Bachelorarbeit am Institut für Informatik der Freien Universität Berlin

Darstellung gigantischer Punktwolken auf Android Geräten

Jakob Krause
Matrikelnummer: xxxxxxx
jakobkrause@zedat.fu-berlin.de

Betreuung und Erstgutachten:
Prof. Dr. Marco Block-Berlitz
Zweitgutachten:

26. April 2016

Inhaltsverzeichnis

1 Einführung

1.1 Motivation

Das Interesse an der Darstellung von großen Punktwolken ist durch das Aufkommen von günstigen 3D Scanner und der 3D Rekonstruktion in den letzten Jahrzehnt stark gewachsen. Aufgrund der technologischen Entwicklungen in diesem Bereich, ist es möglich von kleinen Objekten, wie einem Dinosaurier Schädel, bis hin zu ganzen Städte in hoher Detailstufe digital zu erfassen. In der Denkmalpflege werden Objekte mittlerweile aus Routine abgescannt und archiviert.

Für gewöhnlich sind die erzeugten Modelle enorm groß und daher nicht ohne weiteres darstellbar. Die Modelle bestehen meist aus einer ungeordneten Sammlung von dreidimensionalen Punkten mit Farbinformationen. Desktop Lösung verlassen sich zur Lösung des Problems auf Level-of-Detail(LOD) Konzepte kombiniert mit Out-of-core Verfahren und klugen Caching Strategien um der hohen Datenmenge Herr zu werden. Durch das Aufkommen von leistungsstarken Smartphones entstand eine neue Plattform zum Darstellen von Modellen.

Daraus entstand eine Nachfrage durch Forschungsgruppen für eine mobile Applikation zum Betrachten großer Modelle.

Die Applikation kann Beispielsweise bei 3D Rekonstruktion schnelles Feedback liefern, vereint mit den Vorzügen eines mobilen Gerätes. Dadurch können schnell unvollständige oder schwer zu erfassende Bereiche des Modells beim Scannen erkannt werden. Des weiteren kann die Applikation zur Präsentation von Modellen eingesetzt werden.

Die Herausforderung bestand darin trotz der Limitierungen eines Tablets durch seinen geringen Arbeitsspeicher und der Vergleichsweisen schwache GPU Punktwolken mit mehreren Millionen Punkten flüssig darzustellen. Dabei sollte es möglich sein jederzeit neue Punkte in die bestehende Darstellung hinzuzufügen. Zur Zeit existieren nur bedingt geeignet Softwarelösungen für das Problem.

1.1.1 Archaeocopter und Archaeonautic Projekt

Das Archaeocopter Projekt, welches 2012 von Dr. Benjamin Ducke und Prof. Dr. Marco Block-Berlitz ins Leben gerufen wurde, hat es sich zum Ziel gesetzt eine unbemanntes Flugobjekt (UAV) zu entwickeln, welcher durch halb-autonome Flüge Archäologen bei ihrer Arbeit durch Luftaufnahmen unterstützt. Aus diesen Aufnahmen lassen sich durch 3d Rekonstruktion Modelle generieren. Die Technik wird auch zum Denkmalschutz angewandt.

Die Idee war es aktuelle Verfahren aus der Computervision und künstlichen Intelligenz zusammen mit UAVs mit Kameras für die Datenerhebung einzusetzen. Das innovative

Abbildung 1.1: Archaeocopter Projekt

Verfahren lässt sich auch Unterwasser einsetzen.

Offiziell ging das Projekt im September 2012 mit Unterstützung von Prof. Dr. Raúl Rojas von Berlin's Freie Universität an den Start.

Diese Arbeit ist in Zusammenarbeit mit dem Projekt entstanden.

1.2 Ziele

Ziel dieser Arbeit ist es ein mobile Applikation für das Android Betriebssystem zu entwickeln, welche in der Lage ist Punktwolken mit mehreren Millionen Punkten performant darzustellen. Dabei soll es möglich sein Punkte in die bestehende Darstellung zur Laufzeit hinzuzufügen.

Ein weiteres Ziel der Arbeit ist es, die Architektur möglichst modular und plattformunabhängig zu gestalten um dadurch Austauschbarkeit und Wartbarkeit der einzelnen Komponenten zu gewährleisten. Es sollte möglichst einfach sein, eine Applikation für iOS oder ein anderes mobiles Betriebssystem nachzureichen. Des weiteren soll eine moderne Auswahl von Frameworks getroffen werden. Dabei soll die Applikation möglichst intuitiv benutzbar sein.

2 Theorie und verwandte Arbeiten

2.1 Theorie

2.1.1 Punktwolken

Definition Im Rahmen dieser Arbeitet ist ein Punkt p wie folgt definiert:

$$\begin{aligned} p &\in \{x, y, z, r, g, b\} \\ x, y, z &\in \mathbb{R} \\ r, g, b &\in [0, 1] \end{aligned}$$

X,y und z entspricht der Position im euklidischen Raum. R,g und b sind Farbwerte aus dem RGB-Farbraum.

Definition Eine Punktwolke ist eine Menge von Punkten p.

2.1.2 Einführung in die 3D Rekonstruktion

Die Idee aus einer Folge von Bilder ein 3D Model zu errechnen ist eines der Kernthemen der Computervision. Verwendungen dieser Technik sind vielfältig und finden sich in der Wissenschaft und Wirtschaft wieder. Anwendungen existieren zum Beispiel in der Robotik, [6] in welcher mit Hilfe eines Stereokamerasystems die Position des Roboters innerhalb seiner Umgebung feststellbar ist. Ein weiteres Feld ist die Archäologie und der Denkmalsschutz. Ein Beispiel dafür ist das Archaecopter Projekt.

