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:

18. Mai 2016

Eidesstattliche Erklärung

zur Bachelorarbeit

Name: _____ Vorname: _____

Ich versichere, die Bachelorarbeit selbständig und lediglich unter Benutzung der angegebenen Quellen und Hilfsmittel verfasst zu haben.

Ich erkläre weiterhin, dass die vorliegende Arbeit noch nicht im Rahmen eines anderen Prüfungsverfahrens eingereicht wurde.

Berlin, den _____

(Unterschrift)

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, Modele 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 Freier 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

2.1 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, [7] 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 Archaeocopter Projekt.

Allgemein kann man zwischen aktiver und passiver Rekonstruktion [8] unterscheiden.

2.1.1 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. Im Grunde sind sie der Kamera ähnlich. Genau wie diese besitzen sie ein Sichtfeld. Allerdings liefern sie statt Farbwert Abstandswert von ihrem 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 Entfernung erscheint der Laserpunkt im Sichtfeld der Kamera. Um das Verfahren zu beschleunigen kann statt einem Punkt eine Linie verwendet werden.(siehe Abb. 2.1)

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

2.1.2 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.

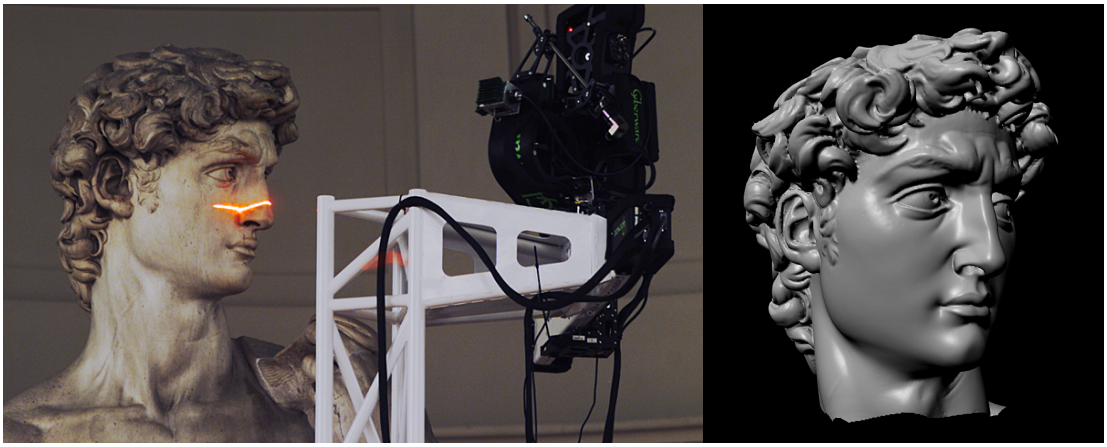

Abbildung 2.1: Triangulierungsmethode beim „The Digital Michelangelo“ Projekt. Quelle: `http://graphics.stanford.edu/projects/mich/`

2.1.3 VisualSFM

Bei VisualSFM[1] handelt es sich um ein 3D Rekonstruktionssystem. Es verfügt über eine grafische Benutzeroberfläche und zeichnet sich mit Skalierbarkeit durch Nutzung der nVidia oder ATI Grafikkarten des Computers aus.

Entwickelt wurde das Programm von dem Studenten Changchang Wu[2], welcher mittlerweile bei google beschäftigt ist. VisualSFM wird von dem Archaeocopter Projekt zur 3D Rekonstruktion verwendet.

2.2 Repräsentationen von Punktwolken

Punkt basierende geometrische Oberflächen können als Stichprobenmenge einer kontinuierlichen Oberfläche verstanden werden. Dabei entstehen dreidimensionale Raumkoordinaten $p_i \in \mathbb{R}^3$. Oft existieren noch weiter Daten zu dem Punkt wie eine Normale n_i oder eine Farbe c_i. Eine Punktwolke S ist eine Menge solcher Punkte.

2.2.1 QSplat Verfahren

Erste Vorschläge zur Darstellung von sehr großer Punktmengen wurden durch das QSplat Verfahren gemacht [12]. Splat bedeutet Klecks wie von Farbklecks.

Die Punktmenge wird durch eine Hierarchie auf Basis von Kugeln modelliert. Jede Kugel repräsentiert einen Knoten in einem binär Baum. Jeder Knoten enthält den Mittelpunkt seiner Kugel, den Radius, die Normale und die Breite des Normalen Kegels und optional eine Farbe. Die Datenstruktur wird zu Beginn erstellt. Der Konstruktionsalgorithmus kann entweder auf einer Punktwolke oder besser einem triangulierten Modell

[1] `http://ccwu.me/vsfm/`
[2] `http://ccwu.me/`

angewendet werden. Bei letzterem ist es leichter die Normalen zu berechnen. Für den Radius der Kugeln wird die Länge der Längsten anliegenden Kante gewählt. Mit Hilfe folgendem Algorithmus kann nun die Datenstruktur erstellt werden.

```
1 BuildTree(vertices[begin..end]) {
2   if (begin == end)
3     return Sphere(vertices[begin])
4   else
5     midpoint = PartitionAlongLongestAxis(vertices[begin..end])
6     leftsubtree  = BuildTree(vertices[begin..midpoint])
7     rightsubtree = BuildTree(vertices[midpoint+1..end])
8     return BoundingSphere(leftsubtree, rightsubtree)
9 }
```

