

## TODOs

1: Akronyme .....	2
2: Referenzen zu späteren Abschnitten .....	3
3: Mehr Überblick .....	4
4: Aktuelle Bilder .....	6
5: Mehr Baumeigenschaften .....	7
6: Baumeigenschaften + ? → Segmente .....	7
7: Segmente + Eigenschaften + ? → Klassifizierung? .....	8
8: Meshing .....	9
9: Anpassen für Segmente .....	9
10: Umsetzen in Rust .....	9
11: Mehr Bilder .....	9
12: Bild .....	11
13: Bild .....	11
14: Triangle Strip für Quadrat .....	14
15: Messwerte .....	14
16: Bilder Crop .....	14
17: Oben/Unten Teilung in 2 Segmente für Debug .....	15
18: Vergleichsbild .....	15
19: Farb- und Tiefenbild .....	16
20: Stark und Schwache Effekt Bild .....	16
21: Vergleich alle Punkt und vereinfachte Versionen .....	16
22: Kopiert vom Fachpraktikum .....	17
23: Orthogonal? .....	18
24: Bedienung/Interface .....	18
25: Referenzen .....	18

# 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: 04.12.2023



**TECHNISCHE UNIVERSITÄT  
ILMENAU**

# Inhaltsverzeichnis

1. Glossar .....	1
1.1. Punktwolken .....	1
1.2. Datenstrukturen .....	1
1.3. Akronyme .....	2

## I. Überblick

1. Testdaten .....	3
2. Ablauf .....	3
2.1. Diskretisierung .....	3
2.2. Segmente bestimmen .....	3
2.3. Segmente unterteilen .....	3
2.4. Analyse und Speichern .....	3
2.5. Speichern vom Octree .....	3
3. Stand der Technik .....	4

## II. Berechnung

1. Separierung in Bäume .....	5
1.1. Diskretisieren .....	5
1.2. Segmente bestimmen .....	5
1.2.1. Bodenhöhe bestimmen .....	5
1.2.2. Bäume bestimmen .....	5
1.3. In Segmente unterteilen .....	5
2. Eigenschaften .....	6
2.1. Nachbarschaft .....	6
2.2. Krümmung .....	6
2.3. Punkthöhe .....	6
2.4. Ausdehnung .....	7
3. Segmentierung von einem Baum .....	7
4. Eigenschaften für Visualisierung .....	7
4.1. Normale .....	7
4.2. Punktgröße .....	7
5. Baumart .....	8

## III. Meshing

1. Triangulierung .....	9
1.1. Ziel .....	9
1.2. Vorverarbeitung .....	9
1.3. Datenstrukturen .....	9
1.3.1. Punktstatus .....	9
1.3.2. Aktive Kante .....	9
1.3.3. Äußere Kanten .....	10
1.3.4. Dreieckszähler .....	10
1.4. Gesuchte Triangulierung .....	10

1.5. Ablauf .....	10
1.5.1. Anfang .....	10
1.5.2. Triangulierten Bereich erweitern .....	10
1.5.3. Komplettes Blatt triangulieren .....	11
1.6. Limitierung .....	11
1.7. Ergebnisse .....	11

#### **IV. Visualisierung**

1. Technik .....	12
2. Punkt .....	12
2.1. Basis .....	12
2.2. Vergleich zu Quadrat als Basis .....	13
2.3. Instancing .....	14
2.4. Kreis .....	14
3. Eigenschaft .....	14
4. Subpunktwolken (Bäume) .....	14
4.1. Auswahl .....	14
4.2. Anzeige .....	15
5. Eye-Dome-Lighting .....	15
6. Detailstufen .....	16
6.1. Berechnung der Detailstufen .....	16
6.2. Auswahl der Detailstufen? .....	17
6.2.1. Abstand zur Kamera .....	17
6.2.2. Auto .....	17
6.2.3. Gleichmäßig .....	17
7. Kamera/Projektion .....	17
7.1. Kontroller .....	17
7.1.1. Orbital .....	18
7.1.2. First person .....	18
7.2. Projektion .....	18
7.2.1. Perspektive .....	18
7.2.2. Orthogonal? .....	18
7.2.3. Side-by-Side 3D? .....	18
8. Bedienung/Interface .....	18
Bibliographie .....	19

# 1. Glossar

## 1.1. Punktwolken

**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.

## 1.2. Datenstrukturen

**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 ein Tree, 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. Eine Veranschaulichung ist in Abbildung 1 gegeben.