Allgemein kann man zwischen aktiver und passiver Rekonstruktion [7] unterscheiden.

2.1.2.1 Aktive Rekonstruktion

Bei aktiver Rekonstruktion wird aktiv mit einem Sensors das Objekt abgetastet um die Struktur zu ermitteln. 3D Scanner sind ein Vertreter dieser Gattung. Sie sind der Kamera ähnlich. Genau wie diese besitzen sie ein Sichtfeld. Allerdings liefern sie statt Farbwert Abstandswert von seinem Sichtfeld. Die Abstandswerte können entweder über die „Time of Flight“ oder die Triangulierungmethode ermittelt werden.

Bei der „Time of Flight“ Methode wird ein Laserstrahl versendet. Aus der Dauer bis die Reflektion ihren Ausgangspunkt erreicht kann die Entfernung ermittelt werden. Diese Methode ist bei nahen und feinen Objekten ungenau weil die Zeit nur zu einer gewissen Genauigkeit gemessen werden kann.

Bei der Triangulierungsmethode wird von einem Laser ein Punkte auf das das Objekt projektiert. Dieser Punkt wird von einer Kamera erfasst. In Abhängigkeit von der

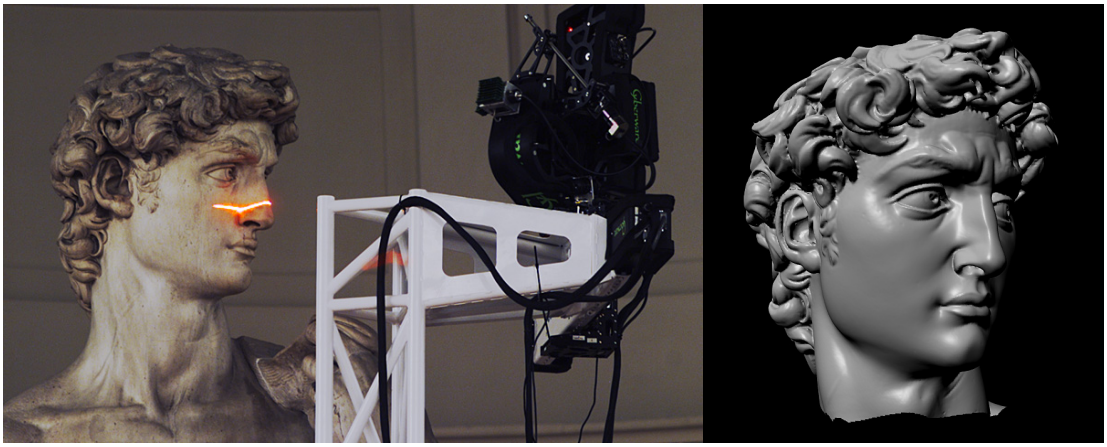

Abbildung 2.1: Triangulierungsmethode beim „The Digital Michelangelo“ Projekt

Entfernung erscheint der Laserpunkt im Sichtfeld der Kamera. Um das Verfahren zu beschleunigen kann statt einem Punkt auf eine Linie verwendet werden.(siehe Abb. 2.1)

Die Methode ist sehr genau und daher für Skulpturen gut geeignet.

2.1.2.2 Passive Rekonstruktion

Unter passiver Rekonstruktion versteht man Methoden welche nicht aktiv eine Szene abtasten, sondern vorhandene photometrische Information(z.B Photos) nutzen um die Tiefe zu berechnen.

Das Stereo Verfahren ist eines der Ersten in diesem Feld. Man geht von 2 auf der x-Achse verschoben Bildern einer Szene aus. Nun gilt es Punktpaare zwischen den beiden Bilder zu finden. Um das zu vereinfachen sucht man nach einer Abbildung von Punkten aus Bild 1 zu Bild 2. Aufgrund der Verschiebung der Bilder auf der x-Achse kann man durch Epipolargeometrie die Tiefe des Punkte berechnen.

2.1.2.3 VisualSFM

-

2.1.3 Dynamic Octree

Octrees sind eine Datenstruktur um dreidimensionale Daten hierarchisch zu untergliedern. Sie wurde 1980 von Donal Maegher beschrieben. Octrees sind im dreidimensionalen das was im eindimensionalen Binarytrees oder im zweidimensionalen die Quadtrees sind.

Jeder Knoten repräsentiert einen Würfel welcher alle eingefügten 3D Punkte beinhaltet. Jeder innere Knoten besitzt immer 8 Kinder. Diese unterteilen den Raum des Knoten in 8 gleichgroße Octanten usw. (siehe Abb. 2.2)

Die eigentlichen Punktdaten sind in den äußeren Knoten gespeichert. Äußeren Knoten können auch leer sein.

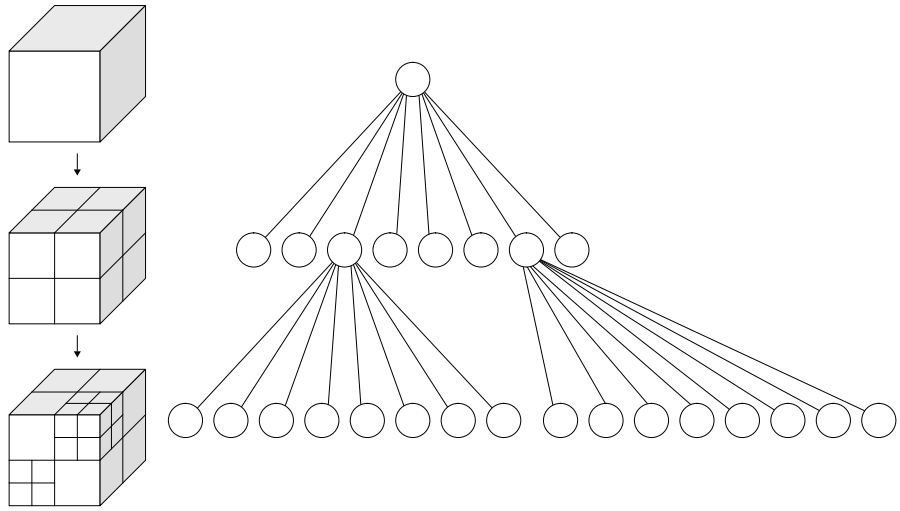

Abbildung 2.2: Schematische Darstellung eines Octrees

Der dynamische Octree unterstützt das inkrementelle Einfügen von Punkten.