Sobald die Datenstruktur aufgebaut ist, kann die Punktmenge gezeichnet werden mit folgendem Algorithmus

```
1  TraverseHierarchy(node) {
2    if (node not visible)
3      skip this branch of the tree
4    else if (node is a leaf node)
5      draw a splat
6    else if (benefit of recursing further is too low)
7      draw a splat
8    else
9      for each child in children(node)
10       TraverseHierarchy(child)
11 }
```

Zuerst wird getestet ob die Kugel des Knoten sichtbar ist. Dafür wird getestet ob die Kugle in dem View Frustum liegt. Falls nicht muss der Knoten und seine Kinder nicht beachtet werden und die Rekursion wird abgebrochen. QSplat führt auch Backface Culling aus mit Hilfe der Normalen Kegeln.

QSplat entscheidet anhand der Größe der projektierten Kugel auf die View-Ebene ob die Rekursion fortgesetzt wird. Überschreitet diese einen gewissen Schwellwert wird die Rekursion beendet und der aktuelle Knoten wird gezeichnet.

Das Verfahren kann modifiziert auch zum Streaming von Punktwolken genutzt werden [13] .

2.2.2 Octree

Octrees sind eine Datenstruktur um dreidimensionale Daten hierarchisch zu untergliedern. Sie wurde 1980 von Donal Maegher beschrieben [10]. Octrees sind Analog im dreidimensionalen zu Quadtrees im zweidimensionalen.

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

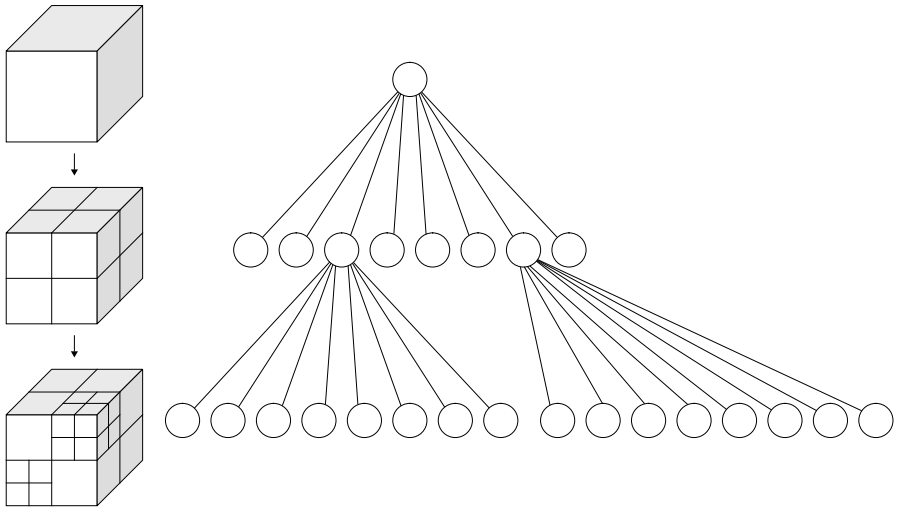

Abbildung 2.2: Schematische Darstellung eines Octrees Quelle:?

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

Der Octree unterstützt das Einfügen von Punkten.

2.2.3 Multiresolution Octree

Die Datenstruktur entstammt aus der Arbeit "Interactive Editing of Large Point Clouds" [14]

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.

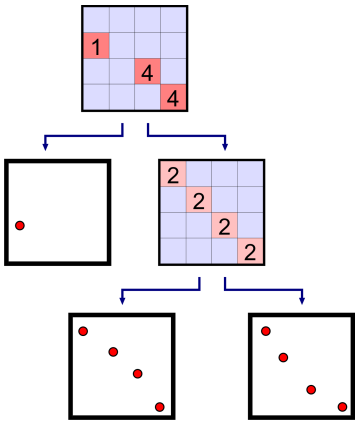

Abbildung 2.3: Schema der Rasterung. Quelle: [14]

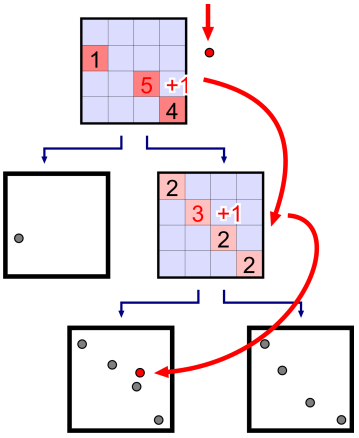

Abbildung 2.4: Schema nach einer Einfüge Operation. Quelle:[14]

Einfügen eines Punktes Beim Einfügen können zwei Fälle auftreten.

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 Breite sowie Höhe der Wurzel verdoppelt sich dabei in 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.