**Quadtree** ist ein Tree, bei dem ein Knoten vier Kinderknoten haben kann. Statt eines Voxels bei einem Octree, kann ein Quadtree ein zweidimensionales Quadrat unterteilen. Eine Veranschaulichung ist in Abbildung 2 gegeben.

**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.

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

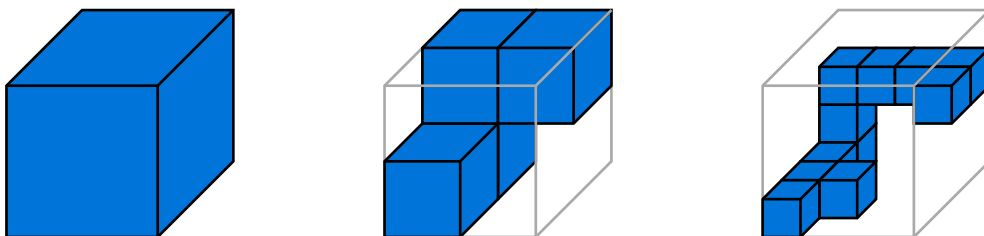


Abbildung 1: Unterschiedliche Stufen von einem Octree.

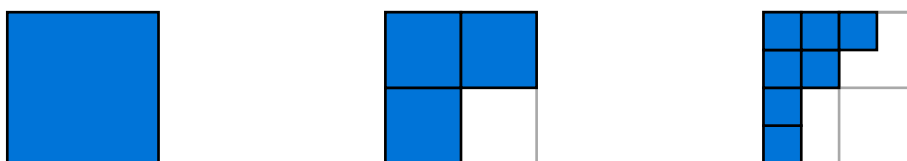


Abbildung 2: Unterschiedliche Stufen von einem Quadtree.

### 1.3. Akronyme

Todo: Akronym

# I. Überblick

## 1. Testdaten

Der benutzte Datensatz [1] 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. 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. Diskretisierung

Die Eingabedaten können beliebig viele Punkte enthalten. Für die weiteren Phasen wird deshalb eine vereinfachte Version berechnet, bei der alle Punkte in diskrete Voxel unterteilt werden. Die vereinfachte Version kann vollständig im Hauptspeicher geladen sein.

### 2.2. Segmente bestimmen

Für jeden gefüllten Voxel wird bestimmt, zu welchem Segment er gehört. Dafür werden die gefüllten Voxel nach der Höhe geordnet. Von Oben nach Unten werden die Voxel zu einem Segment zugeordnet. Dafür werden Voxel dem gleichen Segment zugeordnet, zu dem nach Voxel gehören.

### 2.3. Segmente unterteilen

Für jeden Punkt wird der zugehörige Voxel bestimmt und das Segment vom Voxel dem Punkt zugeordnet. Die Punkte werden in Segmente unterteilt gespeichert.

### 2.4. Analyse und Speichern

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.

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

### 2.5. Speichern 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.

### 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 [2]. Bei einem Scan von Boden aus, kann nur eine kleinere Fläche abgetastet werden, dafür mit erhöhter Genauigkeit, um einzelne Bäume genau zu analysieren [3]. Aus der Luft können größere Flächen erfasst werden, wodurch Waldstücke aufgenommen werden können, aber die Datenmenge pro Baum ist geringer [4].

Nach der Datenerfassung können relevante Informationen aus den Punkten bestimmt werden, dazu gehört eine Segmentierung in einzelne Bäume [5] und die Berechnung von Baumhöhe oder Kronenhöhe [4].

Ein häufiges Format für Lidar-Daten ist das LAS Dateiformat [6]. 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 [7] gespeichert. Die Kompression ist Verlustfrei und ermöglicht eine Kompressionsrate zwischen 5 und 15 je nach Eingabedaten.

*LASTools* [8] 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. Berechnung

### 1. Separierung in Bäume

#### 1.1. Diskretisieren

Die Eingabedaten können beliebig viele Punkte beinhalten, wodurch es wegen Hardwarelimitierungen nicht möglich ist alle Punkte gleichzeitig zu laden. Um eine schnellere Verarbeitung zu ermöglichen, wird zuerst eine vereinfachte Version der ursprünglichen Punktwolke bestimmt.