2.1.4 Multiresolution Octree

Die Datenstruktur entstammt aus der Arbeit "Interactive Editing of Large Point Clouds" [11]

Die Datenstruktur unterstützt einfügen, löschen und bietet unterschiedlich detaillierte Darstellungen des Ausgangsmodells.

Im folgendem wird die Datenstruktur vorgestellt.

Der Multiresolution Octree(MRT) kann als eine spezielle Form des Octree verstanden werden. Wie beim Octree enthalten die äußeren Knoten alle Punkte.

Die Tiefe ergibt sich aus der Eigenschaft das kein Blatt mehr als n_{max} Punkte beinhalten darf. Ist n_{max} nach einer Einfüge Operation überschritten wird das Kind geteilt und die bestehenden Punkte werden auf die 8 neuen Kinder verteilt.

Die inneren Knoten sollen eine vereinfachte bzw. gröbere Version ihrer Kinder liefern.

Dafür haben diese eine dreidimensionale Rasterung gespeichert. Das Raster unterteilt den Würfel in k^3 gleichgroße Rasterzellen (z.B. k=128). Jeder Rasterzelle hat zusätzlich ein Gewicht und ein Farbe als RGB Wert gespeichert. Das Raster selbst ist nicht als einfaches Array gespeichert, sondern als Hashtabelle um Speicherplatz zu sparen. Auf diese Art werden nur die Zellen gespeichert welche Punkte enthalten.

Zellen mit hohem Gewicht werden beim rendern größer gezeichnet. Sobald ein weiterer Punkt in die gleiche Zelle fällt wird das Gewicht inkrementiert.

Der Farbwert entspricht dem des zuerst hinzugefügten Punktes der Zelle. Als Koordinate wird der Wert der Hashfunktion benutzt.

Einfügen eines Punktes Beim Einfügen können zwei Fälle auftreten.

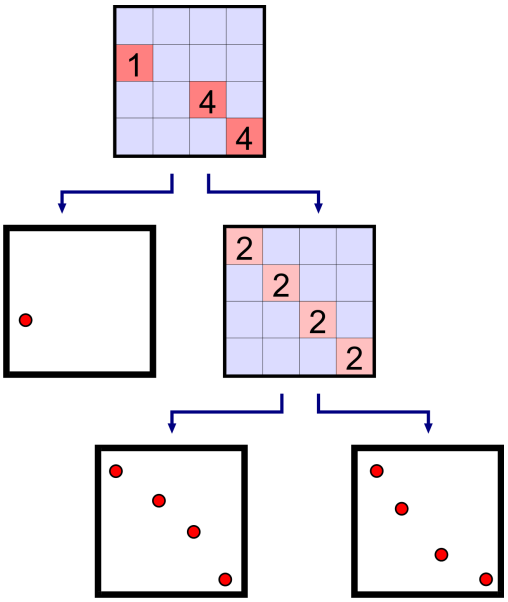

Abbildung 2.3: Schema der Rasterung

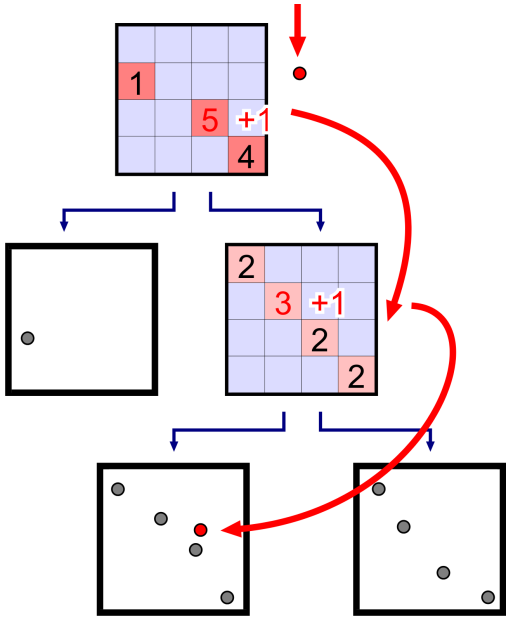

Abbildung 2.4: Schema nach einer Einfüge Operation

1. Fall : Der Punkt liegt außerhalb der Wurzel. Nun muss die bestehende Wurzel so lange erweitert werden bis sie den neuen Punkt mit einschließt.

Die Größe der Wurzel verdoppelt sich dabei bei jedem Schritt.

Sobald der Punkt in der Wurzel liegt tritt der 2. Fall ein.

2. Fall : Der Punkt liegt innerhalb der Wurzel Zuerst wird der Punkt der Rasterung hinzugefügt. Sprich das Gewicht in der entsprechende Rasterzelle wird um 1 erhöht und die Farbe des Punkten wird gegebenenfalls gespeichert. Dann wird ermittelt in welchem der Kinder der Punkt liegt. Nun wird der Vorgang beim Kind wiederholt bis ein äußerer Knoten erreicht wird. Falls die maximale Anzahl Punkte n_{max} überschritten wurde muss der Knoten gespalten werden. Alle bisher gespeicherten Punkte und der neue Punkt werden nun auf die neuen Kinder verteilt.

