

## TODOs

1: Abstrakt .....	3
2: Bilder .....	1
3: Motivation .....	1
4: Referenzen zu späteren Abschnitten .....	1
5: Mehr Überblick .....	1
6: Bild .....	3
7: Bild .....	3
8: Bild .....	3
9: Bild überschneiden gewachsene Bereiche .....	4
10: Bild getrennte Bereiche gerade .....	4
11: Bild .....	4
12: Mehr Baumeigenschaften .....	7
13: Baumeigenschaften + ? → Segmente .....	7
14: Segmente + Eigenschaften + ? → Klassifizierung? .....	7
15: Kugel für Dreieck .....	8
16: Auswahl $\alpha$ .....	13
17: Ergebnisse .....	13
18: Vergleich $\alpha$ .....	13
19: Triangle Strip für Quadrat .....	17
20: Messwerte .....	17
21: Kopiert vom Fachpraktikum .....	22
22: Auswahl der Detailstufen .....	23
23: Frustrum-Culling .....	23
24: Ergebnisse .....	25
25: Triangulierung Ergebnisse .....	25
26: Berechnung Ergebnisse .....	25
27: Visualisierung Ergebnisse .....	25
28: Fazit .....	26

# Masterarbeit

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

**Name:** Anton Wetzel

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

Matrikelnummer: 60451

Studiengang: Informatik Master

**Betreuer:** Tristan Nauber

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

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

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

Fachgebiet: Data-intensive Systems and Visualizati-  
on Group

Datum: 18.01.2024



## **Abstrakt**

Todo: Abstrakt

# **Inhaltsverzeichnis**

## **Abstrakt**

### **I. Überblick**

1. Motivation
2. Ablauf
  - 2.1. Separierung in Segmente
  - 2.2. Segmente verarbeiten
  - 2.3. Berechnung vom Octree
3. Stand der Technik

### **II. Segmentierung in Bäume**

1. Ablauf
2. Bereiche bestimmen
3. Koordinaten bestimmen
4. Punkte zuordnen

### **III. Analyse von Segmenten**

1. Eigenschaften
  - 1.1. Nachbarschaft
  - 1.2. Krümmung
  - 1.3. Punkthöhe
  - 1.4. Ausdehnung
2. Segmentierung von einem Baum
3. Eigenschaften für Visualisierung
  - 3.1. Normale
  - 3.2. Punktgröße
4. Baumart

### **IV. Triangulierung**

1. Ziel
2. Ball-Pivoting-Algorithmus
  - 2.1. Überblick
  - 2.2.  $\alpha$ -Kugel für ein Dreieck
  - 2.3. Ablauf
  - 2.4. Startdreieck bestimmen
  - 2.5. Triangulierten Bereich erweitern
  - 2.6. Komplettes Segment triangulieren
  - 2.7. Vorauswahl
  - 2.8. Auswahl von  $\alpha$
3. Ergebnisse

### **V. Visualisierung**

1. Technik
2. Punkt

- 2.1. Basis
- 2.2. Vergleich zu Quadrat als Basis
- 2.3. Instancing
- 2.4. Kreis
- 3. Ausgewählte Eigenschaft
- 4. Segmente
  - 4.1. Auswahl
  - 4.2. Anzeige
- 5. Eye-Dome-Lighting
- 6. Detailstufen
  - 6.1. Berechnung der Detailstufen
  - 6.2. Auswahl der Detailstufen?
- 7. Filtern mit dem Kamerafrustrum
- 8. Kamera/Projektion
  - 8.1. Kontroller

## **VI. Ergebnisse**

- 1. Testdaten
- 2. Triangulierung
- 3. Berechnung
- 4. Visualisierung

## **VII. Fazit**

## **VIII. Appendix**

- 1. KD-Baum
  - 1.1. Konstruktion
  - 1.2. Suche mit festem Radius
  - 1.3. Suche mit fester Anzahl
  - 1.4. Schnelle Suche
- 2. Baum (Datenstruktur)
  - 2.1. Konstruktion
  - 2.2. Suchanfrage

## **Glossar**

## **Bibliographie**

# I. Überblick

Todo: Bilder

## 1. Motivation

Todo: Motivation

## 2. Ablauf

Der Import wird in mehreren getrennten Phasen durchgeführt. Dabei wird die Arbeit für eine Phase so weit wie möglich parallelisiert.

Todo: Referenzen zu späteren Abschnitten

### 2.1. Separierung in Segmente

Für jeden Punkt wird bestimmt, zu welchem Segment er gehört. Dafür werden die Punkte in horizontale Scheiben unterteilt und von Oben nach Unten die Scheiben in Bereiche unterteilt. Die Punkte in den Bereichen werden dann den Segmenten zugeordnet. Der vollständige Ablauf ist in Abschnitt II gegeben.

### 2.2. Segmente verarbeiten

Für jedes Segment werden die benötigten Eigenschaften für die Visualisierung berechnet. Dabei werden Eigenschaften spezifisch für jeden Punkt und Eigenschaften für das gesamte Segment bestimmt. In Abschnitt III-1 und Abschnitt III-3 sind die Schritte ausgeführt.

Die Triangulierung wird für die Segmente einzeln durchgeführt. Der Algorithmus ist in Abschnitt IV gegeben.

Die fertigen Segmente werden einzeln abgespeichert, dass diese separat angezeigt werden können. Zusätzlich wird ein Octree mit allen Segmenten kombiniert erstellt.

### 2.3. Berechnung vom Octree

Für alle Branch-Knoten im Octree wird mit den Kinderknoten eine vereinfachte Punktwolke als Detailstufe für das Anzeigen berechnet. Die Baumstruktur und alle Knoten mit den zugehörigen Punkten werden abgespeichert. Der Ablauf ist in Abschnitt V-6.1 erklärt.

Todo: Mehr Überblick

## 3. Stand der Technik

Punktwolken können mit unterschiedlichen Lidar Scanverfahren aufgenommen werden. Aufnahmen vom Boden oder aus der Luft bieten dabei verschiedene Vor- und Nachteile.

le [1]. Bei einem Scan von Boden aus, kann nur eine kleinere Fläche abgetastet werden, dafür mit erhöhter Genauigkeit, um einzelne Bäume genau zu analysieren [2]. Aus der Luft können größere Flächen erfasst werden, wodurch Waldstücke aufgenommen werden können, aber die Datenmenge pro Baum ist geringer [3].

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

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

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

## II. Segmentierung in Bäume

### 1. Ablauf

Die Punkte werden in gleichbreite parallele Scheiben entlang der Höhe unterteilt. Danach werden die Scheiben von Oben nach Unten einzeln verarbeitet, um die Segmente zu bestimmen. Dafür werden die Punkte in einer Scheibe zu Bereichen zusammengefasst. Für die Bereiche die zugehörigen Koordinaten vom Mittelpunkte bestimmt und jeder Punkte wird zur nächsten Koordinate zugeordnet.

### 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 in einem Bereich ist, ändert der Punkt nicht den Bereich. Ist der Punkt näher als den Mindestabstand zu einem der Bereiche, so wird der Bereich erweitert. Ist der Punkt von allen bisherigen Bereichen entfernt, so wird ein neuer Bereich angefangen.

Todo: Bild

Die Bereiche sind als Liste gespeichert, wobei für jeden Bereich die Eckpunkte als Liste gegeben sind. Die Eckpunkte sind dabei sortiert, dass für einen Eckpunkt der nächste Punkt entlang der Umrandung der nächste Punkt in der Liste ist. Für den letzten Punkt ist der erste Punkt in der Liste der nächste Punkt.

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

Todo: Bild

Um einen Punkt zu einem Bereich hinzuzufügen, werden alle Kanten entfernt, bei denen der Punkt auf der Seite außerhalb liegt, entfernt und zwei neue Kanten zum Punkt werden hinzugefügt. Dafür werden die beiden Eckpunkte gesucht, bei denen eine zugehörige Kante entfernt wird und die andere nicht. Um die Kanten zwischen den Punkten zu entfernt, werden alle Punkte zwischen den beiden Punkten entfernt und stattdessen der neue Punkt eingefügt, um die beiden neuen Kanten zu ergänzen.