2.2.4 Surface Splats

-

2.2.5 Moving Least-Squares Surfaces

Das moving least-squares (MLS) Verfahren wurde von Levin 1998 vorgestellt [9]. Das Verfahren wurde erstmalig 2001 durch Alexa zur Darstellung von Punktwolken benutzt [1]. Die Grundidee der Arbeit ist, dass die gegebene Menge Punkte S indirekt eine Oberfläche S_A definiert. Es wird eine Projektion $\phi : U \to \mathbb{R}^3$ vorgestellt welche einen beliebigen Punkt aus der Umgebung U von S auf eine Oberfläche S_A, welche das Objekt lokal beschreibt, projektiert. Alle Punkt die auf sich Selbst abbilden ergeben die Abgeschätzte

Oberfläche S_A.

$$S_A := \{x \in \mathbb{R}^3 | \phi(x) = x\}$$

U kann man als eine Vereinigung von Kugeln mit Radius r_k, dessen Mittelpunkt ein Punkt aus S ist, beschreiben.

$$U := \bigcup_i \{x \in \mathbb{R}^3 | \|x - p_i\| < r_K\}$$

Die Projektion eines Punktes r wird in 3 Schritten ermittelt.

1. Ermittle eine Referenzebene $H = \{x | \langle n, x\rangle - D = 0, x \in \mathbb{R}^3\}, n \in \mathbb{R}^3, \|n\| = 1$ durch Minimierung des Ausdrucks

   $$\sum_{i=1}^{N} (\langle n, p_i\rangle - D)^2 \Theta(\|p_i - q\|)$$

   wobei q die Projektion von r auf H ist. Bei Θ handelt es sich um eine monoton, radial, fallende Funktion mit positiven Wertebereich. Typischer weise $\Theta(d) = e^{\frac{d^2}{h^2}}$. h ist typischer Weise der durchschnittliche Abstand von benachbarten Punkten und hat einen direkten Einfluss wie glatt die Oberfläche erscheint.

2. Mit Hilfe von H kann nun ein zwei zweidimensionales Polynom zum Abschätzen der Umgebung von S_A in der Nähe von r gefunden werden.

   $$\sum_{i=1}^{N} (g(x_i, y_i) - f_i)^2 \Theta(\|p_i - q\|))$$

   Hier ist f_i der kürzeste Weg von H zu p_i. Bei $x_i$und $y_i$handelt es sich um die Koordinaten von Punkt q_i.

3. Die Projektion von r ist schlussendlich wie folgt definiert.

   $$\phi(r) = q + g(0, 0)n$$

Neben dem Darstellen von Punkten kann das Verfahren auch eingesetzt werden um weitere Punkte zu generieren.

Beim rendern wird eine Datenstruktur wie bei dem QSplat Verfahren verwendet. Unterscheiden tun sich die Blätter. In denen werden zusätzlich zur Position, Radius, Normalen und Farbe auch eine Referenzebene H sowie die Koeffizienten des Polynoms gespeichert.

Wenn bei dem Render Vorgang ein Blatt erreicht wird und wenn mehr als ein Punkt benötigt wird werden mit Hilfe des Polynoms weiter Punkte für die Umgebung generiert.

2.3 Minimum bounding box

In der Geometrie versteht man unter einer „minimum bounding box“ ein Rechteck(2d) oder Box(3d) welches eine Menge von Punkten umschließt. Dabei ist der die Fläche bzw. das Volumen minimal.

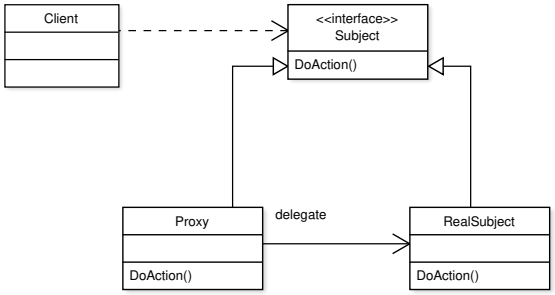

Abbildung 2.5: UML Proxy Pattern

2.4 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.

2.5 Singleton Pattern

-

2.6 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 Client 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 Clienten 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 Client soll lediglich die Schnittstelle kennen. Schichten dahinter bleiben ihm verborgen.

3 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.

3.1 KiwiViewer

KiwiViewer [1] ist eine freie quelloffene Applikation zur Erkundung 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.

3.2 LiMo

Bei LiMo handelt es sich um eine von OGSystems entwickelte Android Applikation zum Betrachten von LiDAR Daten. LiMo wurde für den professionellen Gebrauch entwickelt und ermöglicht das Beobachten von Gebäuden sowie Naturszenen. Daten können auch über einen eigen Webservice gestreamt werdenm weshalb die App auch zu Monitoring Zwecken eingesetzt werden kann.

Die Applikation unterstützt bis zu 5 Millionen Punkte.

Abgrenzung Die App ist auf LiDAR Daten beschränkt. LiDAR ist primär zum scannen von großen Objekten, wie Brücken , geeignet und verfügt daher nicht über die Genauigkeit welche Beispielsweise bei einer Skulptur benötigt wird.

3.3 QSplat Verfahren

Erste Vorschläge zur Darstellung von sehr großer Punktmengen wurden durch das QSplat [12] Verfahren gemacht. Die Punkteenge wird durch eine 'Multiresolution' Hierarchie

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

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 [13] .

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

3.4 Multiresolution Octree

Die Arbeit [14] war eine Kooperation zwischen der Stanford Universität und der Universität Tübingen.

