Darstellung gigantischer Punktwolken auf Android Geräten

Jakob Krause

April 11, 2016

1 Einführung

1.1 Motivation

Das Archaeocopter Projekt, welches 2012 von Dr. Benjamin Ducke and Prof. Dr. Marco Block-Berlitz ins Leben gerufen wurde, suchte nach einer Möglichkeit zur live Darstellung von 3 dimensionalen Punkten währen der Flüge.

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

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

Die Darstellung sollte auf einem Android Tabelett implementiert werden.

Die Herrausforderung bestand darin trotz der Limitierungen eines Tablets (wenig RAM, schwache GPU) Punktwolken mit mehreren Millionen Punkten flüssig Darzustellen. Dabei muss 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 soll möglichst einfach sein eine App für iOS oder ein anderes mobiles Betriebssystem nachzureichen. Desweiteren soll eine progessive Auswahl von benutzten Frameworks/Komponenten getroffen werden.

Die App sollte moeglichst intuitiv benutzbar sein.

2 Vergleichbare Arbeiten

Das Interesse an der Darstellung von großen Punktwolken ist durch das Aufkommen von modernen 3D Scanner in den letzten Jahrzehnt stark gewachsen. Durch den Anstieg der Leistungsfähigkeit von Tabletts wurde schon einige Ansaetze fuer 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" [2] fuer seine Kernfunktionen. Die Punktmenge wird entweder über eine SD Karte, eMail oder URL geladen.

Abgrenzung Die App speichert die geladenen Punkte direkt auf dem Tablett. Daher sind die Modelle auch auf dessen Arbeitspeicher beschränkt. Modelle mit mehreren Millionen Punkten treffen schnell and die Grenzen des RAMs.

2.2 QSplat Verfahren

Erste Vorschläge zur Darstellung von sehr großer Punktmengen wurden durch das QSplat[5] 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 [6] .

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

2.3 Multiresolution Octree

Moderne Ansätze benutzen einen Multiresolution Octree[7]. Diese Datenstruktur speichert die Punkte in ihren äusseren Knoten. Die inneren Knoten beinhalten gröbere Präsentationen für ihre direkten Kinder.

Die Punktdaten werden 'out-of-core' verwaltet, also dynamisch von der Festplatte geladen. Der Octree 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[4]. 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 Clienten übertragen.

Ubertragende 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.

Die Benchmarks aus dieser Implementierung sollen als Vergleich dienen.

Abgrenzung Die Datenstruktur ist nicht dynamisch, also die Menge der Punkte muss von Anfang an fest stehen. Implementierung erfolgte in C++ fur iOS. Netzwerkverkehr erfolgt uber Http Pipelining.

3 Gewählter Lösungsansatz [WIP]

3.1 Datenstrukturen

3.1.1 Dynamic Octree

Octrees sind eine Datenstruktur um 3d dimensionale Daten hierarchisch zu untergliedern. Sie wurde 1980 von Donal Maegher beschrieben. Octrees sind im 3 dimensionalen das was im 1 dimensionalen Binarytrees sind oder im 2 dimensionalen die Quadtrees.

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 Kindern gespeichert

Der dynamische Octree unterstützt das Einfügen von Punkten.

3.1.2 Multiresolution Octree

Die Datenstruktur entstammt aus der Arbeit "Interactive Editing of Large Point Clouds" [7]

Die Datenstuktur unterstützt einfügen und löschen und bietet unterschiedlich detalierte Darstellungen des Ausgangsmodels.

Im folgendem wird die Datenstruktur vorgestellt.