Todo: Bild

Nachdem alle Punkte zu den Bereichen hinzugefügt würden, können Bereiche so gewachsen sein, dass Bereiche sich überlappen. Um diese zu verbinden wird wiederholt überlappende Bereiche gesucht und alle Punkte von einem Bereich zum anderen hinzugefügt. Um zu überprüfen, ob Bereiche sich überlappen, wird für einen der Bereiche alle Kanten überprüft, ob der andere Bereich vollständig außerhalb der Kante liegt. Wenn alle Punkte vom anderen Bereich außerhalb der Kante liegen, trennt die Kante die Bereiche. Wenn keine trennende Kante existiert, so überlappen sich die Bereiche.

Todo: Bild überschneiden gewachsene Bereiche

Todo: Bild getrennte Bereiche gerade

### 3. Koordinaten bestimmen

Mit den Bereichen und den Koordinaten aus der vorherigen Scheibe werden die Koordinaten für die momentane Scheibe berechnet. Für die erste Scheibe wird die leere Menge als vorherigen Koordinaten verwendet. Für jeden Bereich werden die Koordinaten aus der vorherigen Scheibe gesucht, die im Bereich liegen.

Wenn keine Koordinate in dem Bereich liegt, so fängt der Bereich ein neues Segment an. Als Koordinate wird die durchschnittliche Position der Eckpunkte verwendet.

Liegt genau eine Position in Bereich, wird wieder die durchschnittliche Position als Koordinate verwendet, aber die Koordinate gehört zum gleichen Segment, zu dem die Koordinate aus der vorherigen Scheibe gehört.

Wenn mehrere Koordinaten im Bereich liegen, so werden die mehreren Koordinaten mit den zugehörigen Segmenten für die momentane Scheibe übernommen.

### 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 Segment von der Koordinate vom Bereich zugeordnet.

Todo: Bild

## III. Analyse von Segmenten

### 1. Eigenschaften

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

#### 1.1. Nachbarschaft

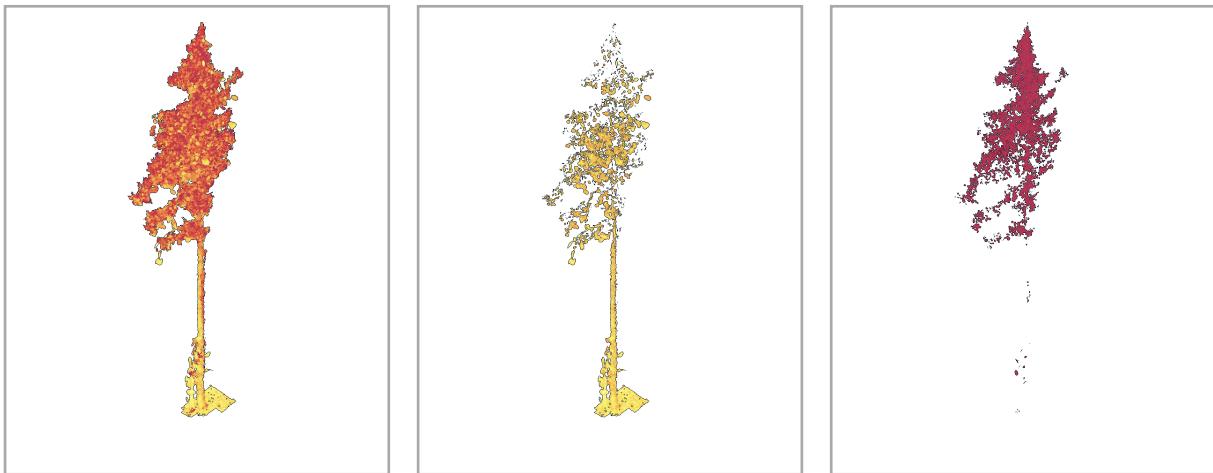
Um relevante Eigenschaften für einen Punkt zu bestimmen, werden die umliegenden Punkte benötigt. Dafür wird für alle Punkte ein **KD-Baum** erstellt. Mit diesem können effizient für einen Punkt die  $k$ -nächsten Punkte bestimmt werden.

#### 1.2. Krümmung

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

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

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



(a) Alle Punkte

(b) Geringe Varianz

(c) Hohe Varianz

Abbildung 1: Punktwolke mit Krümmung markiert.

#### 1.3. Punkthöhe

Für jeden Punkt wird die relative Höhe im Segment bestimmt. Dafür wird die Mindesthöhe  $y_{\min}$  und die Maximalhöhe  $y_{\max}$  im Segment benötigt. Die relative Höhe  $h$  für den Punkt

mit der Höhe  $p_y$  kann mit  $h = \frac{p_y - y_{\min}}{y_{\max} - y_{\min}}$  berechnet werden. Die relative Höhe liegt dabei im Bereich  $[0; 1]$ .

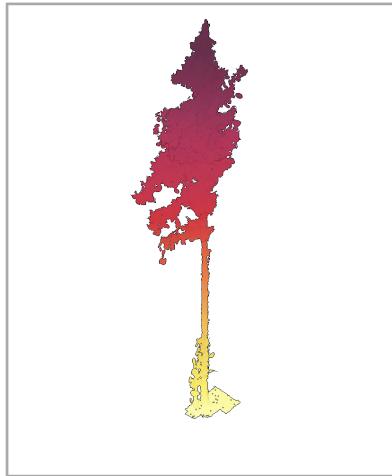
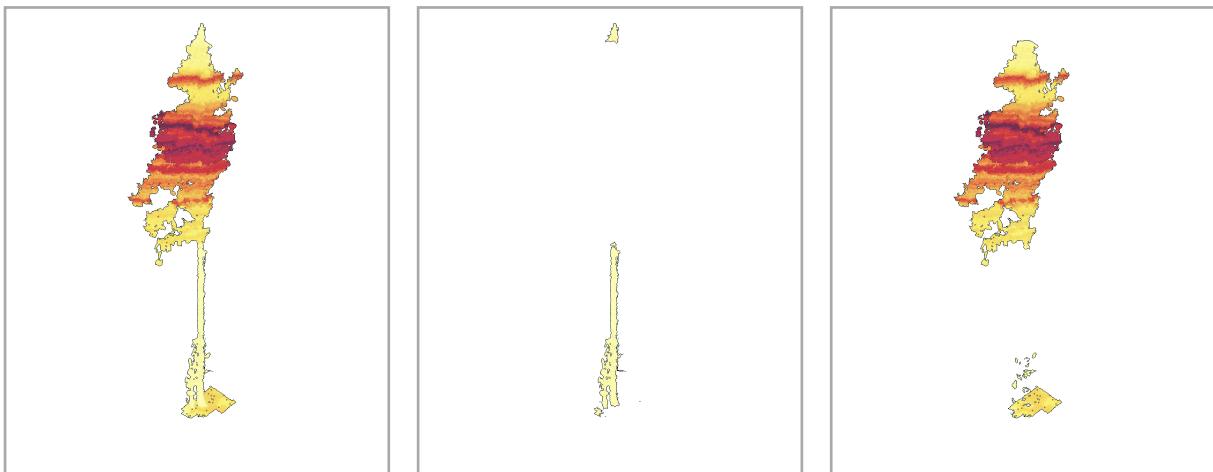


Abbildung 2: Abbildung 2: Punkt-wolke mit Höhe markiert.

#### 1.4. Ausdehnung

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

Die größte Varianz von allen Scheiben wird verwendet, um die Varianzen auf den Bereich  $[0; 1]$  zu normieren. Für jeden Punkt wird die Varianz der zugehörigen Scheibe zugeordnet.



(a) Alle Punkte

(b) Geringe Ausdehnung

(c) Hohe Ausdehnung

Abbildung 3: Punktfolke mit Varianz markiert.