Dafür wird die gesamte Punktwolke in gröbere Voxel unterteilt und Punkte im gleichem Voxel werden zusammengefasst. Die Größe von den Voxeln ist dabei als Makroparameter einstellbar (WIP). Als Standardwert wird 5cm verwendet.

Für jeden Voxel wird gespeichert, wie viele Punkte zum Voxel gehören.

#### 1.2. Segmente bestimmen

##### 1.2.1. Bodenhöhe bestimmen

(Momentan programmiert nur niedrigster Punkt)

Für die Berechnung der Bodenhöhe wird Quadrate entlang der Horizontalen betrachtet. (WIP) Zuerst werden alle Voxel zusammengefasst, welche unabhängig von der Höhe zum gleichen Quadrat gehört.

Weil die zugehörige, zugehörig, zugehöre Punktzahl von jedem Voxel gespeichert ist, kann die Gesamtanzahl der Punkte von einem Quadrat bestimmt werden. Das gewünschte Quantil wird als Makroparameter festgelegt und es wird die Bodenhöhe zu gewählt, dass der Anteil der Punkte unter der Bodenhöhe dem Quantil entspricht. Als Standardwert wird 2% verwendet.

##### 1.2.2. Bäume bestimmen

(Noch im Wandel)

Für die Bestimmung der Bäume werden alle Voxel, die über dem Boden liegen in horizontale Scheiben unterteilt. Alle Punkte unter der Bodenhöhe werden nicht weiter segmentiert.

Von der höchsten Scheibe aus werden die Voxel zu Segmenten zugeordnet. Dafür wird jeder Voxel in der Scheibe betrachtet. Für den momentanen Voxel wird der nächste Voxel in einer höheren Scheibe gesucht, wobei es eine Maximaldistanz gibt. Wird ein naher Voxel gefunden, so wird dem momentanen Voxel das gleiche Segment wie dem gefundenen Voxel zugeordnet. Wenn kein naher Voxel gefunden wird, so ist der momentane Voxel der Anfang von einem neuen Segment.

#### 1.3. In Segmente unterteilen

Nachdem alle Voxel zu einem Segment gehören werden alle originalen Punkte nochmal geladen. Dabei wird für jeden Punkt der zugehörige Voxel bestimmt und dem Punkt das Segment vom Voxel zugeordnet.

Die Punkte werden nach Segment getrennt abgespeichert, um weiter zu verarbeitet zu werden.

## 2. Eigenschaften

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

### 2.1. Nachbarschaft

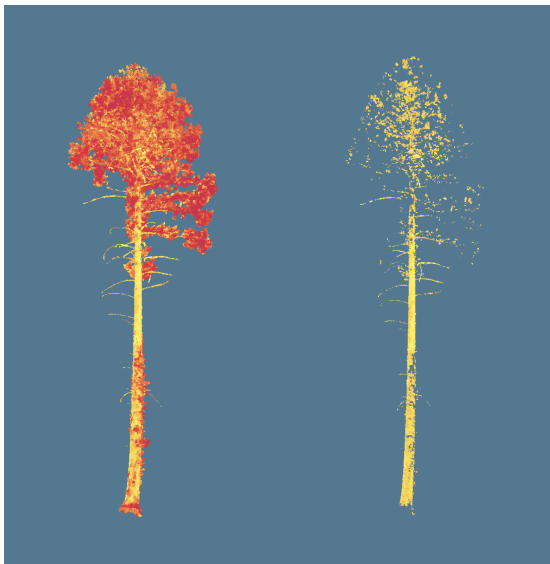
Um relevante Eigenschaften für einen Punkt zu bestimmen, werden die umliegenden Punkte benötigt. Dafür wird für alle Punkte ein **KD-Baum** erstellt. Mit diesem können effizient für ein Punkt die  $k$ -nächsten Punkte bestimmt werden.

### 2.2. 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 Punkt in der Nachbarschaft 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  $[0; 1]$ .