2.2 Verwandte Arbeiten

Das Interesse an der Darstellung von großen Punktwolken ist durch das Aufkommen von modernen 3D Scanner und der 3D Rekonstruktion in den letzten Jahrzehnt stark gewachsen. Durch den Anstieg der Leistungsfähigkeit von Tablets wurde schon einige Ansätze für die Darstellung von sehr großen Punktwolken gemacht. Allerdings kann keine bestehende Arbeit alle Anforderungen erfüllen.

2.2.1 KiwiViewer

KiwiViewer [1] ist eine freie 'open-source' App zum erkunden von Punktwolken. Die App ist für Android sowie iOS verfügbar. Multi-Touch Gestensteuerung wird unterstützt. KiwiViewer bedient sich der "Point Cloud Library"[2] für seine Kernfunktionen. Die Punktmenge wird entweder über eine SD-Karte, eMail oder URL geladen.

Abgrenzung Die App speichert die geladenen Punkte direkt auf dem Tablet. Modelle werden nicht vereinfacht. Daher treffen Modelle mit mehreren Millionen Punkten schnell an die Grenzen der GPU.

2.2.2 QSplat Verfahren

Erste Vorschläge zur Darstellung von sehr großer Punktmengen wurden durch das QSplat [9] Verfahren gemacht. Die Punkteenge wird durch eine 'Multiresolution' Hierarchie auf Basis von 'bounding spheres' modelliert. Abhängig von der Kameraposition wird die Struktur bis zu einer gewissen Tiefe (Detailstufe) durchlaufen. Das Verfahren kann modifiziert auch zum Streaming von Punktwolken genutzt werden [10] .

Abgrenzung Für HD Displays ist der Algorithmus zu rechenintensiv weil die Berechnung pro Display Punkt von der CPU erledigt wird.

2.2.3 Multiresolution Octree

Moderne Ansätze benutzen einen Multiresolution Octree [11]. Diese Datenstruktur speichert die Punkte in ihren äußeren Knoten. Die inneren Knoten beinhalten gröbere Präsentationen von ihren direkten Kindern.

Die Punktdaten werden 'out-of-core' verwaltet, also dynamisch auf die Festplatte ausgelagert sobald der RAM erschöpft ist. Die Datenstruktur bietet den Vorteil, das Punkte dynamisch hinzugefügt werden können.

Abgrenzung Das Paper beschreibt ein Verfahren für Desktop Systeme. Eigenarten von mobilen Geräten wie Netzwerkverkehr sind nicht berücksichtigt.

2.2.4 Knn-Tree iOS

Eine weitere Möglichkeit ist ein Knn-Tree zu verwenden [8]. Allerdings mit der Beschränkung, dass die Punktmenge von Anfang an vollständig ist.

Diese Arbeit richtet sich besonders an die Darstellung auf mobilen Geräten. Die Datenstruktur selbst wird auf einem Server gespeichert. Knoten werden auf Anfrage eines Klienten übertragen.

[1] `http://www.kiwiviewer.org/`
[2] `http://pointclouds.org/`

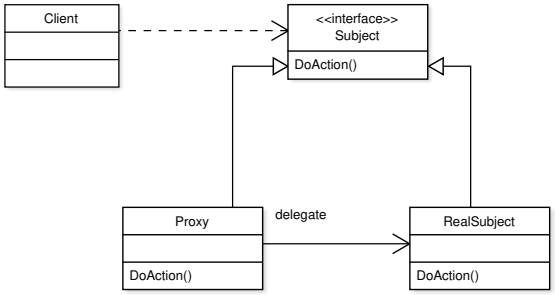

Abbildung 2.5: UML Proxy Pattern

Übertragende Knoten werden mit Hilfe eines LRU-Cache gespeichert. Die Implementierung erfolgte in C++ und OpenGl ES. Für die Netzwerkkommunikation wir 'http pipelining' sowie eine Wavletkompression verwendet.

Abgrenzung Die Datenstruktur ist nicht dynamisch, also die Menge der Punkte muss von Anfang an fest stehen. Implementierung erfolgte in C++ und für iOS. Die Datenstruktur bietet den Vorteil das sie eine geringe Tiefe besitzt.

2.3 Frameworks und Formate

Im folgendem werden die Technologien, Begriffe und Frameworks vorgestellt welche in der Arbeit verwendet werden.

2.3.1 Proxy Design Pattern

Bei einem Proxy(siehe Abb. 2.5) [5] handelt es sich um ein Objekt welches als eine Schnittstelle zu etwas anderem agiert. Zum Beispiel mit einer größeren Datei oder einer anderen teuren Ressource.

Ein klassisches Beispiel sind Platzhalter für Bilder, welche noch nicht fertig geladen wurden, auf Webseiten.

2.3.2 Representational state transfer

Representational state transfer(REST) ist ein Programmierparadigma für Service orientierte verteilte Systeme. REST wurde im Jahr 2000 von Roy Fielding in seiner Doktorarbeit [4] wie folgt beschrieben. REST definiert sich über eine Menge von Beschränkungen bei der Kommunikation von Komponenten. Das prominentest Beispiel ist das World Wide Web.

Die Beschränkungen werden im folgendem kurz vorgestellt.

Client-Server Model

Nach Andrews [2] ist der Client ein auslösender Prozess und der Server ein reagierender Prozess. Clienten stellen Anfragen auf welcher der Server reagiert. Der Klient kann entscheiden wann er mit dem Server interagiert. Der Server wiederum muss auf Anfragen warten und dann auf diese reagieren. Oft ist ein Server auf nicht endender Prozess welcher auf mehrere Clienten reagiert.