Die Ausdehnung eignet sich zur Unterscheidung von Stamm und Krone. Beim Stamm sind die Punkte

Todo: Mehr Baumeigenschaften

## 2. Segmentierung von einem Baum

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

Todo: Baumeigenschaften + ? → Segmente

## 3. Eigenschaften für Visualisierung

### 3.1. Normale

Mit den Eigenvektoren aus Abschnitt III-1.2 wird die Normale für die Umgebung bestimmt. Der Eigenvektor, welcher zum kleinsten Eigenwert gehört, ist orthogonal zur Ebene mit der größten Ausdehnung.

### 3.2. Punktgröße

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

## 4. Baumart

Todo: Segmente + Eigenschaften + ? → Klassifizierung?

- out of scope?
- neural?

## IV. Triangulierung

### 1. Ziel

Eine Triangulierung ermöglicht eine Rekonstruktion der ursprünglichen Oberfläche vom eingescannten Bereich, welche weiterverarbeitet oder anzuzeigen werden kann. Die meisten Programme und Hardware sind auf das Anzeigen von Dreiecken spezialisiert, und können diese effizienter als Punkte darstellen. Die Triangulierung wird dabei für die Segmente getrennt bestimmt.

### 2. Ball-Pivoting-Algorithmus

#### 2.1. Überblick

Beim Ball-Pivoting-Algorithmus werden die Dreiecke der Oberfläche bestimmt, welche von einer Kugel mit Radius  $\alpha$  ( $\alpha$ -Kugel) erreicht werden können. Dabei berührt die Kugel die drei Eckpunkte vom Dreieck und kein weiterer Punkt aus der Punktfolge liegt in der Kugel.

In Abbildung 4 ist ein Beispiel in 2D gegeben. Dabei werden die Linien gesucht, dass der zugehörige Kreis keine weiteren Punkte enthält.

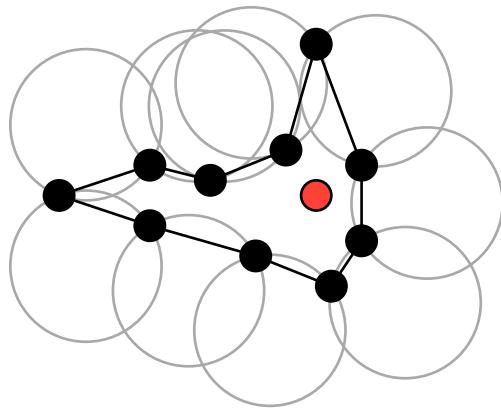


Abbildung 4: Abbildung 4: Ball-Pivoting-Algorithmus in 2D. Für die äußeren Punkte in Schwarz wird eine Oberfläche gefunden. Für den inneren Punkt in Rot kann kein Nachbar gefunden werden, weil alle zugehörigen Kreise weitere Punkte enthalten würden.

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.

#### 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 ...

Todo: Kugel für Dreieck

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

#### 2.3. Ablauf

## 2.4. Startdreieck bestimmen

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

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

Mit dem momentanen Punkt und alle möglichen 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.

## 2.5. Triangulierte 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 nun 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 5 ist ein Beispiel für die Erweiterung in 2D gegeben. Es werden Kanten als Oberfläche gesucht und Punkte werden als Pivot-Element verwendet.

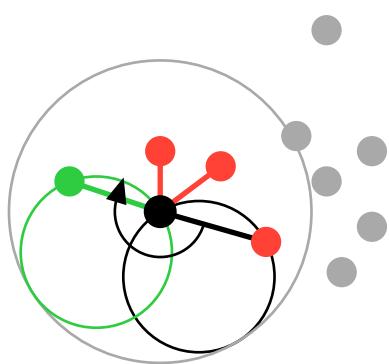


Abbildung 5: Abbildung 5: Erweiterung der gefundenen Oberfläche in 2D. Die vorherige Kante und der momentane Pivot-Punkt sind in Schwarz. Der  $\alpha$ -Kreis rollt entlang der markierten Richtung. In Grün ist der erste Punkt und zugehörigen Kreis, welche berührt werden. Die weiteren Punkte in Rot sind Kandidaten, liegen aber weiter in der Rotation, weshalb die grüne Kante zur Oberfläche hinzugefügt wird.

### 2.5.1. Kandidaten bestimmen

Um den ersten Punkt  $p$  zu finden, werden zuerst alle möglichen Punkte bestimmt, welche von der Kugel bei der kompletten Rotation berührt werden können. Dafür wird der Mittelpunkt  $mp = \frac{p_1+p_2}{2}$  der Kante und der Abstand  $d = |p_{1,2} - mp|$  berechnet. Der Abstand zwischen dem Zentrum der Kugel und den Endpunkten von der Kante ist immer  $\alpha$ , dadurch ist der Abstand vom Zentrum zum Mittelpunkt  $x = \sqrt{\alpha^2 - d^2}$ . In Abbildung 6 ist die Berechnung veranschaulicht.

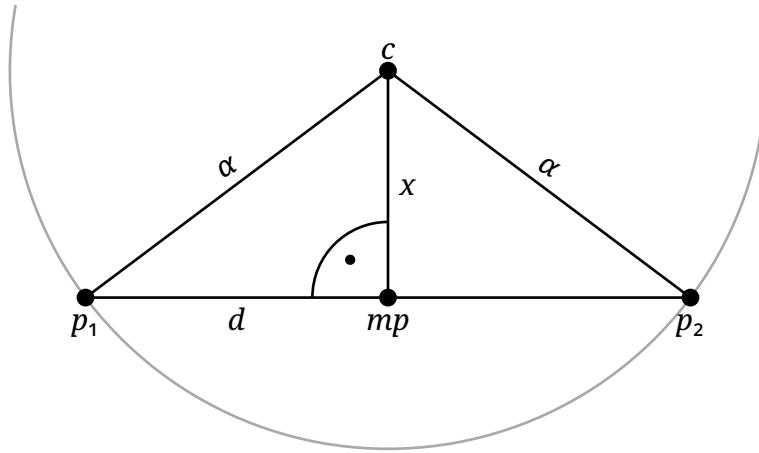


Abbildung 6: Berechnung des Abstands der  $\alpha$ -Kugel vom Mittelpunkt der Kante

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

### 2.5.2. Besten Kandidaten bestimmen

Für jeden Kandidaten  $p$  wird berechnet, wie weit die Kugel um die Kante gerollt werden muss, bis die Kugel den Kandidaten berührt. Dafür wird zuerst das Zentrum  $c_p$  der  $\alpha$ -Kugel bestimmt, welche  $p_1$ ,  $p_2$  und  $p$  berührt. Die Kugel wird dabei wie in Abbildung 7 bestimmt, dass die Kugel auf der korrekten Seite vom potenziellen Dreieck liegt.  $p$  kann so liegen, dass es keine zugehörige  $\alpha$ -Kugel gibt, in diesem Fall wird  $p$  nicht weiter betrachtet. Für jeden Kandidaten wird der Winkel  $\varphi$  berechnet, wie weit um die Kante die Kugel gerollt wurde.