Die Veröffentlichung stellte einen Multiresolution Octree (siehe Kapitel 2.2.3) gekoppelt mit out-of-core Mechanismen vor zur Darstellung von enorm großen Punktwolken vor. Für das das out-of-core Verfahren wird ein Least recently used Cache verwendet. Sind angeforderte Daten nicht im Cache enthalten werden sie von der Festplatte geladen. Dadurch ist die Größe der Punktdaten lediglich nur durch dieses Speichermedium beschränkt. Anders als bestehende Ansätze bietet die Datenstruktur Operation wie Löschen und Einfügen von Punkten unabhängig von der Komplexität der Szene. Des weiteren wird ein System zum Bearbeiten von Punktwolken diskutiert.

Abgrenzung Die Arbeit beschreibt ein Verfahren für Desktop Systeme. Eigenarten von mobilen Geräten werden nicht berücksichtigt. Das Verfahren ist sehr speicherintensiv und kann daher nicht ohne weiter auf ein mobiles Gerät übertragen werden.

3.5 Knn-Tree iOS

Eine weitere Möglichkeit ist ein Knn-Tree zu verwenden [11]. 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 Clients übertragen.

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.

4 Frameworks, Programmiersprachen und Laufzeitumgebung

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

4.1 Java

Bei Java [6] handelt es sich um eine 1995 von Sun Microsystems veröffentliche Programmiersprache. Entwickelt wurde die Sprache von James Gosling. Java ist Objektorientiert, nebenläufig und plattformunabhängig. Letzteres wird erreicht indem Java Code, genauer gesagt Java Byte Code, auf einer virtuellen Maschine (JVM) interpiretiert wird. Moderne Implementierungen der JVM unterstützen sogenannte „Just in Time“ Kompilierung. Das bedeutet, dass eine Übersetzung in Maschinencode während der Laufzeit vorgenommen wird.

Des weiteren kümmert die Laufzeitumgebung von Java um das Speicher Management. Nach dem Tiobe Index zu urteilen ist Java eine der populärsten Programmiersprachen der Welt[1].

4.2 Android

Android[2] ist ein mobiles Betriebssystem welches momentan von Google entwickelt wird. Es wurde mit 33 Mitgliedern der Open Handset Alliance entwickelt. Ziel war es einen offenen Standard für mobile Geräte zu schaffen[3].

Android baut auf dem Linux Kernel auf und ist für eine Bedienung über Touchdisplays ausgelegt. Daher werden Eingaben hauptsächlich über Gesten und Tippen am Display vorgenommen. Android verfügt über Ableger für Fernseher(Android TV), Autos (Android Auto) sowie Smartwatches(Android Wear). Android hat seit mehreren Jahren einen dominanten Marktanteil[4] bei mobile Geräten und ist das mit Abstand meist genutzt Betriebssystem weltweit[5].

Java ist die bevorzugte wenn auch nicht einzig mögliche Programmiersprache für Android. Allerdings kommt bei Android keine Standard JVM zum Einsatz sondern eine

[1] http://www.tiobe.com/tiobe_index?page=Java
[2] https://www.android.com/
[3] http://www.openhandsetalliance.com/press_110507.html
[4] http://www.idc.com/prodserv/smartphone-os-market-share.jsp
[5] https://en.wikipedia.org/wiki/Usage_share_of_operating_systems

modifizierte Version. Bis Android 4.4 kam die Virtuelle Maschine Dalivk zum Einstatz. Diese wurde vollständig in Android 5.0 ersetzt durch Android Runtime(ART)[6].

4.3 Google Protocol Buffers

Bei Google Protocol Buffers[7] handelt es sich um eine Plattform unabhängige Datenstruktur zum Serialisieren von Daten.

Die Datenstruktur wird in einem einheitlichen Schema festgelegt. Zum Beispiel:

```
1 message Person {
2   required string name = 1;
3   required int32 id = 2;
4   optional string email = 3;
5 }
```

Das Schema definiert Namespaces über diese dann die eigentlich Daten ausgelesen werden können. In dem Beispiel hat der Datentyp Person drei Felder. Jedes dieser Felder ist getypt und mit einem Schlüsselwort versehen welches angibt ob das Feld Pflicht ist.

Es besitzt also jede Person einen String für den Namen aber nicht immer einen String für die Mail.

Aus dem Schema kann für jede unterstützte Programmiersprache(Java, C++, Python, JavaNano, Ruby, Objective-C und C#) Klassen generiert werden welche die Daten (de)serialisieren. Das macht es möglich Clienten in anderen Sprachen zu schreiben.

Protocol Buffers sind komplett abwärtskompatibel. Die Schema können problemlos erweitert werden.

Ein weiter Vorteil ist der merklich verringerte Overhead im Vergleich zu JSON oder XML. Nachteilig ist das die übertragenden Daten nicht ohne weiteres für einen Menschen lesbar sind. Protocol Buffers werden intensiv intern bei Google selbst eingesetzt. Ein weiter prominenter Nutzer ist Blizzard beim „BATTLE.NET“ [8].

4.4 Open Graphics Library for Embedded Systems

Open Graphics Library for Embedded Systems (OpenGL ES) bietet ein offenes Interface für Grafikhardware. Dieses besteht aus einer Sammlung von Prozeduren und Funktion die es dem Programmierer ermöglichen Shaderprgramme, Objekte und Operation zu spezifizieren um dreidimensionale Farbbilder zu produzieren. Viele der Funktion von OpenGL ES implementieren das Zeichnen von geometrischen Objekten wie Punkte oder Linien. OpenGL ES setzt voraus das die Grafikhardware über einen Framebuffer verfügt.