Todo: Aktuelle Bilder

### 2.3. Punkthöhe

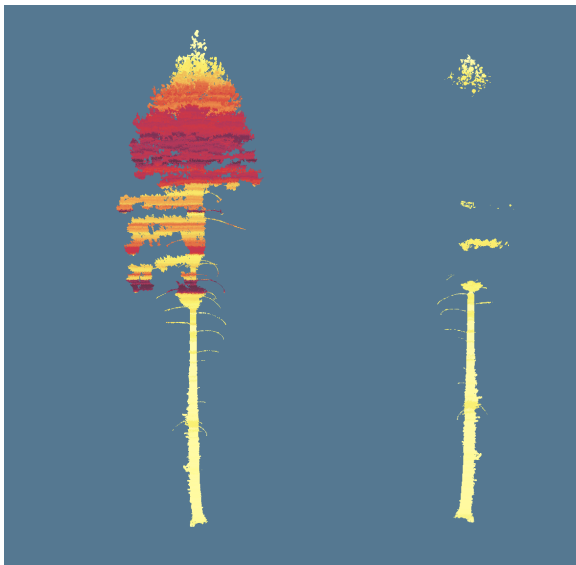
Für jeden Punkt wird die relative Höhe im Segment bestimmt. Dafür wird die Mindesthöhe  $y_{\min}$  und die Maximalhöhe  $y_{\max}$  im Segment benötigt. Die Höhe  $h$  für den Punkt mit

der 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  $[0; 1]$ .

## 2.4. Ausdehnung

Der Baum wird entlang der Horizontalen in gleichhohe Scheiben unterteilt. Die Höhe der Scheiben ist dabei einstellbar. Für jede werden alle Punkte in der Scheibe betrachtet. Zuerst wird der geometrische Schwerpunkt der Positionen berechnet, womit die durchschnittliche Standardabweichung entlang der Horizontalen bestimmt wird.

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



Todo: Mehr Baumeigenschaften

## 3. Segmentierung von einem Baum

Todo: Baumeigenschaften + ? → Segmente

## 4. Eigenschaften für Visualisierung

### 4.1. Normale

Mit den Eigenvektoren aus Abschnitt 2.2 wird die Normale für die Umgebung bestimmt. Der Eigenvektor, welcher zum kleinsten Eigenwert gehört, ist orthogonal zur Ebene mit der größten Ausdehnung.

### 4.2. Punktgröße

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

## 5. Baumart

Todo: Segmente + Eigenschaften + ? → Klassifizierung?

- out of scope?
- neural?

# III. Meshing

Todo: Meshing

Todo: Anpassen für Segmente

Todo: Umsetzen in Rust

Todo: Mehr Bilder

## 1. Triangulierung

### 1.1. Ziel

Eine Triangulierung ermöglicht eine Rekonstruktion der ursprünglichen gescannten Oberfläche, um diese weiterzuverarbeiten oder anzuzeigen. Die meisten Programme und Hardware sind auf das Anzeigen von Dreiecken spezialisiert, und können diese effizienter als Punkte darstellen.

### 1.2. Vorverarbeitung

Damit das Blatt mit den Punkten trianguliert werden kann, müssen die Punkte zu einer Oberfläche gehören. Das ist bei Scans von dreidimensionalen Objekten in der Regel der Fall.

Bei Regionen, wo mehrere Scans überlappen, können Punkte entlang der Oberfläche sehr nah aneinander liegen. Durch Ungenauigkeiten beim Messen oder der Transformation in das globale Koordinatensystem sind aber Unterschiede orthogonal zur Oberfläche möglich. Für die Triangulierung dieser Punkte würde ein Dreieck orthogonal zur eigentlichen Oberfläche benötigt werden.

Bei Bereichen, bei denen die Oberfläche zwischen Scans unterschiedlich ist, werden beide Oberfläche trianguliert und beim Übergang ist keine eindeutige Triangulierung möglich.

### 1.3. Datenstrukturen

#### 1.3.1. Punktstatus

Für jeden Punkt wird gespeichert, ob dieser bereits zu einem Dreieck gehört. Dabei ist der Standardwert inaktiv für alle Punkte und während der Triangulierung wird der Wert auf aktiv gesetzt.

#### 1.3.2. Aktive Kante

Eine aktive Kante bildet den Übergang zwischen den bereits Triangulierten Bereich und den Rest des Blattes. Eine aktive Kante beinhaltet die beiden Indices der Eckpunkte, zwischen denen die Kante ist und den Index eines inneren Punktes, um die Richtung zum noch nicht Triangulierten Bereich zu bestimmen.