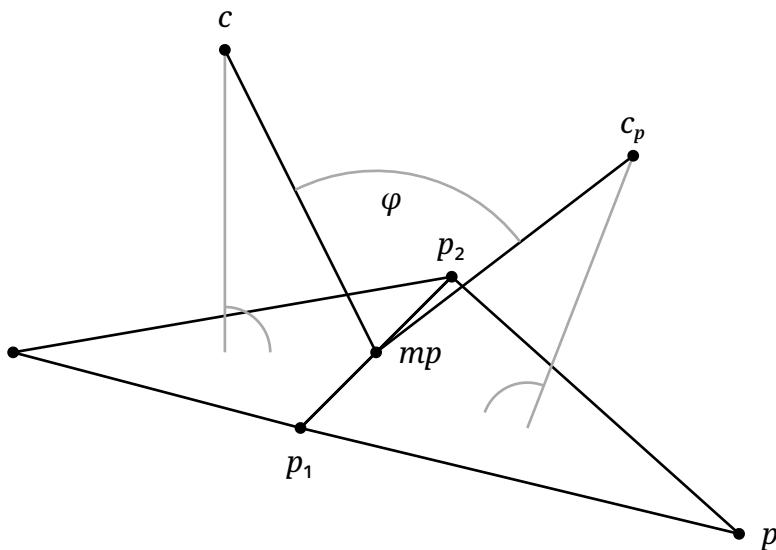


Abbildung 7: Berechnung von Zentrum der  $\alpha$ -Kugel und zugehöriger Winkel für einen Kandidatenpunkt

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

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

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

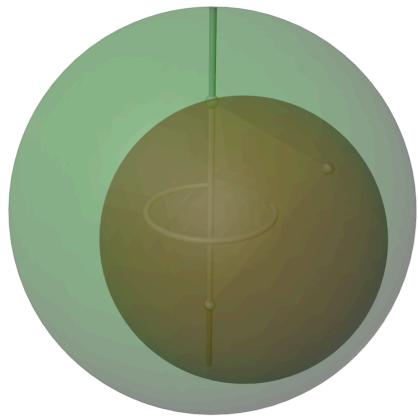
Es muss nicht kontrolliert werden, ob ein Punkt in der  $\alpha$ -Kugel von  $(p_1, p_2, p_3)$  liegt, weil diese immer leer ist. Würde ein weiterer Punkt in der Kugel liegen, so würde der zugehörige Winkel  $\varphi$  von diesem Punkt kleiner sein, weil der Punkt beim Rollen um die Kante früher von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

### 2.5.3. Triangulierung erweitern

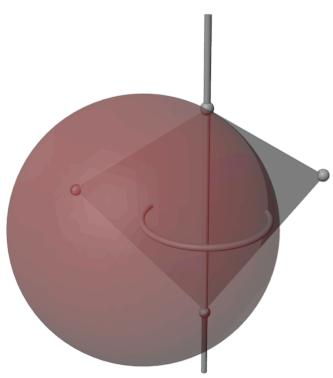
Das neu gefundene Dreieck mit den Eckpunkten  $(p_1, p_2, p_3)$  wird zur Triangulierung hinzugefügt. Die Kante  $(p_1, p_2)$  wird von den äußeren Kanten entfernt, dafür werden die Kanten zwischen  $(p_1, p_3)$  und  $(p_3, p_2)$  hinzugefügt. Wenn eine neue Kante in den äußeren Kanten bereits vorhanden ist, wird diese nicht hinzugefügt, sondern entfernt, weil das zugehörige zweite Dreieck bereits gefunden wurde. Ein Veranschaulichung ist in Abbildung 8 gegeben.



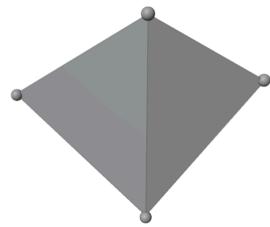
(a) Kante mit zugehörigem Dreieck, Kugel und Ring mit Radius  $x$



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



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



(d) Triangulierung, mit dem neuen Dreieck hinzugefügt

Abbildung 8: Erweiterung der Triangulierung in 3D.

## 2.6. Komplettes Segment triangulieren

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

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

## 2.7. Vorauswahl

Vor der Triangulierung wird mit dem Mindestabstand  $d$  die Anzahl der Punkte und deformierte Dreiecke verringert. Dafür wird ein Subset der Punkte bestimmt, dass die Punkte paarweise mindestens den Mindestabstand voneinander entfernt sind.

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

## 2.8. Auswahl von $\alpha$

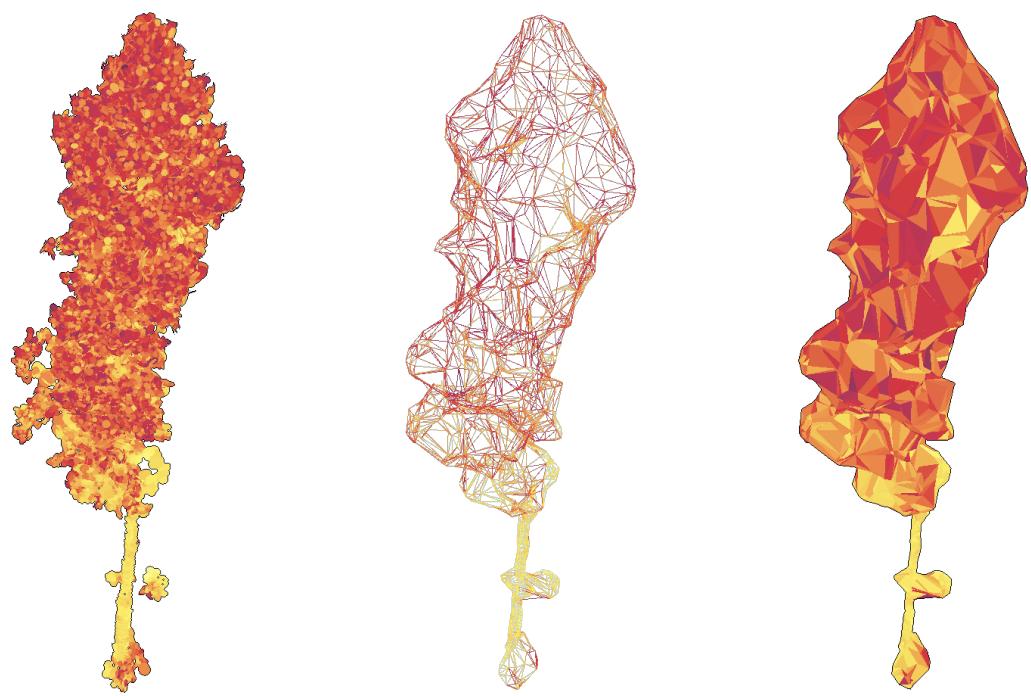
- Groß genug für keine Lücken
- Klein genug für gute Laufzeit

Todo: Auswahl  $\alpha$

## 3. Ergebnisse

Todo: Ergebnisse

Todo: Vergleich  $\alpha$



(a) Punkte

(b) Dreiecke umrandet

(c) Dreiecke ausgefüllt

Abbildung 9: Beispiel für eine Triangulierung von einem Baum.

# V. Visualisierung

## 1. Technik

Das Projekt ist unter <https://github.com/antonWetzel/treee> verfügbar. Für die technische Umsetzung wird die Programmiersprache Rust und die 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 von den Systemeigenschaften.

Als Eingabeformat werden Dateien im LASZip-Format verwendet. Dieses wird häufig für Punktwolken verwendet. Weiter Formate können einfach eingebunden werden, solange eine Rust verfügbar ist.

## 2. Punkt

### 2.1. Basis

Als Basis für einen Punkt wird ein Dreieck gerendert. Das Dreieck muss so gewählt werden, dass der Einheitskreis mit Zentrum  $(0, 0)$  vollständig enthalten ist.

Das kleinste Dreieck ist ein gleichseitiges Dreieck. In Abbildung 10 ist die Konstruktion für die Seitenlänge gegeben. Ein mögliches Dreieck hat die Eckpunkte  $(-\tan(60^\circ), -1)$ ,  $(\tan(60^\circ), -1)$  und  $(0, 2)$ .

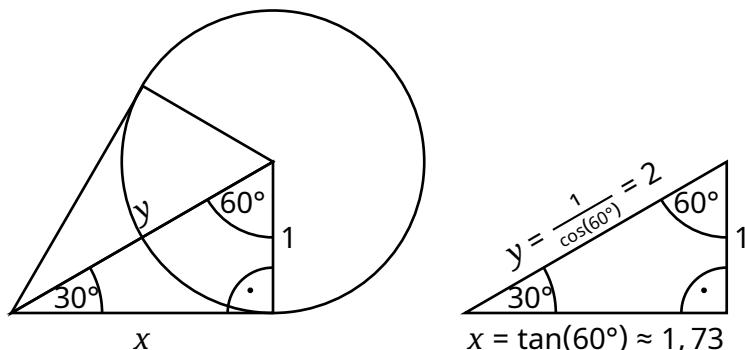


Abbildung 10: Seitenlänge für das kleinste gleichseitige Dreieck, welches den Einheitskreis enthält.