[6] http://developer.android.com/about/versions/android-5.0-changes.html
[7] https://developers.google.com/protocol-buffers/
[8] https://news.ycombinator.com/item?id=11444846

4.4.1 Vertex

Unter einem Vertex(plural die Vertices) versteht man eine Datenstruktur welche einen zwei- oder dreidimensinalen Punkt im Raum beschreibt.

4.4.2 Shader

Bei Shadern handelt es sich um vom Nutzer geschriebene Programme welche an unterschiedlichen Station der Rendering Pipeline aufgerufen werden. Shader werden in der C nahen OpenGL Shading Language geschrieben.

4.4.3 Pipeline

Der Vorgang des Renderings lässt sich vereinfacht als Datenverarbeitung von nacheinander folgenden Stufen darstellen.

Vertex Spezifikation In diesem Schritt werden wird ein Stream von Vertices für OpenGL vorbereitet. Dazu muss festgelegt werden, um was für ein Grundobjekt(Primitive) die Daten darstellen. Beispiele sind Punkte, Linien oder Dreiecke.

Vertex Shader Dieser Shader erhält einzelne Elemente aus dem Vertexstream und gibt dann ein einzelnes Vertex aus. Der Shader wird vom Benutzer Programmiert. Typischer Weise wird hier die Projektionsmatrix auf die Punkte angewendet.

Tessellation In dieser optionalen Stufe werden Patch(ein Primitve für Tesselation) in mehrere kleinere Primitives zerlegt.

Geometry Shader Der optionale Geometry Shader erhält als Eingabe eine Primitive und gibt keine oder mehr Primitives zurück.

Vertex Post-Processing In diesem Schritt werden Teile außerhalb des Kamerafensters verworfen, dieser Prozess nennt sich Clipping. In dieser Stufe werden werden die dreidimensionalen Koordinaten zu zweidimensionalen Kamerakoordinaten umgerechnet.

Primitive Assembly Diese Stufe erstellt teilt Primitves zu einer Sammlung von finalen kleinsten Primitves. Zum Bespiel wird aus einer Liste von Vertices vom Primitive Typ GL_LINE_STRIP mit 8 Mitglieder 7 neue Primitives vom Typ „line base“.

Rasterization In diesem Schritt werden die Primitives in diskrete Elemente unterteilt(gerastert). Diese nennt man Fragmente.

Fragment Shader Die Fragmente aus der Raster bildet den Input dieses Shaders. Die Ausgabe ist ein Farb-, Tiefen und sogenannter Stencilwert.

Per-Sample Operations Anschließend kann für den Output vom Fragment Shader eine Anzahl von Test ausführen. Beispielsweise der Tiefen Test(Depth Test) um zu vermeiden das Objekte welche verdeckt sind gezeichnet werden. Andere Test sind Scissor Test, Stencil Test und der Pixel Ownership Test.

4.5 NanoHTTPD

NanoHTTPD[9] 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.

4.6 la4j

La4j[10] ist eine aus einem Studentenprojekt entstandene offene Java Bibliothek. Die Bibliothek liefert Objekte für Lineare Algebra(Matrizen und Vektoren) und Algorithmen.

4.7 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.

4.8 Volley

Bei Volley [11] 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 [12] entwickelt.

[9] `https://github.com/NanoHttpd/nanohttpd`
[10] `http://la4j.org/`
[11] `http://developer.android.com/training/volley/index.html`
[12] `https://android.googlesource.com/platform/frameworks/volley`

5 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. Dabei unterscheiden wir zwischen 2 Dimension. Zum einem die Sichtbarkeit und zum anderen die Detailstufe der Punkte. Zum ermitteln der Punkte nach diesen Kriterien kommt ein Multi-Resolution Octree(siehe Section 2.2.3) zum Einsatz. Dadurch wird die GPU effizient eingesetzt.

Die Defizite beim Speicher werden durch eine Client-Server Architektur ausgeglichen (siehe Abb. 5.1). 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 „Least recently used“ Prinzip gespeichert. Zum Rendern werden die Punkte als „vertex buffer object“ in den Speicher der GPU geschrieben und anschließend gezeichnet. Ein Vorteil der Client Sever Architektur ist das mehrere

Als Programmiersprache wurde Java gewählt aufgrund des guten Kompromisses aus Portabilität und Performance sowie der guten Integration in das Android Betriebssystem.

5.1 Server

Der Server hat drei Aufgaben. Zum ersten erstellt er aus gegebenen Punkten einen MRT. Seine zweite Aufgabe ist es Punktanfragen zu bedienen und als letztes ein Proxy Objekt (siehe Kapitel 2.4) des MRT an die Klienten zu verteilen.