Zustandslosigkeit

Jede Anfrage vom Clienten muss alle Informationen enthalten welche Notwendig sind um die Anfrage zu verarbeiten. Des weiteren darf kein gespeicherter Kontext auf dem Server vorliegen auf welchen Bezug genommen wird. Alle Zustände werden auf dem Klienten gespeichert.

Caching

Server Antworten müssen implizit oder explizit als cachebar gekennzeichnet sein. Die Idee ist den Netzwerkverkehr effizienter zu machen. Bemerkenswert dabei ist das dadurch ganze Interaktionen wegfallen können.

Einheitliche Schnittstelle

Ein integraler Bestandteil einer REST Architektur ist eine einheitliche Schnittstelle. Das vereinfacht die System Architektur und die Sichtbarkeit von Interaktionen ist verbessert. Sie ist durch 4 weitere Eigenschaften beschrieben.

Adressierbarkeit von Ressourcen Jede Information, die über einen URI kenntlich gemacht wurde, wird als Ressource gekennzeichnet. Die Ressource selbst wie in einer Repräsentation übertragen welche sich von der internen Repräsentation unterscheidet. Jeder REST-konforme Dienst hat eine eindeutige Adresse, den Uniform Resource Locator (URL).

Repräsentationen zur Veränderung von Ressourcen Wenn ein Client die Repräsentation einer Ressource besitzt mit seinen Metadaten, reicht dies aus um die Ressource zu modifizieren bzw zu löschen.

Self-descriptive messages Jede Nachricht enthält beschreibt wie seine Informationen zu verarbeitet sind. Z.B durch Angabe seine Internet Media Types(MIME-Type)

„Hypermedia as the Engine of Application State“ Bei „Hypermedia as the Engine of Application State“ HATEOAS navigiert der Client einer REST-Schnittstelle ausschließlich über URLs, welche vom Server bereitgestellt werden. Abhängig von der gewählten Repräsentation geschieht die Bereitstellung der URIs über Hypermedia. Abstrakt betrachtet stellen HATEOAS-konforme REST-Services einen endlichen Automaten dar,

dessen Zustandsveränderungen durch die Navigation mittels der bereitgestellten URIs erfolgt. Durch HATEOAS ist eine lose Bindung gewährleistet und die Schnittstelle kann verändert werden.

Mehrschichtige Systeme

Der Klient soll lediglich die Schnittstelle kennen. Schichten dahinter bleiben ihm verborgen.

2.3.3 Google Protocol Buffers

Bei Google Protocol Buffers[3] handelt es sich um eine Plattform unabhängige Datenstruktur von Google. Die Datenstruktur wird in einem einheitlichen Format festgelegt. Nun kann für jede unterstützte Sprache Klassen generiert werden welche die Daten (de)serializieren. Das macht es möglich Clienten in anderen Sprachen zu schreiben. Ausserdem verringert es den Overhead merklich im Vergleich zu JSON oder XML. Nachteilig ist das die übertragenden Daten nicht ohne weiteres für einen Menschen lesbar sind.

2.3.4 Open Graphics Library for Embedded Systems(OpenGL ES)

OpenGL Es ist eine sprachenunabhängige Programmierschnittstelle zur Entwicklung von 3D-Computergrafik. Die Spezifikation beschreibt eine vereinfachte Version der OpenGL-Spezifikation, welche sich besonders für den Einsatz im Bereich von eingebetteten Systemen eignet. [1]

2.3.5 NanoHTTPD

NanoHTTPD[4] bezeichnet sich selbst als einen schlanker HTTP Server welche darauf ausgerichtet ist sich einfach in bestehende Anwendungen einbetten zu lassen.

Das Project ist Open Source und wird aktiv auf github entwickelt.

2.3.6 la4j

La4j [5]ist eine offene Java Bibliothek für Lineare Algebra.

2.3.7 DEFLATE

Bei DEFLATE [3] handelt es sich um einen Kompression Algorithmus von Phil Katz aus dem Jahre 1993. Er kombiniert die die Kompressions Algorithmen LZ77 oder LZSS mit einer Huffman Kodierung. Der Algorithmus zeichnet such durch eine solide Kompression in kurzer Zeit aus. Der Algorithmus findet zum Beispiel im .png Format Anwendung.

[3] `https://developers.google.com/protocol-buffers/`
[4] `https://github.com/NanoHttpd/nanohttpd`
[5] `http://la4j.org/`

2.3.8 Volley

Bei Volley [6] handelt es sich um ein HTTP Bibliothek zum Verwalten von Netzwerkanfragen. Volley bietet automatisches koordinieren von Anfragen. Es ermöglicht mehrere neben läufige Netzwerkverbindungen, Anfragen Priorisierung und einiges mehr.

Die Bibliothek ist frei und wird unter diesem Link [7] entwickelt.

[6] `http://developer.android.com/training/volley/index.html`
[7] `https://android.googlesource.com/platform/frameworks/volley`

3 Gewählter Lösungsansatz

Mobile Geräte verfügen über begrenzte Ressourcen was Rechenleistung und Speicher angeht. Punktwolken mit mehr als einer Million Punkte bringen die GPU schnell an Ihre Grenzen. Daher wurde ein Ansatz gewählt der es ermöglicht präzise Punkte auszuwählen. Darin unterscheiden wir zwischen 2 Dimension. Einmal die Sichtbarkeit und zum anderen die Detailstufe der Punkte. Zum ermitteln der Punkt unter diesen Kriterien kommt ein Multi-Resolution Octree(siehe Section 2.1.4) zum Einsatz. Dadurch wird die GPU effizient eingesetzt.