Für jeden Punkt wird mit der Position  $p$ , Normalen  $n$  und Größe  $s$  die Position der Eckpunkte vom Dreieck 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. Ein Beispiel ist in Abbildung 11 gegeben.

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.

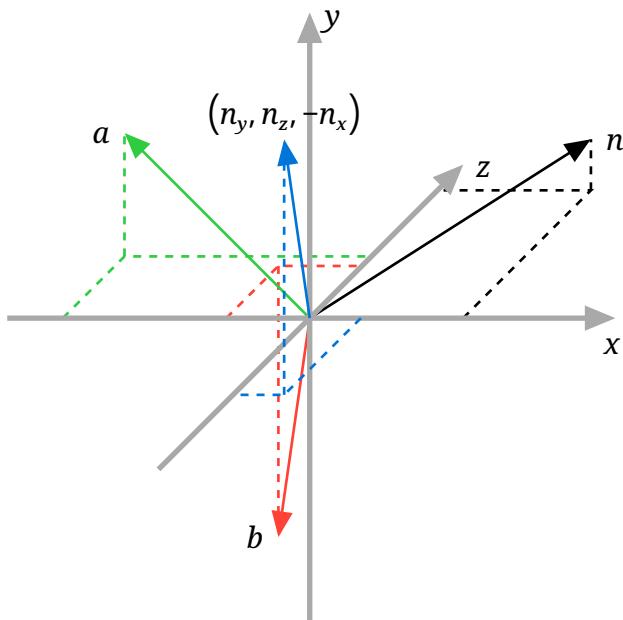


Abbildung 11: Beispiel für die Berechnung von  $a$  und  $b$  paarweise orthogonal zu  $n$ .

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 12 ist die Berechnung dargestellt.

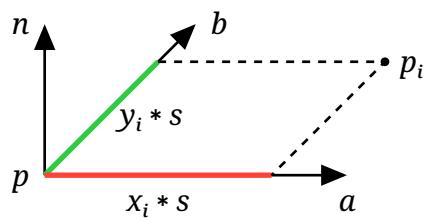


Abbildung 12: Berechnung von einem Eckpunkt.

## 2.2. Vergleich zu Quadrat als Basis

Als Basis kann ein beliebiges Polygon gewählt werden, solange der Einheitskreis vollständig enthalten ist. Je mehr Ecken das Polygon hat, desto kleiner ist der Bereich vom Polygon, der nicht zum Kreis gehört. Für jede Ecke vom Polygon muss aber die Position berechnet werden.

Ein Dreieck kann mit nur einem Dreieck dargestellt werden, für ein Quadrat werden zwei Dreiecke benötigt. Für das Dreieck werden dadurch drei Ecken und eine Fläche von  $\frac{w \cdot h}{2} = \frac{\tan(60^\circ) \cdot 2 \cdot 2}{2} = \tan(60^\circ) \cdot 2 \approx 3.46s$  benötigt. Für das Quadrat werden sechs Ecken und eine Fläche von  $w \cdot h = 2 \cdot 2 = 4$  benötigt. In Abbildung 13 ist ein grafischer Vergleich.

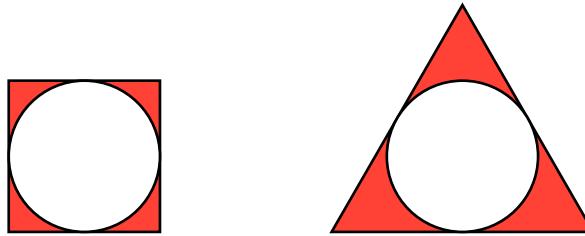


Abbildung 13: Quadrat und Dreieck, welche den gleichen Kreis enthalten.

Todo: Triangle Strip für Quadrat

Todo: Messwerte

### 2.3. Instancing

Weil für alle Punkte das gleiche Dreieck als Basis verwendet wird, muss dieses nur einmal zur Grafikkarte übertragen werden. Mit **Instancing** wird das gleiche Dreieck für alle Punkte verwendet, während nur die Daten spezifisch für einen Punkt sich ändern.

### 2.4. Kreis

Die Grafikpipeline bestimmt alle Pixel, welche im transformierten Dreieck liegen. Für jedes Pixel kann entschieden werden, ob dieser im Ergebnis gespeichert wird. Dafür wird bei den Eckpunkten die untransformierten Koordinaten abgespeichert, dass diese später verfügbar sind. Für jedes Pixel wird von der Pipeline die interpolierten Koordinaten berechnet. Nur wenn der Betrag der interpolierten Koordinaten kleiner als eins ist, wird der Pixel im Ergebnis abgespeichert.



(a) Dreiecke

(b) Quadrate

(c) Kreise

Abbildung 14: Unterschiedliche Formen für das Anzeigen der Punkte.

### **3. Ausgewählte Eigenschaft**

Die ausgewählte Eigenschaft wird durch Einfärbung der Punkte angezeigt. Dabei kann die ausgewählte Eigenschaft geändert werden, ohne die anderen Informationen über die Punkte neu zu laden. Die Eigenschaften sind separat als 32 bit uint gespeichert und werden mit einer Farbpalette in einen Farbverlauf umgewandelt. Auch die Farbpalette kann unabhängig ausgewählt werden.

## **4. Segmente**

### **4.1. Auswahl**

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

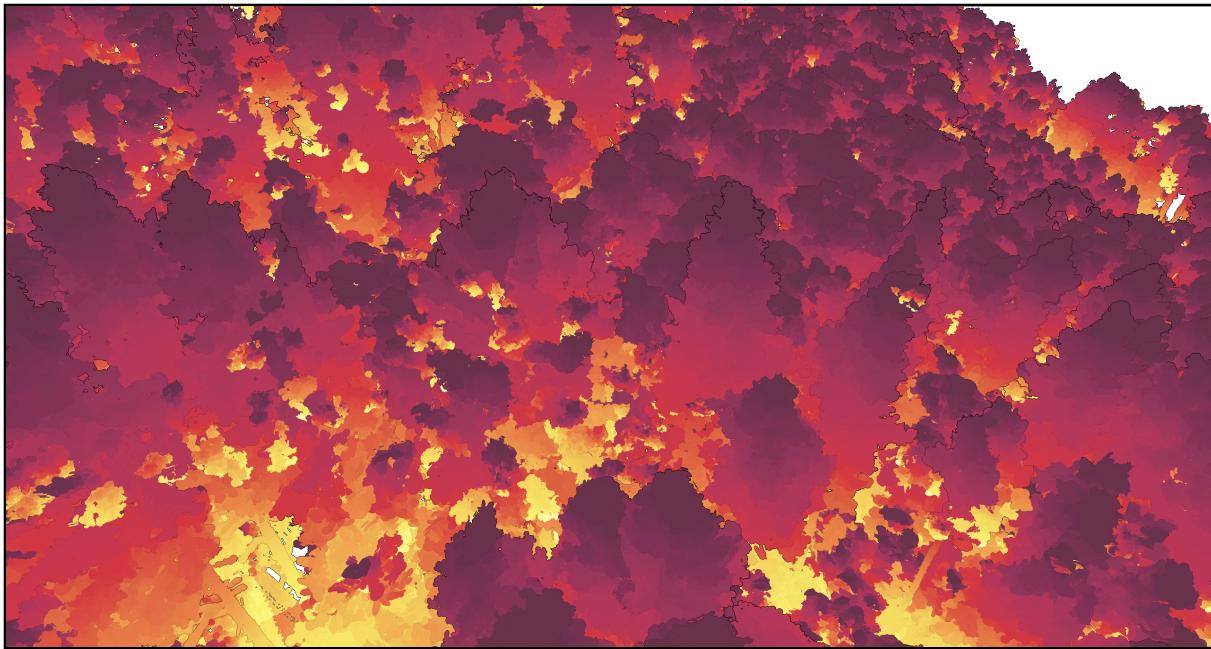
Im Octree wird vom Root-Knoten aus die Leaf-Knoten gefunden, welche den Strahl enthalten. Dabei werden die Knoten näher an der Position der Kamera bevorzugt. Für den Leaf-Knoten sind die Segmente bekannt, welche Punkte in diesem Knoten haben. Für jedes mögliche Segment wird für jeden Punkt überprüft,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