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

5.1.1 Multi Resolution Tree

Der MultiResolution Tree besteht aus 4 Klassen (siehe Abb. 5.2). 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 Kapitel2.2.3. Jeder Knoten besitzt eine ID. Diese sind die Koordinaten seines absoluten Mittelpunkts. Erwähnenswert sind die Entscheidungen bei der Raster Klasse. Die Rasterwerte sind in einer interne DefaultHashMap vermerkt. Diese bildet einen dreidimensionalen Vektor auf

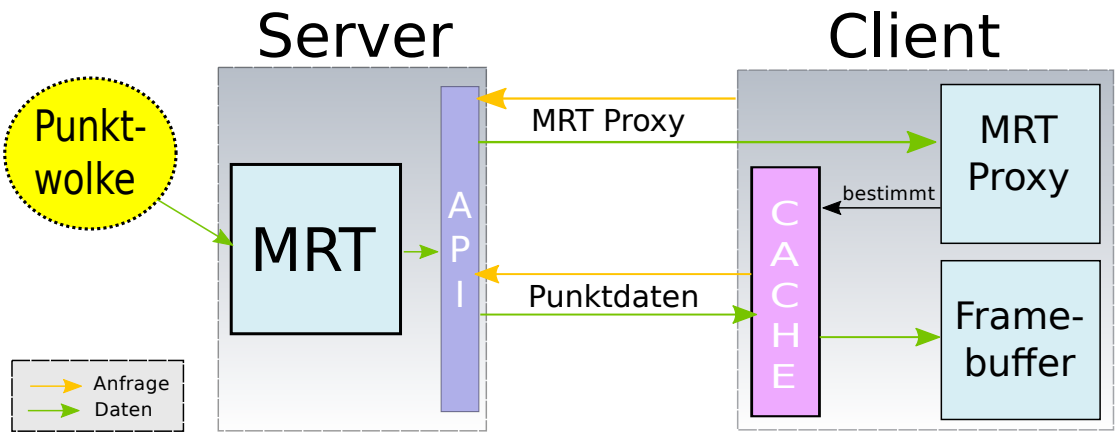

Abbildung 5.1: Übersicht

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. Der 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 weitere Hashmap als Index. Dieser bildet Ids auf die entsprechenden Knoten ab. Der Index wird in nach einer festen Anzahl Einfüge Operationen aktualisiert.

5.1.2 RESTful API

Unter einer RESTful API versteht man ein Webinterface welches die Beschränkungen von REST (siehe Kapitel 2.6) einhält.

Die API wurde mit Hilfe des NanoHTTPD Frameworks (siehe Kapitel 4.5) implementiert. NanoHTTPD wurde ausgewählt weil es schlank und einfach ist.

Ressourcen werden über GET Anfragen mit folgender Form abgerufen

```
1 GET SERVER_IP:PORT/?parameter
```

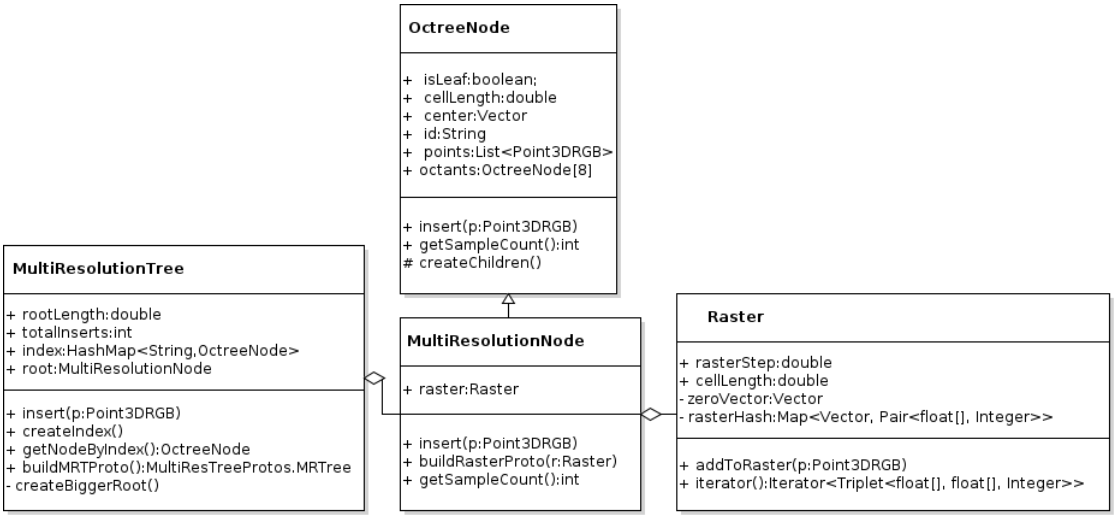

Abbildung 5.2: Multiresolution Tree UML

Parameter	Werte	Erklärung
mode	„proxy“, „samples“	Definiert den Typ der Anfrage
id	String	Id für Punktdaten

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

5.1.2.1 MRT-Proxy Anfrage

Da die Ermittlung gebrauchten Punkte auf dem Klienten statt findet muss auch dieser in Kenntnis über die Struktur des MRTs sein. Aus diesem Grund Stellt der Server ein MRT-Proxy zur Verfügung. Der MRT-Proxy ist im Prinzip gleich dem Multiresolution Octree, allerdings haben seine Knoten keine Punkte oder 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.