Die Defizite beim Speicher werden durch eine Client-Server Architektur ausgeglichen. Das eigentliche Model wird von einem Server verwaltet. Der Mobile Client hat selber nur den momentan benötigten Satz Punkte gespeichert. Um Netzwerkverkehr niedrig zu halten werden die Punkte von Client in einem Cache nach dem „Last Recently Used“ Prinzip gespeichert. Zum Rendern werden die Punkte als „vertex buffer object“ in den Speicher der GPU geschrieben und anschließend gezeichnet.

Also Programmiersprache wurde Java gewählt aufgrund ihres guten Kompromisses aus Portabilität und Performance.

3.1 Architektur

3.1.1 Server

Der Server hat drei Aufgaben. Zum ersten erstellt er aus gegebenen Punkten einen MRT. Seine zweite Aufgabe ist es aus den Knoten des MRT auf Anfrage zu verschicken und als letztes einen eine Proxy Objekt (siehe Kapitel 2.3.1) des MRTs an die Klienten zu verteilen.

Zur Interaktion stellt der Server ein Interface auf Basis des HTTP Protokolls bereit.

3.1.1.1 Multi Resolution Tree

Der MultiResolution Tree besteht aus 4 Klassen (siehe Abb. 3.1). Die Klasse “MultiResolutionTree” dient als Schnittstelle für alle äußeren Komponenten. Dadurch ist eine sinnvolle Kapselung gewährleistet. Sie beherbergt einen Zeiger auf den Wurzelknoten des MRT. Des weiteren legt sie einen Index von den Knoten an um schnellen Zugriff zu ermöglichen. Zusätzlich existiert einer Factory Methode zum erstellen von Google Protocol Buffer Objekten.

Die Implementierung des MRT ist analog zu der Beschreibung in Kapitel2.1.4. Jeder Knoten besitzt eine ID. Diese sind die Koordinaten seines absoluten Mittelpunkts. Erwähnenswert sind die Entscheidungen bei der Raster Klasse. Die Rasterwerte sind in

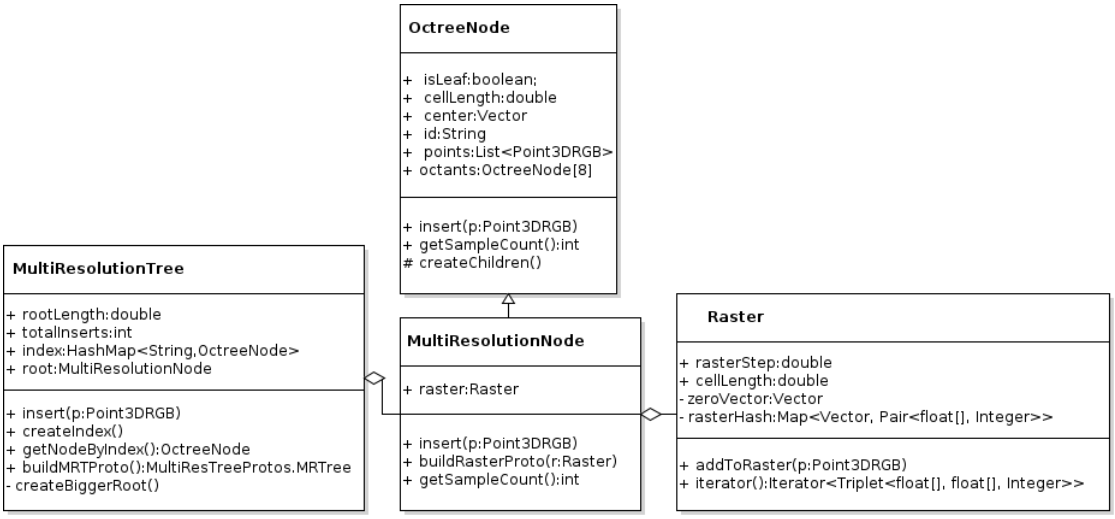

Abbildung 3.1: Multiresolution Tree UML

einer interne DefaultHashMap vermerkt. Diese bildet einen dreidimensionalen Vektor auf Farbwerte und Gewicht ab. Bei einer DefaultHashMap werden bei Abfrage von nicht existierender Einträge Standartwerte zurück gegeben. Dadurch wird im Vergleich zu einem n^3Array Speicherplatz gespart. Die eigentliche Rastervorgang wird durch die folgende Hashfunktion erreicht.

$$H(p) = (\left\lfloor \frac{x(p)}{cellLength} \right\rfloor, \left\lfloor \frac{y(p)}{cellLength} \right\rfloor, \left\lfloor \frac{z(p)}{cellLength} \right\rfloor)$$

wobei die Koordinate von p relativ zu Ursprung des Würfel ist. Die Variable$cellLength$ entspricht der Länge einer Gitterzelle. Also:

$$cellLength = \frac{cube.length}{k}$$

Einfach gesprochen werden die Koordinaten auf ein Vielfaches der $cellLength$ abgerundet.

Damit schnelle Zugriffe bei Anfragen auf die entsprechenden Knoten bzw. Punkte möglich sind existiert eine Hashmap als Index. Dieser bildet Ids auf die entsprechenden Knoten ab. Der Index wird in nach einer festen Anzahl Einfüge Operationen aktualisiert.

3.1.1.2 RESTful API

Die API wurde mit Hilfe des NanoHTTPD Frameworks (siehe Kapitel 2.3.5) implementiert. NanoHTTPD wurde ausgewählt weil es schlank und einfach ist.

Prinzipiell wird bei jeder Anfrage eine Fallunterscheidung an dem “mode” Parameter gemacht. Die Anfragen werden im folgenden vorgestellt.