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

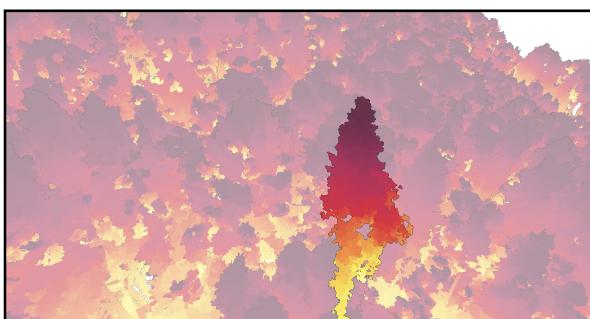
### **4.2. Anzeige**

Im Octree kann zu den Punkten in einem Leaf-Knoten mehrere Segmente gehören. Um die Segmente einzeln anzuzeigen, wird jedes Segment separat abgespeichert. Sobald ein einzelnes Segment ausgewählt wurde, wird dieses geladen und anstatt des Octrees angezeigt. Dabei werden alle Punkte des Segments ohne vereinfachte Detailstufen verwendet. Beispiele sind in Abbildung 15 gegeben.

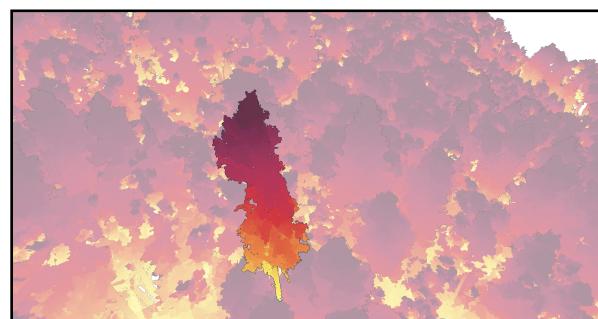
Die momentan geladenen Knoten vom Octree bleiben dabei geladen, um einen schnellen Wechsel zu ermöglichen.



(a) Alle Segmente



(b) Einzelnes Segment

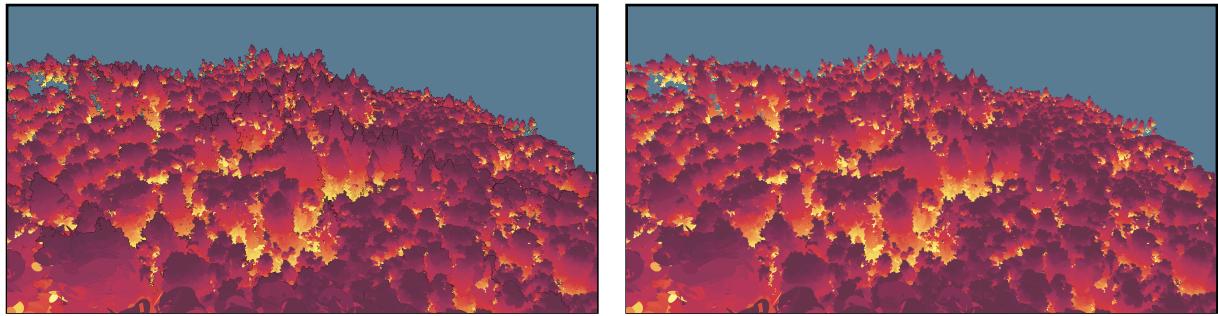


(c) Einzelnes Segment

Abbildung 15: Waldstück mit ausgewählten Segmenten.

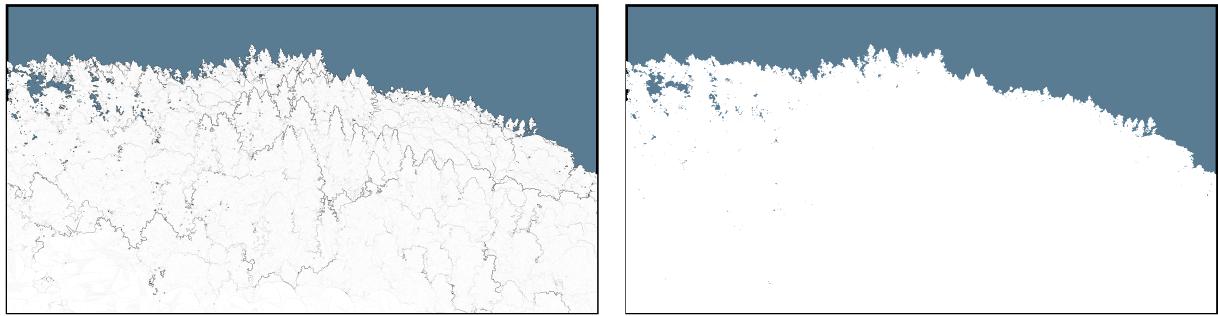
## 5. Eye-Dome-Lighting

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. Ein Veranschaulichung ist in Abbildung 16> gegeben.



(a) Ohne Eye-Dome-Lighting

(b) Mit Eye-Dome-Lighting



(c) Einfarbig ohne Eye-Dome-Lighting

(d) Einfarbig mit Eye-Dome-Lighting

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

Beim Rendern von 3D-Scenen wird für jedes Pixel die momentane Tiefe vom Polygon an dieser Stelle gespeichert. Das wird benötigt, dass bei überlappenden Polygone das nähere Polygon an der Kamera angezeigt wird. Nachdem die Szene gerendert ist, wird mit den Tiefeninformationen für jedes Pixel der Tiefenunterschied zu den umliegenden Pixeln bestimmt. Das Tiefenbild für die Veranschaulichung ist in Abbildung 17 gegeben.

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

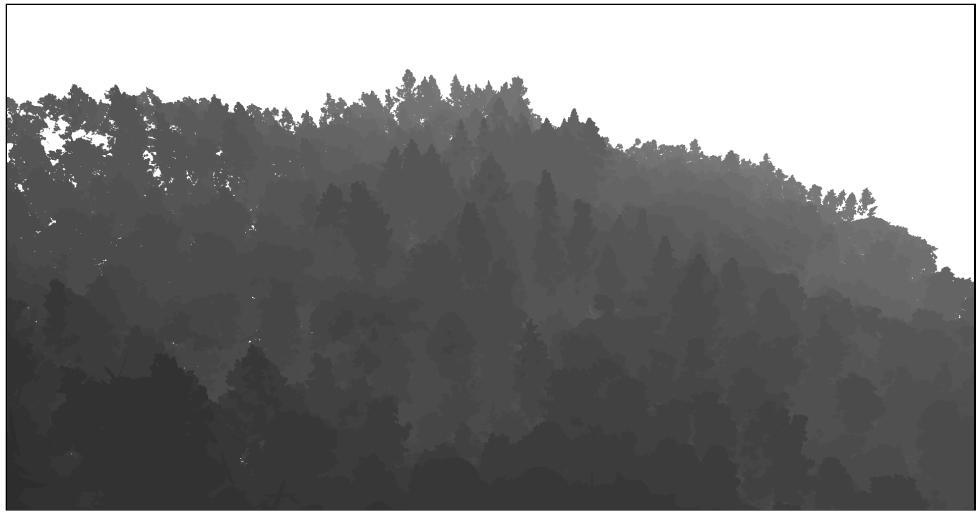


Abbildung 17: Tiefenbild nach dem render der Szene. Je heller eine Position ist, desto weiter ist das Polygon zugehörig zur Koordinate von der Kamera entfernt.