Eine Anfrage wäre zum Beispiel:

```
1 GET 192.168.2.1:8080/?mode=tree
```

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  }
```

5.1.2.2 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 auf 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;
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.

5.2 Client

Bei dem Client handelt es um eine Android Anwendung. Der Client 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.

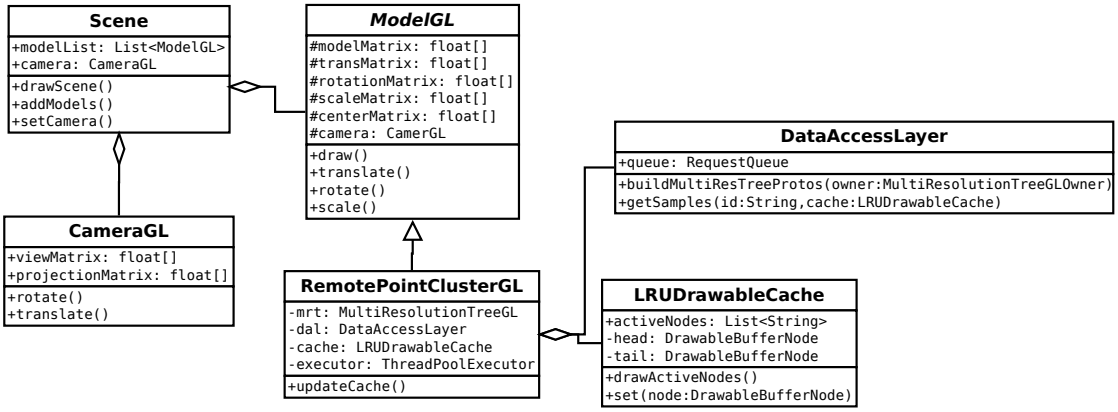

Abbildung 5.3: Client UML

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.

5.2.1 Netzwerkverkehr

5.2.1.1 DataAcessLayer Klasse

Bei der DataAcessLayer Klasse handelt es sich um ein Singleton(siehe Kapitel 2.5)welches als Schnittstelle für jeglichen Netzwerkverkehr fungiert. Sie verfügt über Funktion zum Anfordern von Punkten oder des MRT Proxy. Für asynchronen Anfragen kommt das „Volley Framework“ zum Einsatz. (siehe Kapitel 4.8). Das Framework stellt eine Prioritätswarteschlange für HTTP Anfragen zur Verfügung. Volley ermöglicht es die Callback Funktion, welche nach Eintreffen einer Antwort aufgerufen wird, zu überschreiben, um eigene Logik zu definieren.

Die zwei Methoden der Klasse werden im folgendem vorgestellt.

buildMultiResTreeProtos(owner:MultiResTreeOwner) Die Methode dekoriert eine Instanz welche das MultiResTreeOwner Interface implementiert (z.B RemotePointClusterGL) mit dem MRT-Proxy.

Als erster Schritt wird eine HTTP-Anfrage für den MRT Proxy an den Server gesendet. Beim Empfangen der Antwort werden die Daten deserialisiert und im Anschluss mit Hilfe einer Factory Methode zum MRT-Proxy gebaut. Die als Parameter übergebene Instanz erhält einen Zeiger auf den Proxy.

getSamples(id:String, cache:LRUDrawableCache) Die Methode fordert Punktdaten vom Server an und speichert diese in einem Cache Objekt welches später zum Zeichnen der Punkte verwendet wird. Bei erhalten der Antwort werden die Daten zuerst dekomprimiert(DEFLATE) und deserialisiert. Im Anschluss werden die Farb-, Positions- und

Gewichtswerte in einen nativen Buffer geschrieben damit OpenGL bei Bedarf darauf zugreifen kann.

5.2.2 Rendering

5.2.2.1 Shader

Die Shader sind als Textdatei in dem Android Ressourcen gespeichert. Bei der Initialisierung der OpenGL View Instanz werden sie kompiliert und gelinkt. Im folgendem werden die Shader vorgestellt.

```
1  # vertex_shader.glsl
2  attribute vec3 a_Position;
3  attribute vec3 a_Color;
4  attribute float a_Size;
5  uniform mat4 u_Matrix;
6  varying vec4 v_Color;
7
8  void main() {
9    v_Color = vec4(a_Color, 1.0);
10   gl_Position = u_Matrix * (vec4(a_Position, 1.0));
11   gl_PointSize = min(4.0, sqrt(a_Size));
12 }
```

In dem Vertex Shader werden die Postionen der Punkte mit der Projektionsmatrix multipliziert und an die nächste Stufe der OpenGL Pipeline weiter gegeben. Des weiteren wird die Größe der zu zeichnenden Punktes berechnet. Es wird die Wurzel gezogen um sehr große Werte abzuschwächen. Durch die min() Funktion wird eine obere Grenze von 4 eingeführt. Das ist nötig den bei sehr detaillierten Objekten kann das Gewicht schnell sehr groß werden. Die Farbwerte werden einfach an den Fragment Shader durch eine varying Variable weitergereicht.