3.1.1.3 MRT-Proxy Anfrage

Da die Ermittlung gebrauchten Punkte auf dem Klienten geschieht muss auch dieser in Kenntnis über die Struktur der MRTs sein. Aus diesem Grund Stellt der Server ein MRT-Proxy zur Verfügung. Der MRT-Proxy ist im Prinzip der gespeicherte Multiresolution Octree allerdings haben seine Knoten keine Punkte bzw Rasterung gespeichert, sondern nur Ids um beim Server die entsprechenden Punkte anzufragen. Diese können entweder Original Punkte aus Blättern sein oder Punkte aus der Rasterung.

Zum versendet wird aus der Datenstruktur ein Protocol Buffer Objekt erstellt. Diese kann dann problemlos serialisiert werden. Das Protocol Buffer Objekt ist wie folgt definiert.

```
1  package DataAccesLayer;
2
3  option java_outer_classname = "MultiResTreeProtos";
4
5  message MRTree{
6          required MRNode root = 1;
7          message MRNode {
8                  required string id = 1;
9                  repeated double center = 2 [packed=true];
10                 required double cellLength = 3;
11                 required int32 pointCount = 4;
12                 required bool isLeaf = 5;
13                 repeated MRNode octant = 6;
14         }
15 }
```

3.1.1.4 Punktanfrage

Der 2. Typ Anfragen liefert Punkte an den Klienten aus. Bei dieser Anfrage wird vom Klienten immer eine Id als Parameter mitgesendet. Diese Id passt auf einen Knoten des MRT.

Falls es sich um ein inneren Knoten handelt wird die Rasterung zu einer Liste von Punkten exportiert. Beim einem Blatt wird lediglich auch die vorhandene Punktliste zugegriffen.

Der Server greift über die MultiResolutionTree Klasse auf den entsprechende Knoten zu und generiert ein Protocol Buffer Objekt. Die Spezifikation des Objektes lautet:

```
1  package DataAccesLayer;
2
3  option java_outer_classname = "RasterProtos";
4
5  message Raster{
6          repeated Point3DRGB sample = 1;
```

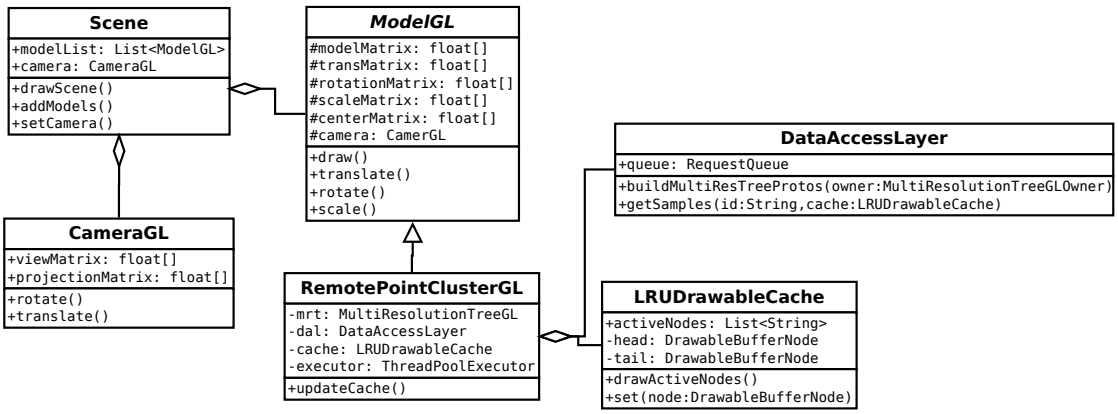

Abbildung 3.2: Client UML

```
7           message Point3DRGB{
8                   repeated float position = 1;
9                   repeated float color = 2;
10                  required int32 size = 3;
11          }
12  }
```

Es handelt sich also um eine Liste von Punkten mit Positions-, Farb- und einer Gewichtswerten (size).

Alle Protocol Buffer Objekte werden bevor sie über das Netzwerk versendet werden serialisiert und durch den DEFLATE Algorithmus komprimiert, um den Netzwerkverkehr möglichst niedrig zu halten.

3.1.2 Client

Bei dem Klienten handelt es um eine Android Anwendung. Der Klient ist verantwortlich für das Darstellen der Punktwolke und reagiert auf Eingaben des Users.

Der Client verfolgt eine eventbasierte Architektur. Wird vom Nutzer eine Translation(2-Finger Geste) oder Rotation(Slide Geste) an dem Punktmodel ausgeführt kann sich die Menge der zu zeichnenden Punkte ändern.

Die Menge der momentan zu zeichnenden Punkte wird im folgenden als aktive Punkte bezeichnet. Die Knoten welche die aktiven Punkte beinhalten werden als aktive Knoten bezeichnet.

3.1.2.1 Scene Klasse

Die Scene Klasse siehe(Abbildung 3.2)ist die Schnittstelle für alle Rendering relevanten Aktionen. Nutzereingaben werden von dieser Klasse entgegengenommen und entsprechend weitergeleitet. Die Kamera mit Projektions Matrix ist hier gespeichert.

Alle zu zeichnenden Objekte werden in modelList gespeichert. Bei Aufruf der drawScene Methode wird die draw Methode von jedem Element der Liste ausgelöst.

3.1.2.2 RemotePointClusterGL

RemotePointClusterGL kümmert sich um das Ermitteln und Speichern aktiver Punkte im MRT, dafür hat sie einen Verweise auf den Cache und dem MRT-Proxy. Desweiteren greift die Klasse auf den DataLayer zu um entweder nach dem MRT Proxy zu fragen oder nach neuen Punkten.