### **1.3.3. Äußere Kanten**

Die äußeren Kanten sind eine Warteschlange, welche die aktiven Kanten enthält. Während der Triangulierung werden die Kanten abgearbeitet, bis die Warteschlange leer ist.

### **1.3.4. Dreieckszähler**

Für die berechneten Kanten werden in einer Hashtabelle die zugehörige Anzahl von anliegenden Dreiecken gespeichert. Wenn die Kante noch keinen Eintrag in der Tabelle hat, wird für die Anzahl der Standardwert null verwendet.

## **1.4. Gesuchte Triangulierung**

Als Triangulierung wird eine Delaunay-Triangulierung gesucht, diese vermeidet spitzwinklige Dreiecke. Für eine Delaunay-Triangulierung werden Dreiecke gesucht, in deren Umkreis kein anderer Punkt liegt. Wegen der 3-Dimensionalität wird die Kugel verwendet, welche den Umkreis vom Dreieck als Großkreis hat.

Weil die Bedingung unabhängig davon ist, welche der Kanten von dem gesuchten Dreieck die aktive Kante ist, wird die gleiche Triangulierung unabhängig von der Auswahl der aktiven Kante gefunden.

## **1.5. Ablauf**

### **1.5.1. Anfang**

Als Anfang wird ein beliebiger Punkt im Blatt als erster Punkt ausgewählt. Von diesem Punkt wird ein Dreieck gesucht, welches den Anfang des triangulierten Bereiches bildet.

Als zweiter Punkt wird der räumlich nächste Punkt zum ersten Punkt ausgewählt. Dieser gehört immer zur Delaunay-Triangulierung, weil alle anderen Punkte weiter entfernt sind und somit dieser Punkt im Umkreis von den zwei weiter entfernten Punkten liegen müsste.

Als dritter Punkt wird der Punkt ausgewählt, dass der Winkel mit dem dritten Punkt als Winkelscheitel mit den Winkelschenkeln zu den beiden ersten Punkten maximiert wird. Dieser Punkt hat den kleinsten Umkreis und somit gehört das Dreieck mit den drei Punkten zur Triangulierung. Dafür werden alle Punkte iteriert, die in der Nachbarschaft vom ersten und zweiten Punkt sind.

Für alle drei Punkte wird der Punktstatus aktiv gesetzt und die drei Kanten vom Dreieck bilden die äußeren Kanten. Für die Kanten wird der zugehörige Dreieckszähler um eins erhöht.

### **1.5.2. Triangulierten Bereich erweitern**

Solange die äußeren Kanten nicht leer sind, kann der triangulierte Bereich erweitert werden. Dafür wird zuerst eine beliebige aktive Kante ausgewählt und aus den äußeren Kanten entfernt.

Wenn zu dieser Kante bereits zwei oder mehr Dreiecken gehört, wird diese entfernt, ohne ein weiteres Dreieck zu suchen. Sonst wird wieder der beste dritte Punkt gesucht, der den Winkel zu den beiden bekannten Punkten maximiert. In Abbildung 3 ist der Ablauf grafisch dargestellt.

Als zusätzliche Bedingung kommt hinzu, dass der dritte Punkt auf der anderen Seite zum inneren Punkt der aktiven Kante liegen muss. Wenn kein dritter Punkt gefunden wird, wird kein weiteres Dreieck zur Triangulierung hinzugefügt.

Für alle drei Punkte wird der Punktstatus aktiv gesetzt und die zwei neuen Kanten werden zu den äußeren Kanten hinzugefügt. Für alle drei Kanten vom Dreieck wird der zugehörige Zähler um eins erhöht.

Todo: Bild

Abbildung 3: Erweiterung um zwei Dreiecke. Für die ausgewählte aktive Kante in Rot wird das Dreieck in Grau gefunden und zum bereits Triangulierten Bereich in Grün hinzugefügt. Die neuen aktiven Kanten sind gepunktet eingezeichnet.

### 1.5.3. Komplettes Blatt triangulieren

Ein Blatt kann mehrere nicht-zusammenhängende Bereiche enthalten. Deshalb werden alle Punkte betrachtet und von jedem eine Triangulierung gestartet, wenn der Punktstatus noch nicht aktiv ist.