## 6. Detailstufen

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

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

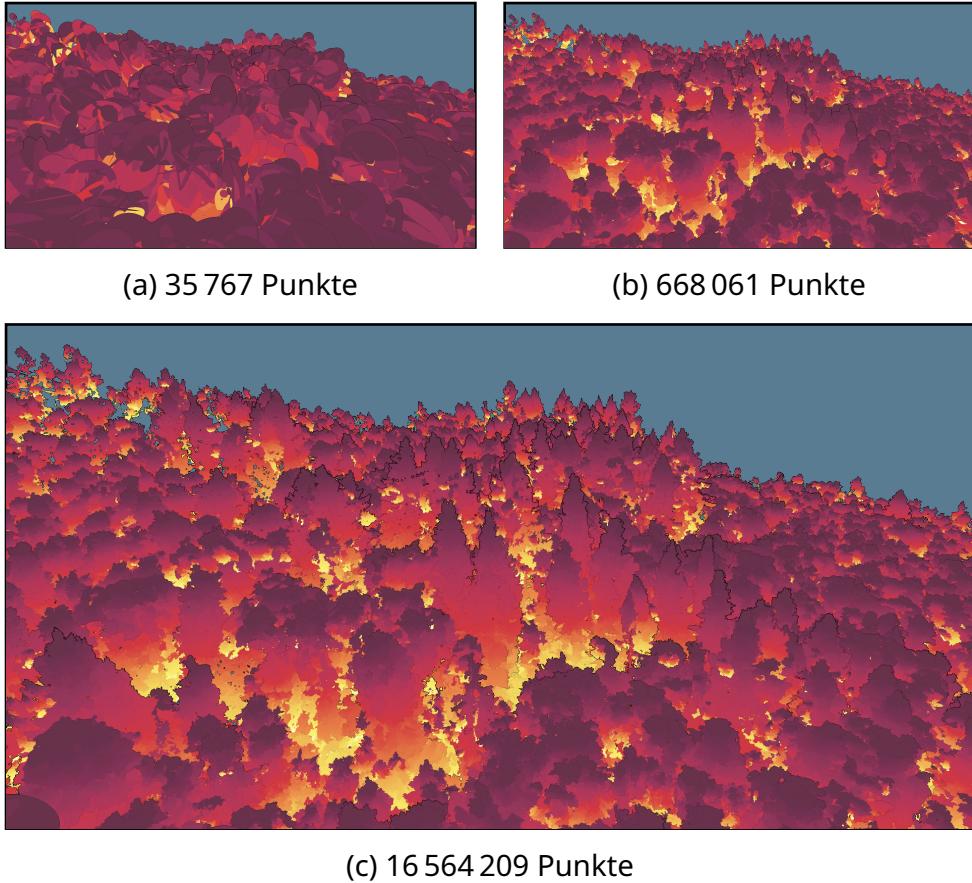


Abbildung 18: Waldstück mit unterschiedlichen Detailstufen, unabhängig von der Entfernung zur Kamera. Je höher die Detailstufe, desto mehr Punkte werden gerendert.

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

Für jeden Branch-Knoten wird eine Punktwolke berechnet, welche als Vereinfachung der Punkte der zugehörigen Kinderknoten verwendet werden kann. Dafür wird der Algorithmus aus Abschnitt V-6.1 verwendet.

Beim Anzeigen wird vom Root-Knoten aus zuerst geprüft, ob der momentane Knoten von der Kamera aus sichtbar ist. Für die Knoten wird mit den Algorithmen aus Abschnitt V-6.2 entschieden, ob die zugehörige vereinfachte Punktwolke gerendert werden soll oder rekursiv die Kinderknoten betrachtet werden sollen.

### **6.1. Berechnung der Detailstufen**

Todo: Kopiert vom Fachpraktikum

Die Detailstufen werden wie bei „Fast Out-of-Core Octree Generation for Massive Point Clouds“ [8] von den Blättern des Baumes bis zur Wurzel berechnet. Dabei wird als Einga-

be für einen Knoten die Detailstufen der direkten Kinder verwendet. Als Anfang werden alle originalen Punkte in einem Blatt als Eingabe benutzt.

Dadurch haben zwar Berechnungen der größeren Detailstufen für Knoten näher an der Wurzel nur Zugriff auf bereits vereinfachte Daten, dafür müssen aber auch viel weniger Punkte bei der Berechnung betrachtet werden. Solange die Detailstufen eine gute Vereinfachung der ursprünglichen Punkte sind, kann so der Berechnungsaufwand stark verringert werden.

Der Voxel, welcher zu dem Knoten gehört, wird in eine feste Anzahl von gleichgroßen Teilvoxel unterteilt. Für jeden Teilvoxel werden alle Punkte kombiniert, die im Teilvoxel liegen. Aus den Punkten im Teilvoxel wird ein repräsentativer Punkt bestimmt. Weil die Anzahl der Teilvoxel unabhängig von der Größe vom Voxel ist, sind die Teilvoxel für größere Detailstufen größer und mehr Punkte werden kombiniert.

## 6.2. Auswahl der Detailstufen?

Todo: Auswahl der Detailstufen

### 6.2.1. Abstand zur Kamera

- Schwellwert
- Abstand zur kleinsten Kugel, die den Voxel inkludiert
- Abstand mit Größe des Voxels dividieren
- wenn Abstand größer als Schwellwert
  - Knoten rendern
- sonst
  - Kinderknoten überprüfen

### 6.2.2. Auto

- wie Abstand zur Kamera
- messen, wie lang rendern dauert
- Dauer kleiner als Mindestdauer
  - Schwellwert erhöhen
- Dauer kleiner als die Maximaldauer
  - Schwellwert verringern

### 6.2.3. Gleichmäßig

- gleich für alle Knoten
- auswahl der Stufe

## 7. Filtern mit dem Kamerafrustrum

Todo: Frustrum-Culling

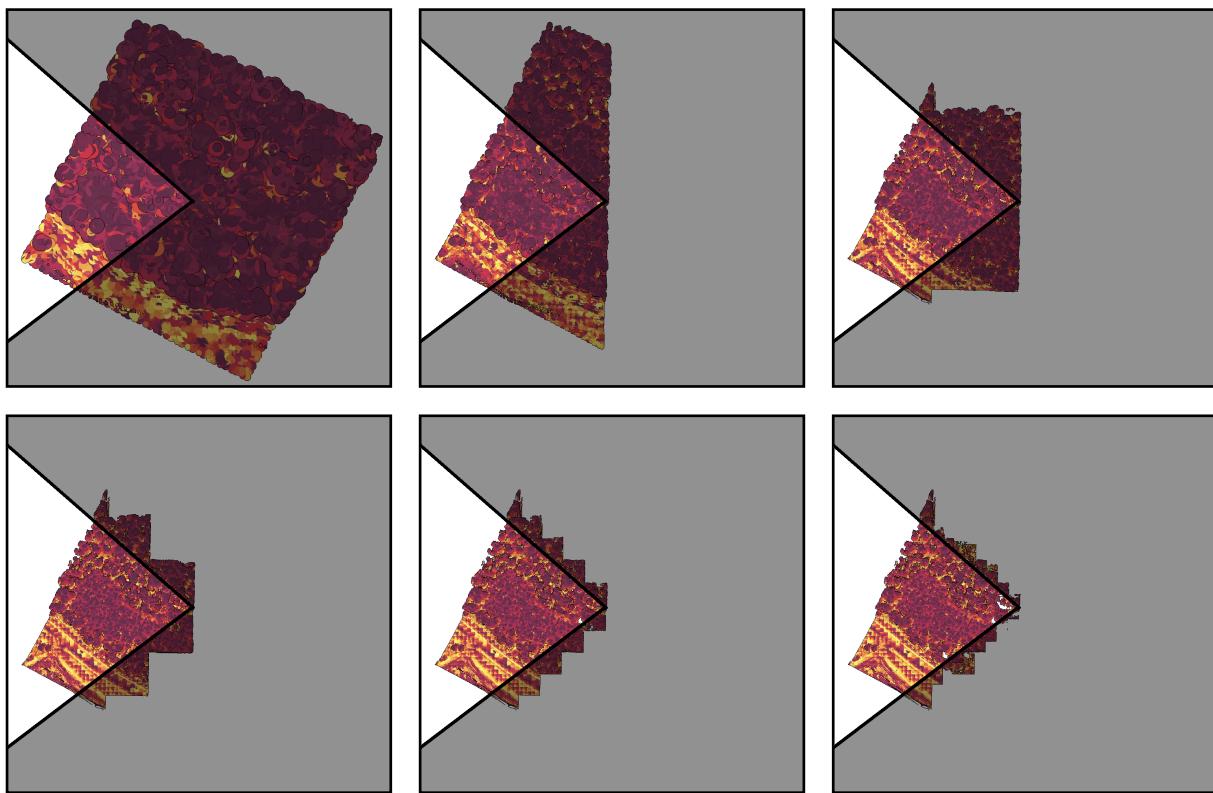


Abbildung 19: Unterschiedliche Detailstufen mit den unterschiedlichen sichtbaren Knoten.