3.1.2.3 Ermittlung der aktiven Knoten

Bevor etwas gerendert werden kann, muss bestimmt werden, welche Punkte sichtbar sind und in welcher Detailstufe man diese darstellen möchte. Dafür ist der MRT-Proxy nötig, daher wird dieser beim Start der App angefordert (siehe Kapitel 3.1.1.3).

Nach jeder Kamerabewegung kann sich die Menge der aktiven Punkte ändern, dann wird update cache ausgeführt. Rotiert der Nutzer etwa die Szene wird der MRT-Proxy mit Hilfe des folgenden Algorithmus traversiert um die Ids von aktiven Knoten zu ermitteln.

```
1  public List<String> getIdsViewDependent(){
2          List<String> ids = new LinkedList<>();
3          _getIdsViewDependent(root, ids);
4          return ids;     V
5  }
6
7  private void _getIdsViewDependent(OctreeNodeGL currentNode,
       List<String> ids) {
8          if ((currentNode.isLeaf ||
9          currentNode.getDetailFactor(this.owner) <
               DETAIL_THRESHOLD)){
10                 ids.add(currentNode.id);
11                 return;
12         }
13         for (OctreeNodeGL node : currentNode.octants ) {
14                 if (node.isVisible(owner) && node.
                       pointCount > 0)
15                         _getIdsViewDependent(node, ids);
16         }
17 }
```

Salop gesagt besuche alle sichtbaren Knoten solange bis entweder ein Blatt erreicht ist oder eine ausreichende Detailstufe.

Der Vorgang findet in eigenem Thread statt damit der Nutzer nicht warten muss um neue Eingabe zu machen. Für die Verwaltung weiterer Threads kommt die von Java mitgelieferte ThreadPoolExecutor Klasse zum Einsatz.

Nachdem alle nötigen Knoten festgestellt wurden, wird geprüft ob sich diese schon im Cache befinden. Falls nicht müssen diese vom Server angefordert werden.

3.1.2.4 DataAcessLayer Klasse

Zum asynchronen nachladen von Punkten kommt das „Volley Framework“. (siehe Kapitel 2.3.8). Das Framework stellt ein RequestQueue zur Verfügung. Volley ermöglicht es die Callback Funktion beim Eintreffen einer Anfrage zu überschreiben, um eigene Logik zu definieren. Sobald ein Anfrage empfangen wurde wird diese entpackt und deserialsiert. Anschliessend wird wenn es sich um Punktdaten handel dem Cache hinzugefügt oder bei einem Proxy Objekt wird das alte überschrieben.

3.1.2.5 Rendering

Das Rendering ist mit OpenGL realisiert. Jeder Knoten im Cache zeigt auf einem Buffer gefüllt mir Positions-, Farb- und Gewichtswerten. Beim zeichen werden alle aktiven Knoten ,welche im Cache sind, gezeichnet. Dabei wird jeder Buffer einmalig als Vertex Buffer Object(VBO) in den Speicher der GPU geschrieben. Das verbessert die Performance enorm weil so nicht in jedem Frame die Daten neu übertragen werden müssen.

-

4 Experimente und Auswertung

-

Abbildungsverzeichnis

Literaturverzeichnis

[1] Opengl es wiki. `https://de.wikipedia.org/wiki/Open_Graphics_Library_for_Embedded_Systems`, note = Accessed: 2016-04-12.

[2] Gregory R. Andrews. Paradigms for process interaction in distributed programs. *ACM Comput. Surv.*, 23(1):49–90, March 1991.

[3] P. Deutsch. Deflate compressed data format specification version 1.3, 1996.

[4] Roy Thomas Fielding. *Architectural Styles and the Design of Network-based Software Architectures*. PhD thesis, 2000. AAI9980887.

[5] Erich Gamma, Richard Helm, Ralph Johnson, and John Vlissides. *Design Patterns: Elements of Reusable Object-oriented Software*. Addison-Wesley Longman Publishing Co., Inc., Boston, MA, USA, 1995.

[6] Denis Klimentjew. *Grundlagen und Methodik der 3D-Rekonstruktion und ihre Anwendung für landmarkenbasierte Selbstlokalisierung humanoider Roboter*. PhD thesis, 2008.

[7] Stawros Ladikos et al. *Real-Time Multi-View 3D Reconstruction for Interventional Environments*. PhD thesis, Technische Universität München, 2011.

[8] Marcos Balsa Rodriguez, Enrico Gobbetti, Fabio Marton, Ruggero Pintus, Giovanni Pintore, and Alex Tinti. Interactive Exploration of Gigantic Point Clouds on Mobile Devices. In David Arnold, Jaime Kaminski, Franco Niccolucci, and Andre Stork, editors, *VAST: International Symposium on Virtual Reality, Archaeology and Intelligent Cultural Heritage*. The Eurographics Association, 2012.

[9] Szymon Rusinkiewicz and Marc Levoy. Qsplat: A multiresolution point rendering system for large meshes. In *Proceedings of the 27th Annual Conference on Computer Graphics and Interactive Techniques*, SIGGRAPH '00, pages 343–352, New York, NY, USA, 2000. ACM Press/Addison-Wesley Publishing Co.

[10] Szymon Rusinkiewicz and Marc Levoy. Streaming qsplat: A viewer for networked visualization of large, dense models. In *Proceedings of the 2001 Symposium on Interactive 3D Graphics*, I3D '01, pages 63–68, New York, NY, USA, 2001. ACM.

[11] Michael Wand, Alexander Berner, Martin Bokeloh, Philipp Jenke, Arno Fleck, Mark Hoffmann, Benjamin Maier, Dirk Staneker, Andreas Schilling, and Hans-Peter Seidel. Special section: Point-based graphics: Processing and interactive editing of huge point clouds from 3d scanners. *Comput. Graph.*, 32(2):204–220, April 2008.