## 1.6. Limitierung

Weil die Triangulierung für jedes Blatt einzeln durchgeführt wird, kann zwischen den Punkten unterschiedlicher Blätter nicht trianguliert werden. Dadurch entsteht bei der Zusammenführung aller Triangulierung zwischen den Blättern sichtbare Lücken.

Durch die Limitierung auf die Nachbarschaft, bei der Suche des dritten Punktes für eine Kante, entstehen bei einer starken Veränderung der Punktdichte Lücken. Die Nachbarschaft von einem Punkt nah der Veränderung der Punktdichte beinhaltet nur die Punkte aus der Region mit der größeren Dichte. Dadurch findet die Suche nicht den optimalen dritten Punkt vom gewünschten Dreieck aus der Region mit der niedrigeren Dichte.

## 1.7. Ergebnisse

Todo: Bild

## IV. 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 4 ist die Konstruktion für die Längen gegeben. Ein mögliches Dreieck hat die Eckpunkte  $(-\tan(60^\circ), -1)$ ,  $(\tan(60^\circ), -1)$  und  $(0, 2)$ .

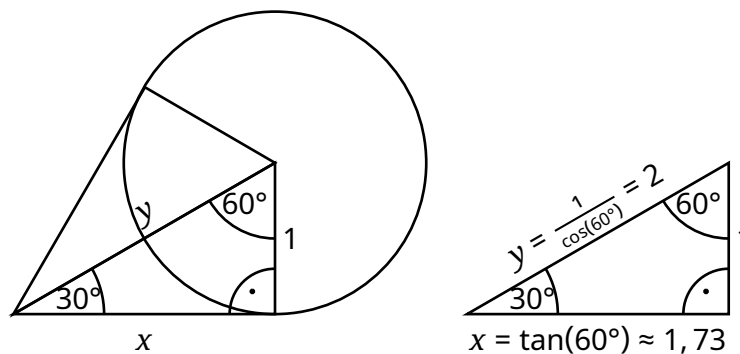


Abbildung 4: Längen für das kleinste gleichseitige Dreieck, welches den Einheitskreis enthält.

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 5 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.



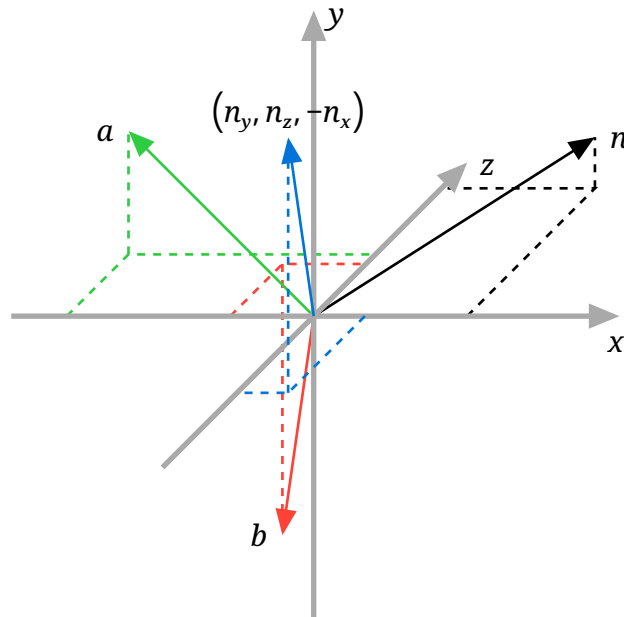


Abbildung 5: Beispiel für die Berechnung von  $a$  und  $b$  paarweise orthogonal zu  $n$ .

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 6 ist die Berechnung dargestellt.

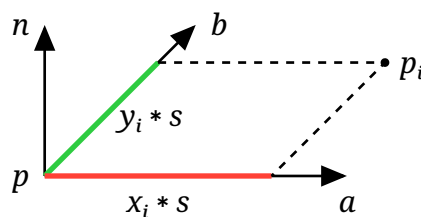


Abbildung 6: Berechnung von einem Eckpunkt.

## 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 einem Dreieck dargestellt werden, für eine Quadrat werden zwei 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 7 ist ein graphischer Vergleich.

