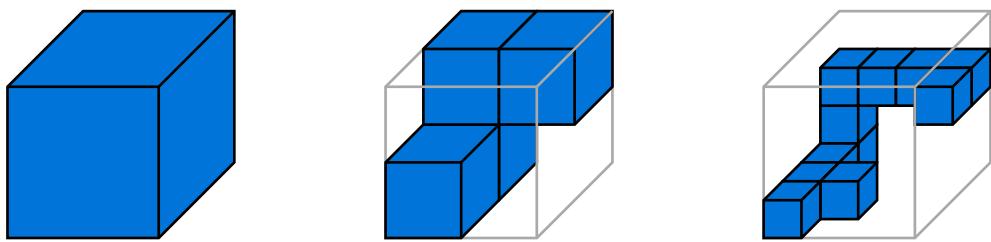


Abbildung 52: Unterschiedliche Stufen von einem Octree.

#### 9.4.2 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 Punktwölken 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.

**Greedy-Algorithmus** ist eine Kategorie von Algorithmen, bei denen das Ergebnis schrittweise berechnet wird. Bei jedem Schritt wird mit den momentanen Informationen die beste Entscheidung getroffen, wodurch das Ergebnis schnell, aber meist nicht global optimal berechnet wird.

## Bibliographie

- Burt, Andrew, Mathias Disney, und Kim Calders. 2019. „Extracting individual trees from lidar point clouds using treeseg“. *Methods in Ecology and Evolution* 10 (3): 438–45. <https://doi.org/10.1111/2041-210X.13121>
- Disney, Mathias. 2019. „Terrestrial LiDAR: a three-dimensional revolution in how we look at trees“. *New Phytologist* 222 (4): 1736–41. <https://doi.org/10.1111/nph.15517>
- Donager, Jonathon J, Andrew J Sánchez Meador, und Ryan C Blackburn. 2021. „Adjudicating perspectives on forest structure: how do airborne, terrestrial, and mobile lidar-derived estimates compare?“. *Remote Sensing* 13 (12): 2297–98
- Graham, Lewis. 2012. „The LAS 1.4 specification“. *Photogrammetric engineering and remote sensing* 78 (2): 93–102
- Guo, Jianwei, Shibiao Xu, Dong-Ming Yan, Zhanglin Cheng, Marc Jaeger, und Xiaopeng Zhang. 2020. „Realistic Procedural Plant Modeling from Multiple View Images“. *IEEE Transactions on Visualization and Computer Graphics* 26 (2): 1372–84. <https://doi.org/10.1109/TVCG.2018.2869784>
- Hackenberg, Jan, Heinrich Spiecker, Kim Calders, Mathias Disney, und Pasi Raumonen. 2015. „SimpleTree —An Efficient Open Source Tool to Build Tree Models from TLS Clouds“. *Forests* 6 (11): 4245–94. <https://doi.org/10.3390/f6114245>
- Hoare, C. A. R. 1961. „Algorithm 65: Find“. *Commun. ACM* 4 (7): 321–22. <https://doi.org/10.1145/366622.366647>
- Hug, C., Peter Krzystek, und Wolfgang Fuchs. 2004. „Advanced Lidar Data Processing with Lastools“. In . <https://api.semanticscholar.org/CorpusID:14167994>
- Isenburg, Martin. 2013. „LASzip: lossless compression of LiDAR data“. *Photogrammetric engineering and remote sensing* 79 (2): 209–17
- Liu, Yanchao, Jianwei Guo, Bedrich Benes, Oliver Deussen, Xiaopeng Zhang, und Hui Huang. 2021. „TreePartNet: neural decomposition of point clouds for 3D tree reconstruction“. *ACM Trans. Graph.* 40 (6). <https://doi.org/10.1145/3478513.3480486>
- Majercik, Alexander, Cyril Crassin, Peter Shirley, und Morgan McGuire. 2018. „A Ray-Box Intersection Algorithm and Efficient Dynamic Voxel Rendering“. *Journal of Computer Graphics Techniques (JCGT)* 7 (3): 66–81. <http://jcgta.org/published/0007/03/04/>
- Schütz, Markus, Stefan Ohrhallinger, und Michael Wimmer. 2020. „Fast Out-of-Core Octree Generation for Massive Point Clouds“. *Computer Graphics Forum* 39 (7): 155–67. <https://doi.org/10.1111/cgf.14134>
- Suzuki, Taro, Shunichi Shiozawa, Atsushi Yamaba, und Yoshiharu Amano. 2021. „Forest Data Collection by UAV Lidar-Based 3D Mapping: Segmentation of Individual Tree Information from 3D Point Clouds“. *International Journal of Automation Technology* 15: 313–23. <https://doi.org/10.20965/ijat.2021.p0313>
- Torr, P.H.S., und A. Zisserman. 2000. „MLESAC: A New Robust Estimator with Application to Estimating Image Geometry“. *Computer Vision and Image Understanding* 78 (1): 138–56. <https://doi.org/10.1006/cviu.1999.0832>

Weiser, H. u. a. 2022. „Individual tree point clouds and tree measurements from multi-platform laser scanning in German forests“. *Earth System Science Data* 14 (7): 2989–3012. <https://doi.org/10.5194/essd-14-2989-2022>

Weiser, Hannah u. a. 2022. „Terrestrial, UAV-borne, and airborne laser scanning point clouds of central European forest plots, Germany, with extracted individual trees and manual forest inventory measurements“. PANGAEA. 2022. <https://doi.org/10.1594/PANGAEA.942856>