## 8. Kamera/Projektion

### 8.1. Kontroller

- bewegt Kamera
- kann gewechselt werden, ohne die Kameraposition zu ändern

#### 8.1.1. Orbital

- rotieren um einen Punkt im Raum
- Kamera fokussiert zum Punkt
- Entfernung der Kamera zum Punkt variabel
- Punkt entlang der horizontalen Ebene bewegbar
- To-do: Oben-Unten Bewegung

#### 8.1.2. First person

- rotieren um die Kameraposition
- Bewegung zur momentanen Blickrichtung
- Bewegungsgeschwindigkeit variabel
- To-do: Oben-Unten Bewegung

# VI. Ergebnisse

Todo: Ergebnisse

## 1. Testdaten

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

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

## 2. Triangulierung

Todo: Triangulierung Ergebnisse

## 3. Berechnung

Todo: Berechnung Ergebnisse

## 4. Visualisierung

Todo: Visualisierung Ergebnisse

## VII. Fazit

Todo: Fazit

## VIII. Appendix

### 1. KD-Baum

Für eine Menge von Punkten kann ein KD-Baum bestimmt werden. Mit diesem kann effizient bestimmt werden, welche Punkte innerhalb einer Kugel mit beliebiger Position und Radius liegen. Ein Beispiel für einen KD-Baum ist in Abbildung 20 gegeben.

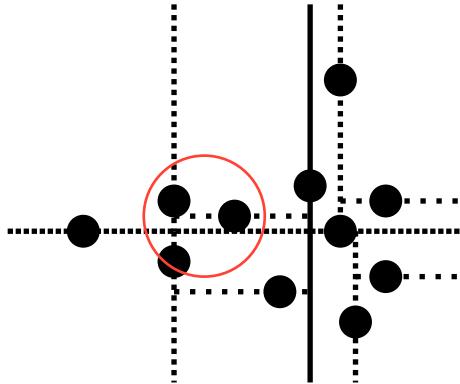


Abbildung 20: Abbildung 20: KD-Baum für Punkte in 2D. Für jede Unterteilung ist die Trenngerade gepunkteter gezeichnet. Weil der rote Kreis vollständig auf einer Seite der ersten Unterteilung ist, müssen die Punkte auf der anderen Seite nicht betrachtet werden.

#### 1.1. Konstruktion

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

Zuerst wird für die Punkte entlang der ersten Dimension der Median bestimmt. Dabei wird der *Quickselect*-Algorithmus [10] verwendet. Der Median hat als Index die halbe Anzahl der Punkte. Ist die Anzahl der Punkte ungerade, so kann der Index auf- oder abgerundet werden, solange bei der Suche die gleiche Strategie verwendet wird. Wie beim *Quicksort*-Algorithmus wird ein beliebiges Pivot-Element ausgewählt, mit diesem die Positionen entlang der Dimension unterteilt werden. Die Positionen werden einmal iteriert und kleinere Positionen vor dem Pivot und größere Positionen nach dem Pivot verschoben. Der Pivot ist am Index, wo es in der sortierten List wäre. Um den Median zu finden, wird nur der Teil von den Punkten betrachtet, welcher den zugehörigen Index beinhaltet. Die Unterteilung wird so lange wiederholt, bis der Median bekannt ist.

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

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

#### 1.2. Suche mit festem Radius

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

sucht werden. Liegen Teile der Kugel auf beiden Seiten, so müssen beide Teilbaum weiter durchsucht werden.

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

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

### 1.3. Suche mit fester Anzahl

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

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

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

### 1.4. Schnelle Suche

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

## 2. Baum (Datenstruktur)

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

### 2.1. Konstruktion

Zuerst wird die räumliche Ausdehnung der Daten bestimmt. Dieser Bereich wird dem Root-Knoten zugeordnet. Solange noch zu viele Datenwerte im Bereich von einem Knoten liegen, wird dieser weiter unterteilt. Dafür wird der zugehörige Bereich entlang aller Dimensionen halbiert und jeder Teilbereich einem Kinderknoten zugeordnet. Bei einem Quadtree in 2D entstehen dadurch vier Kinderknoten und bei einem Octree in 3D acht Kinderknoten. Der Daten gehören nicht mehr zum unterteilten Knoten, sondern zu den Kinderknoten. Der unterteilte Knoten speichert stattdessen die Kinderknoten.

In Abbildung 21 und Abbildung 22 sind Beispiele in 2D und 3D gegeben.

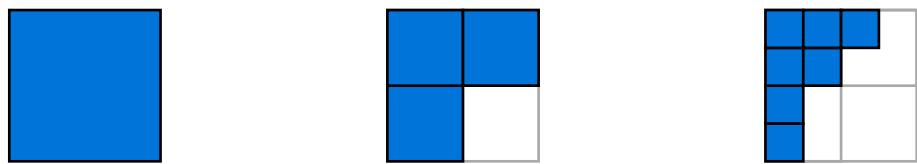


Abbildung 21: Unterschiedliche Stufen von einem Quadtree.

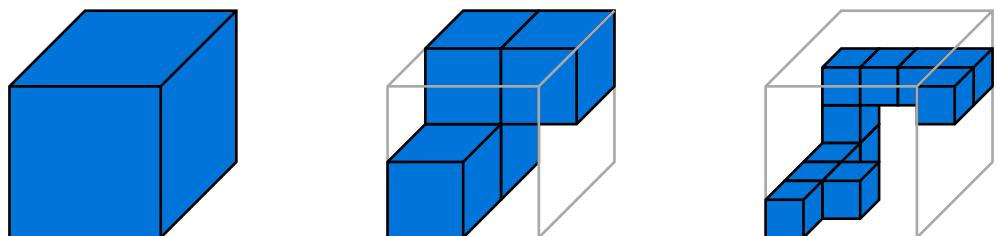


Abbildung 22: Unterschiedliche Stufen von einem Octree.

## 2.2. Suchanfrage

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

## 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 explicit abgespeichert oder relative zu den umliegenden Voxeln bestimmt werden.
- Tree** ist eine Datenstruktur, bestehend aus Knoten, welche wiederum Kinderknoten haben können. Die Knoten selber können weitere Informationen enthalten.
- Octree** ist eine Baumdatenstruktur, bei dem ein Knoten acht Kinderknoten haben kann. Mit einem Octree kann ein Voxel aufgeteilt werden. Jeder Knoten gehört zu einem Voxel, welcher gleichmäßig mit den Kinderknoten weiter unterteilt wird.
- Quadtree** ist eine Baumdatenstruktur, bei dem ein Knoten vier Kinderknoten haben kann. Statt eines Voxels bei einem Octree, wird ein zweidimensionales Quadrat unterteilen.
- Leaf-Knoten** ist ein Knoten, welcher keine weiteren Kinderknoten hat. Für Punktwolken gehört jeder Punkt zu genau einem Leaf-Knoten.
- Branch-Knoten** ist ein Knoten, welcher weitere Kinderknoten hat.
- Root-Knoten** ist der erste Knoten im Tree, alle anderen Knoten sind direkte oder indirekte Kinderknoten vom Root-Knoten.
- KD-Baum** ist eine Datenstruktur, um im  $k$ -dimensionalen Raum für eine Position die nächsten Punkte zu bestimmen.

## Bibliographie

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