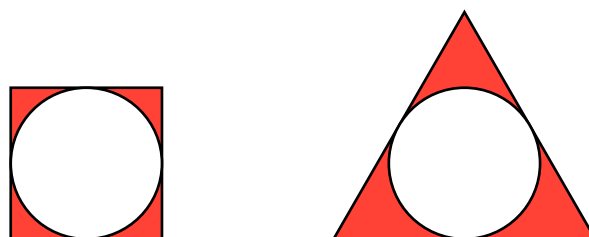


Abbildung 7: Quadrat und Dreieck, welche den gleichen Kreis enthalten.

Todo: Triangle Strip für Quadrat

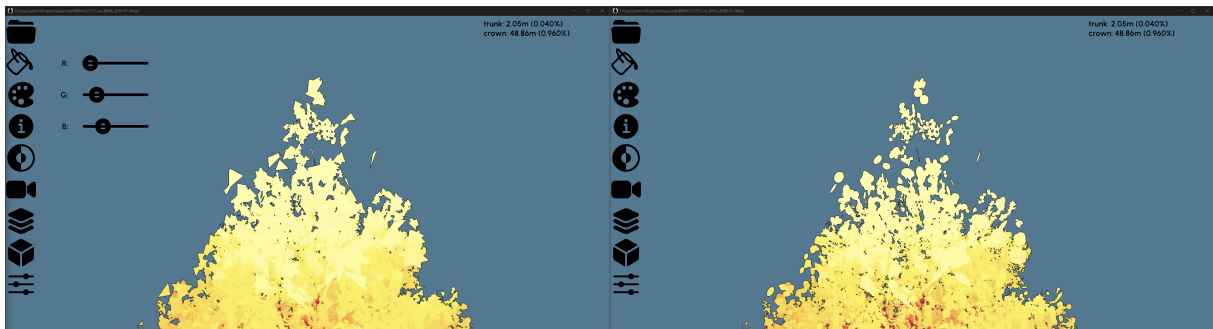
Todo: Messwerte

## 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 jeden 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 jeden Pixel wird von der Pipeline die interpolierten Koordinaten berechnet. Nur wenn der Betrag der interpolierten Koordinaten kleiner 1 ist, wird der Pixel im Ergebnis abgespeichert.



Todo: Bilder Crop

## 3. 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 ein Farbverlauf umgewandelt. Auch die Farbpalette kann unabhängig ausgewählt werden.

## 4. Subpunktvolken (Bäume)

### 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.

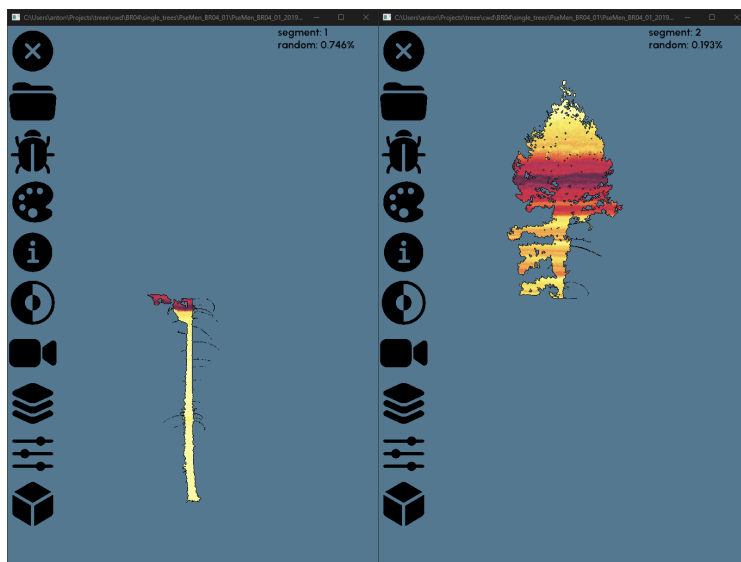
Im Octree wird vom Root-Knoten aus die Blatt-Knoten gefunden, welche den Strahl enthalten. Dabei werden die Knoten näher an der Position der Kamera bevorzugt. Für den Blattknoten sind die Segmente bekannt, welche Punkte in diesem Knoten haben. Für jedes mögliche Segment wird für jeden Punkt überprüft,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

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



Note: Oben/Unten Teilung in 2 Segmente für Debug

## 5. Eye-Dome-Lighting

Um die Punktwolke auf 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.