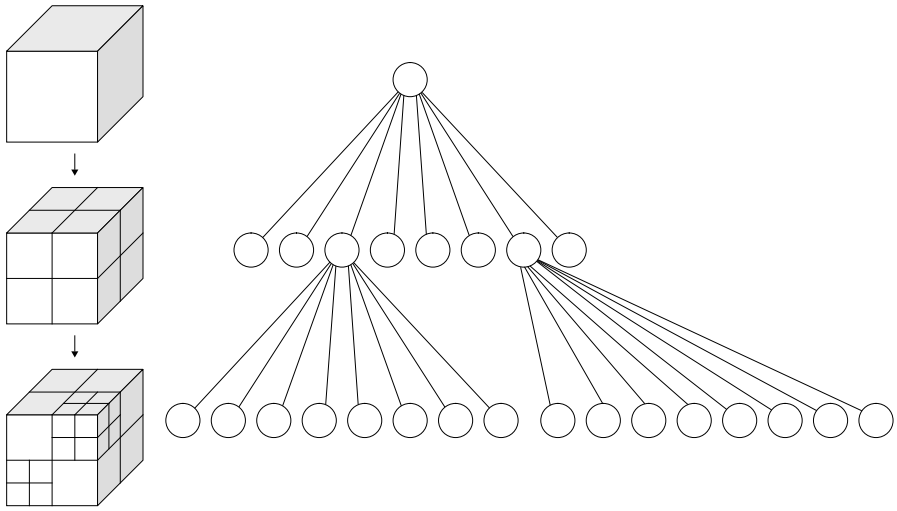

Figure 1: Schematische Darstellung eines Octrees

Der Multiresolution Octree kann als eine spezielle Form des Octree verstanden werden. Wie beim Octree enthalten die Kinder alle Punkte.

Die Tiefe ergibt sich aus der Eigenschaft das kein Kind 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 neuen Kinder verteilt.

Die inneren Knoten sollen eine vereinfachtere Version ihrer Kinder liefern.

Dafür haben inneren Knoten eine 3 dimensionale 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. Zu beginn sind alles Rasterzellen mit Gewicht 0 initialisiert. 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 Zellen fällt wird das Gewicht incrementiert.

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 auftretten.

1. Fall : Der Punkt liegt ausserhalb der Wurzel. Nun muss die bestehende Wurzel so lange erweiter 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 in Kraft

2. Fall : Der Punkt liegt innerhalb der Wurzel Zuerst wird der Punkt der Rasterung hinzugefügt. Sprich das Gewicht in der entsprechende Rasterzelle wird incrementiert und die Farbe des Punkten wird gegebenfalls gespeichert. Dann wird ermittelt in welchem der Kinder der Punkt liegt. Nun wird der Vorgang beim Kind wiederhollt 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.

3.2 Architektur

Die Applikation besteht entspricht einer Client-Server Architektur.

Client und Server sind beide in Java geschrieben.

3.2.1 Server

Der Server verwaltet den Multiresolution Tree. Der http Server stellt dem Client die Datenstruktur zur Verfügung um benötigte Punkte zu ermitteln.

Die Knoten des Baumes enthalten lediglich Keys zu den Daten.

Als HTTP Server kommt “nanohttpd” zum Einsatz, weil es schlank und einfach ist.

Die eigentlichen Punkte werden in einer Key-Value Datenbank gespeichert.

Das Model selbst ist auf die Blätter verteilt. Die inneren Knoten erhalten vereinfachte Abbilder von seinen Kindern.

- Programmiersprache Java
- dynamischer ’Multi Resolution’ Octree
- Google Protocol Buffers

3.2.2 Client

Der Android-Client erhält ein Abbild der Datenstruktur vom Server und synchronisiert diese regelmaessig. Beim traversieren der Struktur werden an den Server Anfragen gesendet.

Empfangene Daten werden in einem ’last recently used’ Cache gespeichert. Dieser Cache wird gleichzeitig von OpenGl als Buffer für die Punkte benutzt.

Es existiert ein Buffer für die Punktposition, die RGB Werte und das Gewicht.

Die Anfragen zum Server laufen über das http Protokoll. Für Anfragen an den Server wird das Volley Framework verwendet. Anfragen werden in einer Warteschlange gespeichert und gesendet. Antworten werden in eigenen Threads verarbeitet.

- Programmiersprache Java
- 'Volley'[3] for asynchrone http Anfragen
- OpenGl ES für das Rendering
- UI

3.2.3 Netzwerktransfer

Bestehende Ansätze machen sich eine Kompression zu nutze um den Netzwerkverkehr möglichst kompakt zu halten.

Unsere Lösung bedient sich vorerst nur einer Serializierung in Form von "Protocol Buffers". Dabei handelt es sich um eine platform 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.

Spezifizierung der Rasterung

```
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 }
```

Als Übertragungsprotkoll kommt HTTP zum Einsatz.

Schwierigkeiten

Benchmarks

References

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

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

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

[4] 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.

[5] 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.

[6] 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.

[7] 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.