Bringing Your Hands Into Virtual Reality

Schlussbericht

Studiengang:	Informatik
Autoren:	Simon Meer
Betreuer:	Prof. Urs Künzler
Experten:	Yves Petitpierre
Datum:	08.06.2015

Versionen

Version	Datum	Status	Bemerkungen
0.1	14.04.2015	Entwurf	Outline erstellt
0.9	08.06.2015	Entwurf	Draft zum Durchlesen erstellt

Inhaltsverzeichnis

1 Einleitung

Eines der faszinierendsten Entwicklungen unserer Zeit ist zweifelsohne das Aufkommen von Virtual Reality (VR) zu einem erschwinglichen Preis. Passend zu dieser Entwicklung finden auch zunehmend neue Peripherie-Geräte ihren Weg in den Markt. Mitunter zu den offensichtlich natürlichsten Methoden des Inputs gehören ganz klar die eigenen zwei Hände. Es scheint deshalb logisch, diese als Peripherie zu verwenden.

In dieser Arbeit geht es darum, ein Head-Mounted Display (HMD), die Oculus Rift, mit einem Hand-Sensor, der Leap Motion, zu koppeln, und eine funktionierende Applikation damit zu entwickeln.

2 Aufgabenstellung und Zielformulierung

Damit klar wird, was genau die Aufgabe ist, und was zur Durchführung verwendet wird, soll dieses Kapitel ein paar Fakten liefern.

2.1 Aufgabenstellung

Es soll eine Applikation namens "IMVR" entwickelt werden, welche Gebrauch von der Oculus Rift macht, um die Bilder- und Musiksammlung des Anwenders ansprechend darzustellen, z.B. in Form eines 3D-Karussels. Die zusätzliche "Tiefe", die durch den Einsatz eines stereoskopischen HMD entsteht, soll dem Anwender helfen, sich in seiner Medienbibliothek schneller zurechtzufinden.

Zusätzlich dazu soll die Leap Motion dazu verwendet werden, um vollständige Handfreiheit zu gewähren: Der Anwender soll komplett ohne Maus und Tastatur imstande sein, sich durch seine Bilder zu navigieren.

Kurz zusammengefasst muss die Applikation:

- Die Bild- und Musikbibliothek des Benutzers in stereoskopischem 3D darstellen.
- Diese freihändig durchsuchbar machen mit Sortier- und evtl. Gruppierfunktion.
- Die Bilder betrachtbar und die Musik abspielbar machen.
- Metainformationen darstellen (z.B. in Form von Diagrammen).

Zusätzlich zur Applikation selbst soll noch ein zusätzliches Tool entwickelt werden, welches im Voraus die Dateien auf dem Host-System indexiert und für die visuelle Applikation bereitstellt.

2.2 Hilfsmittel & Hardware

Verschieden Hilfsmittel werden für die Durchführung des Projektes gebraucht. Seitens Software sind diese:

Name	Beschreibung
Unity3D	Spielengine und Entwicklungsumgebung
Visual Studio 2012 & 2013	Entwicklungsumgebung von Microsoft
Blender	Open-Source 3D Modelling-Tool
Krita	Open-Source Grafikbearbeitungs-Tool
LaTeX	Hilfsmittel für die Erstellung wissenschaftlicher Dokumente

Tabelle 2.1: Übersicht der eingesetzten Software.

Im Hardware-Departement wurde folgendes eingesetzt:

Name	Beschreibung
Oculus Rift	HMD von Oculus VR
Leap Motion	Hand- und Fingererkennungsgerät von Leap Motion, Inc.

Tabelle 2.2: Übersicht der eingesetzten Hardware.

3 Design

Bei der Entwicklung wurden diverse Design-Entscheidungen gefällt, welche zum Teil bereits in einem Vorprojekt analysiert wurden. Diese sollen in diesem Kapitel aufgelistet und erläutert werden.

3.1 Systemarchitektur

Um die Applikation zu bedienen, setzt der Anwender die Oculus Rift auf und bewegt sich im von der mitgelieferten Kamera erkennbaren Bereich.

Die Leap Motion wird prinzipiell so verwendet, wie von der Herstellerfirma vorgesehen. Das heisst, diese wird mit dem offiziellen Aufsatz [1] an der Oculus Rift befestigt, und deckt so den frontalen Sichtbereich des Anwenders ab. Dies lässt sich gut auf Abbildung 3.1 erkennen.

Ebenfalls erkennbar ist, dass beide Geräte per USB mit dem Host-Computer verbunden sind und Daten an die jeweiligen Services schicken. Diese Services werden durch die in Unity verwendeten Plugins angesteuert, und die ausgewerteten Daten in IMVR verwendet.