Todo: Vergleichsbild

Beim Rendern von 3D-Szenen wird für jeden 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 Scene gerendert ist, wird mit den Tiefeninformationen für jeden Pixel der Tiefenunterschied zu den umliegenden Pixeln bestimmt.

Todo: Farb- und Tiefenbild

Je größer der Unterschied ist, desto stärker wird der Pixel im Ergebnisbild eingefärbt. Dadurch werden Kanten hervorgehoben, je nachdem wie groß der Tiefenunterschied ist. Für den Effekt kann die Stärke und die Farbe angepasst werden.

Todo: Stark und Schwache Effekt Bild

## 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 Punkt 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.

Todo: Vergleich alle Punkt und vereinfachte Versionen

Für die gesamte Punktwolke 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 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 6.2 entschieden, ob die zugehörige vereinfachte Punktwolke gerendert oder der gleiche Algorithmus wiederholt wird für die Kinderknoten.

### 6.1. Berechnung der Detailstufen

Die Detailstufen werden wie bei „Fast Out-of-Core Octree Generation for Massive Point Clouds“ [9] von den Blättern des Baumes bis zur Wurzel berechnet. Dabei wird als Eingabe 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 gröberen 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 gleich große Zellen unterteilt. Für jede Zelle mit Punkten wird ein repräsentativer Punkt bestimmt. Dafür wird für die Zelle die Kombination aller Eingabepunkte, welche in der Zelle liegen berechnet. Die Anzahl der Zellen ist dabei unabhängig von der Größe des ursprünglichen Voxels, wodurch bei gröberen Detailstufen durch den größeren Voxel auch die Zellen größer werden und mehr Punkte zusammengefasst werden.

## 6.2. 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 Minstdauer
  - Schwellwert erhöhen
- Dauer kleiner als Maximaldauer
  - Schwellwert verringern

### 6.2.3. Gleichmäßig

- gleich für alle Knoten
- auswahl der Stufe

## 7. Kamera/Projektion

### 7.1. Kontroller

- bewegt Kamera

- kann gewechselt werden, ohne die Kameraposition zu ändern

#### **7.1.1. Orbital**

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

#### **7.1.2. First person**

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

### **7.2. Projektion**

#### **7.2.1. Perspektive**

- Projektion mit Field of View Kegel

#### **7.2.2. Orthogonal?**

Todo: Orthogonal?

#### **7.2.3. Side-by-Side 3D?**

## **8. Bedienung/Interface**

Todo: Bedienung/Interface

Todo: Referenzen

## Bibliographie

- [1] 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>
- [2] 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*, Nr. 12, S. 2297, 2021.
- [3] M. Disney, „Terrestrial LiDAR: a three-dimensional revolution in how we look at trees“, *New Phytologist*, Nr. 4, S. 1736–1741, 2019, doi: <https://doi.org/10.1111/nph.15517>.
- [4] 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*, S. 313–323, 2021, doi: [10.20965/ijat.2021.p0313](https://doi.org/10.20965/ijat.2021.p0313).
- [5] A. Burt, M. Disney, und K. Calders, „Extracting individual trees from lidar point clouds using treeseg“, *Methods in Ecology and Evolution*, Nr. 3, S. 438–445, 2019, doi: <https://doi.org/10.1111/2041-210X.13121>.
- [6] L. Graham, „The LAS 1.4 specification“, *Photogrammetric engineering and remote sensing*, Nr. 2, S. 93–102, 2012.
- [7] M. Isenburg, „LASzip: lossless compression of LiDAR data“, *Photogrammetric engineering and remote sensing*, Nr. 2, S. 209–217, 2013.
- [8] C. Hug, P. Krzystek, und W. Fuchs, „Advanced Lidar Data Processing with Lastools“, 2004. Verfügbar unter: <https://api.semanticscholar.org/CorpusID:14167994>
- [9] M. Schütz, S. Ohrhallinger, und M. Wimmer, „Fast Out-of-Core Octree Generation for Massive Point Clouds“, *Computer Graphics Forum*, Nr. 7, S. 155–167, 2020, doi: <https://doi.org/10.1111/cgf.14134>.