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:

13. April 2016

Inhaltsverzeichnis

1 Einführung

1.1 Motivation

Das Archaeocopter Projekt, welches 2012 von Dr. Benjamin Ducke und Prof. Dr. Marco Block-Berlitz ins Leben gerufen wurde, suchte nach einer Möglichkeit zur live Darstellung von dreidimensionalen Punkten während der Flüge des Archaeocopters.

Die Idee ist mit Hilfe der Darstellung feststellen zu können, für welche Bereiche des Objektes noch weitere Information gesammelt werden müssen um eine ausreichende dreidimensionale Darstellung zu erhalten.

Dadurch kann während des Fluges ermittelt werden, welche Bereiche schwer zu erkennen sind.

Die Darstellung sollte für ein Android Tablet implementiert werden.

Die Herausforderung bestand darin trotz der Limitierungen eines Tablets (wenig RAM, 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.

1.2 Ziele

Hauptziel der Arbeit war es eine Datenstruktur zu implementieren welche es ermöglicht Punktwolken im 7 stelligen Bereich auf Android Systemen flüssig darzustellen. Die Punktwolke wird während des Fluges der Drohne berechnet und erweitert, daher muss die Datenstruktur dynamisch sein. Um mit der Anzahl an Punkten zurechtzukommen sollen Punkte abhängig von der Kameraposition geliefert werden. Also bei Sichtbarkeit und in einer sinnvollen Detailstufe.

Das System folgt der Client-Server Architektur. Daher muss eine effiziente Lösung für den Netzwerkverkehr gefunden werden.

Ein weiteres Ziel der Arbeit war 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 App für iOS oder ein anderes mobiles Betriebssystem nachzureichen. Des weiteren soll eine moderne Auswahl von Frameworks/Komponenten getroffen werden.

Die App sollte möglichst intuitiv benutzbar sein.

2 Vergleichbare 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.

2.1 KiwiViewer

KiwiViewer ist eine freie 'open-source' App zum erkunden von Punktwolken.

Die App ist fuer Android sowie iOS verfügbar. Multi-Touch Gestensteuerung wird unterstützt.

KiwiViewer [1] bedient sich der "Point Cloud Library" [3] 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 QSplat Verfahren

Erste Vorschläge zur Darstellung von sehr großer Punktmengen wurden durch das QSplat [6] 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 [7] .

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

2.3 Multiresolution Octree

Moderne Ansätze benutzen einen Multiresolution Octree [8]. 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 Geraten wie Netzwerkverkehr sind nicht berücksichtigt.

2.4 Knn-Tree iOS

Eine weitere Möglichkeit ist ein Knn-Tree zu verwenden [5]. 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.

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.

3 Grundlagen und Frameworks

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

- wip design patterns

3.1 Representational state transfer(REST)

wip

3.2 Google Protocol Buffers

Dabei 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.

3.3 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. [2]

3.4 NanoHTTPD

NanoHTTPD ist ein schlanker HTTP Server. Da wir nur eine simple REST Api implementieren wollen ist Nano eine besser Wahl als zum Beispiel das überladene Apache Tomcat.

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

3.5 la4j

La4j ist eine offene Bibliothek für Lineare Algebra. Die Vektor Klasse der Bibliothek wird von dem Server verwendet.

3.6 DEFLATE

wip

- günstige Kompression

3.7 Volley

wip

- asynchrone http Anfragen für android

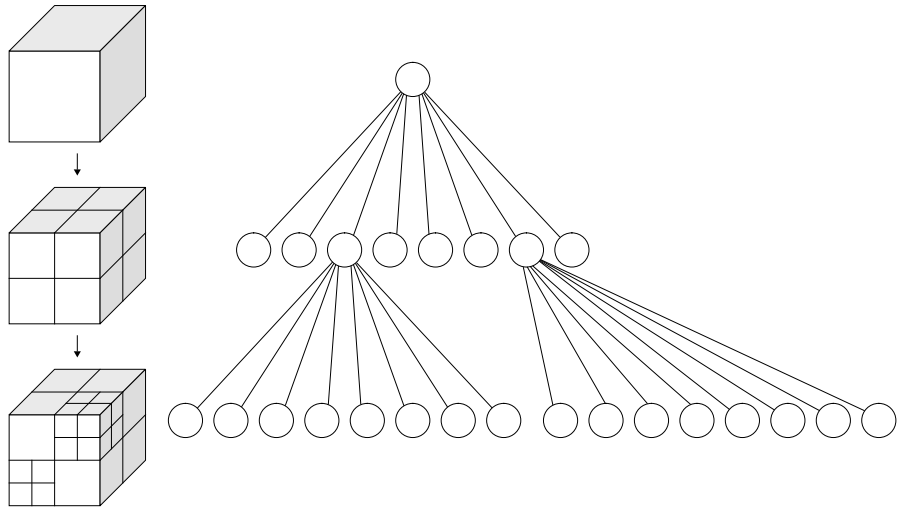

Abbildung 1: Schematische Darstellung eines Octrees

4 Gewählter Lösungsansatz [WIP]

4.1 Datenstrukturen

4.1.1 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.