```
1  precision mediump float;
2  varying vec4 v_Color;
3  void main() {
4    gl_FragColor = v_Color;
5  }
```

Der Fragment Shader empfängt die Farbwerte und gibt sie weiter an die vorgegebene Variable gl_FragColor und legt damit den Farbwert des Fragmentes fest.

5.2.2.2 Scene Klasse

Die Scene Klasse siehe(Abbildung 5.3)ist die Schnittstelle für alle Rendering relevanten Aktionen. Alle zu zeichnenden Objekte sowie die Kamera sind in dieser Klasse gespeichert. Nutzereingaben werden von dieser Klasse entgegengenommen und entsprechend verarbeitet.

drawScene() Beim Aufruf der Methode wird die draw() Methode von jedem Element der Liste ausgelöst.

5.2.2.3 CameraGL Klasse

-

5.2.2.4 RemotePointClusterGL

Die RemotePointClusterGL kümmert sich um das Ermitteln aktiver Punkte mit Hilfe des MRT, dafür sind Verweise auf den Cache und dem MRT-Proxy gespeichert. Des weiteren greift die Klasse auf die DataAccesLayer Instanz zu, um entweder den MRT Proxy zu aktualisieren oder neuen Punkten anzufordern. Bei Initialisierung der Instanz wird der Proxy vom Server erfragt. Der gespeicherte LRU-Cache beinhaltet die Punktdaten

Immer wenn sich die Kameraposition ändert wird die updateCache() aufgerufen um den aktiven Knoten zu ermitteln.

updateCache() Die Methode bestimmt die momentan aktiven Punkten bzw. Knoten mit Hilfe des Proxy Objektes.

Knoten(mit ihren Punkten) werden zu den aktiven Knoten hinzugefügt wenn folgende Kriterien erfüllt sind:

- die Punkte sind sichtbar
- die Auflösung der Punkte ist ausreichen oder schon Maximal

Um das zu erreichen wird der Proxy mit Hilfe des folgenden Algorithmus traversiert.

```
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 }
```

Einfach gesagt, besuche Knoten solange sichtbar bis entweder ein Blatt erreicht ist oder eine ausreichende Detailstufe. Bei dem Test auf Sichtbarkeit wird geprüft ob die projektierte Box (von einem Knoten) sich mit der View-Ebene schneidet. In jedem Schritt wird der detailFactor() ermitteln und mit einem festgelegten Schwellwert verglichen. Der

Schwellwert ist experimentell ermittelt und kann abhängig von der Leistung des Gerätes gewählt werden. Sobald der detailFactor klein genug ist, gilt der Knoten als ausreichend aufgelöst und wird gezeichnet. Der detaiFactor()berechnet sich wie folgt.

```
1 public float getDetailFactor(ModelGl m){
2   float[] boundingBox = getBoundingBox(projectPoints(edgePoints,m.
        camera.projectionMatrix, m.camera.viewMatrix, m.getModelMatrix())
        );
3   float[] centerProj = projectPoint(center, m.camera.projectionMatrix,
        m.camera.viewMatrix, m.getModelMatrix());
4   float zNorm = centerProj[2];
5   return getArea(boundingBox) * 1/(zNorm * zNorm);
6 }
```

Der komplette Vorgang findet in einem 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 werde diese vom Server angefordert.

draw() Zum zeichnen des Models wird die draw Methode aller aktiven Knoten im Cache aufgerufen.

5.2.2.5 Drawable-Cache Klasse

Diese Klasse speichert und cached die Punktdaten vom Server. Des weiteren sendet sie Punktdaten als Vertex Buffer Object(VBO) an die GPU. Der Vorteil von VBOs liegt darin, dass Punktdaten nicht bei jedem Frame neu übermittelt werden müssen. Was die Performance stark verbessert. Schlussendlich wird in dieser Klasse das Zeichnen durch OpenGL initiiert.

Der Cache besitzt eine Prioritätswarteschlange von Knoten nach dem zuletzt benutzt Prinzip. Die Knoten besitzen die Punktdaten und einen Vermerk ob diese schon im Speicher der GPU gelandet sind. Die Positions-, Farb- und Gewichtsdaten werden in einem einzigen Buffer pro Knoten ineinander abgespeichert um die Performance zu verbessern [1].

Beim zeichnen wird zuerst geprüft ob die Daten des Knoten schon als VBO auf der GPU sind. Falls ja werden die Punkte mit der folgenden Methode gezeichnet.

```
1     public void draw(){
2         glDrawArrays(GL_POINTS, 0, pointCount);
3     }
```

5.2.3 User Interface

[1] https://developer.apple.com/library/ios/documentation/3DDrawing/Conceptual/OpenGLES_ProgrammingGuide/TechniquesforWorkingwithVertexData/TechniquesforWorkingwithVertexData.html

6 Experimente und Auswertung

-

7 Zusammenfassung und Ausblick

-

Abbildungsverzeichnis

Literaturverzeichnis

[1] Marc Alexa, Johannes Behr, Daniel Cohen-Or, Shachar Fleishman, David Levin, and Claudio T. Silva. Point set surfaces. In *Proceedings of the Conference on Visualization '01*, VIS '01, pages 21–28, Washington, DC, USA, 2001. IEEE Computer Society.

[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] James Gosling and Henry McGilton. The java language environment. *Sun Microsystems Computer Company*, 2550, 1995.

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

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

[9] David Levin. The approximation power of moving least-squares. *Mathematics of Computation of the American Mathematical Society*, 67(224):1517–1531, 1998.

[10] Donald Meagher. Geometric modeling using octree encoding. *Computer graphics and image processing*, 19(2):129–147, 1982.

[11] 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.

[12] 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.

[13] 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.

[14] 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.