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

Der dynamische Octree unterstützt das inkrementelle einfügen von Punkten.

4.1.2 Multiresolution Octree(MRT)

Die Datenstruktur entstammt aus der Arbeit “Interactive Editing of Large Point Clouds” [8]

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

Im folgendem wird die Datenstruktur vorgestellt.

Der Multiresolution Octree 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. Die Hashfunktion lautet

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

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

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

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 2 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 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.

4.2 Architektur

Die Applikation folgt der Client-Server Architektur.

4.2.1 Server

Der Http-Server bietet ein einfaches REST-Interface für den Klienten an. Allgemein hat er zwei Aufgaben. Einmal den MRT Proxy(siehe REST API) mit dem Klienten zu synchronisieren und zum anderen angefragte Punktmengen bereitzustellen.

REST API Die API wurde mit Hilfe des NanoHTTPD Frameworks implementiert.

Prinzipiell wird bei jeder Anfrage eine Fallunterscheidung an dem "mode" Parameter gemacht. Es gibt 2 Typen von Anfragen. Einmal kann nach dem MRT-Proxy gefragt werden.

Der MRT-Proxy ist im Prinzip ein Multuresolution Octree allerdings haben seine Knoten keine Punkte gespeichert, sondern Ids um beim Server die entsprechenden Punkte anzufragen. Diese können entweder Original Punkte sein oder Punkte aus der Rasterung.

In diesem Fall wird ein Protocol Buffer Objekt von dem MRT erstellt. Die Spezifikation lautet wie folgt:

```
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 }
```

Der 2. Typ Anfragen liefert Punkte an den Klienten aus. Bei dieser Anfrage wird vom Klienten immer ein Id mitgesendet.

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).

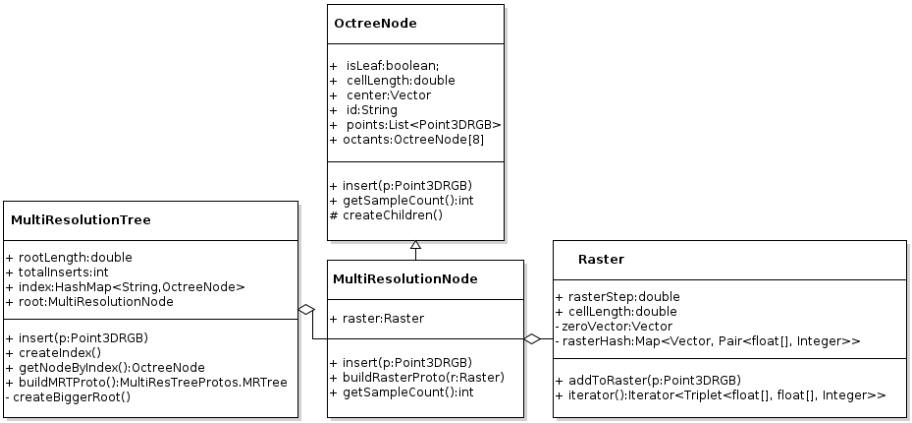

Abbildung 2: Multiresolution Tree UML

Alle Daten werden bevor sie über das Netzwerk versendet werden, durch den DEFLATE Algorithmus komprimiert um den Netzwerkverkehr möglichst niedrig zu halten.

MultiResolution Tree(WIP) Der MultiResolution Tree besteht aus 4 Klassen. Die Klasse "MultiResolutionTree" dient als Schnittstelle für alle äußeren Komponenten. 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.

Der Index wird in regelmäßigen Abständen aktualisiert.

4.2.2 Client(WIP)

- Programmiersprache Java
- 'Volley'[4] for asynchrone http Anfragen
- OpenGl ES für das Rendering
- UI

5 Schwierigkeiten

-

6 Auswertung

-

Literatur

[1] Kiwi viewer. `http://www.kiwiviewer.org/`, note = Accessed: 2014-12-03.

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

[3] Point cloud library. `http://www.pointclouds.org/`. Accessed: 2014-12-03.

[4] Volley. `https://github.com/mcxiaoke/android-volley`. Accessed: 2014-12-03.

[5] 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.

[6] 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.

[7] 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.

[8] 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.