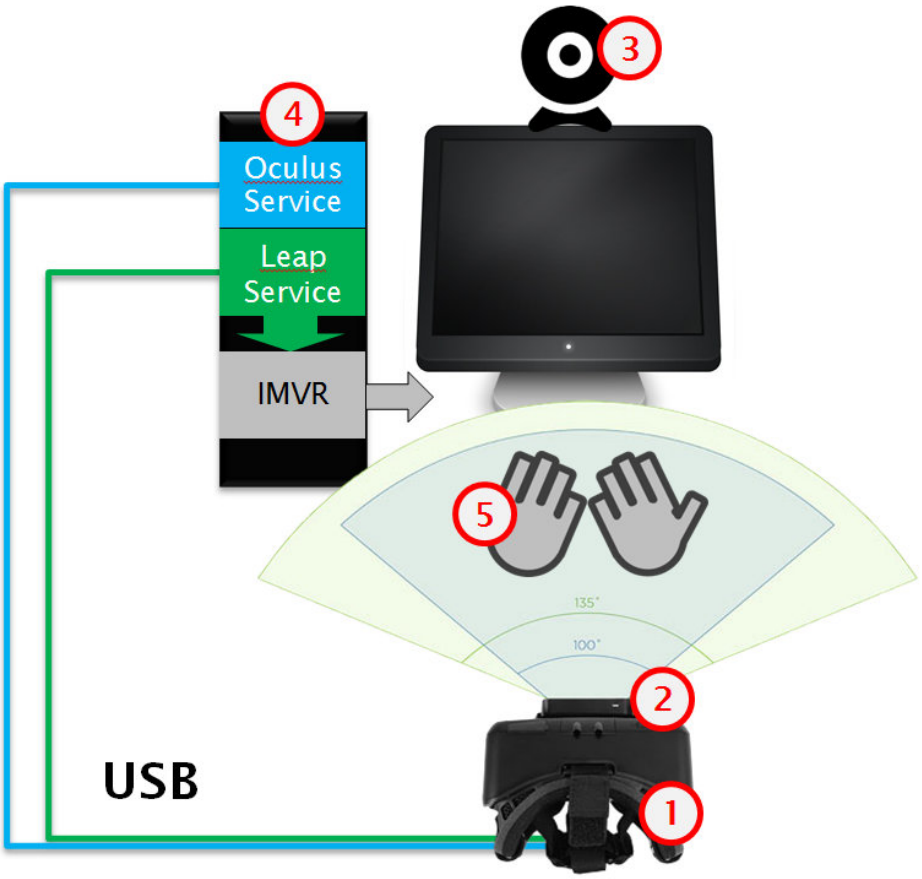

Abbildung 3.1: Eine Übersicht der Technologien und wie sie verbunden sind.

Nr.	Komponente	Beschreibung
1.	Oculus Rift DK2	HMD für den grafischen Output.
2.	Leap Motion	Gerät, welches Hände erkennt und ihre Koordinaten an den Computer sendet.
3.	Oculus Rift Kamera	Kamera, welche seit dem DK2 für das örtliche Tracking zuständig ist.
4.	Computer	Host-System für IMVR.
5.	Benutzer	Benutzer, der die Oculus Rift trägt und mit seinen Händen das Programm steuert.

3.2 Systemdesign

4 Implementation des Indexers

Ein Teil der Arbeit war es, einen Indexer zu implementieren, der die Dateien des Benutzers durchläuft und - wie es der Name vermuten lässt - interessante Dateien indexiert. In diesem Kapitel soll auf den Aufbau und die Implementationsdetails eingegangen werden.

4.1 Aufbau

Der Indexer sowie die Datenstruktur und alle anderen Tools, die eine ergänzende Funktion zur Hauptapplikation haben, wurden in einer separaten Visual Studio Solution zusammengefasst. In dieser befinden sich vier Projekte:

Name	Beschreibung
IMVR.Commons	DLL-Projekt, welches die Klassen enthält, die zwischen Applikation und Indexer geteilt werden.
IMVR.Commons.Tests	Testprojekt mit Unit-Tests um die Integrität der Daten sicherzustellen.
IMVR.Indexer	Konsolenprojekt, welches die Musikdateien auf dem Host-System indexiert.
IMVR.SpeechServer	Konsolenprojekt, welches in der Vorarbeit verwendet wurde und in dieser Arbeit keine Verwendung fand.

Tabelle 4.1: Die Projekte in Auxiliary Tools

Der Indexer (IMVR.Indexer) ist als parallelisiertes, knotenbasiertes System konzipiert worden. Die Parallelität wurde deshalb gewählt, weil es beim Einholen der verschiedenen Datenquellen teils zu Wartezeiten kommt, die gut anders genutzt werden können. Dieser Faktor kam besonders ins Spiel als noch zusätzlich zu den Musikdaten auch Bilderdaten gesammelt worden sind.

Ein weiterer Grund für die Parallelisierung ist, dass als mögliches Feature eine real-time Indexierung vorgesehen war, die es letztendlich allerdings nicht in das fertige Programm schaffte. Weitere Informationen zu diesem Thema werden in Abschnitt 4.4 gegeben.

4.2 Datenquellen

Da IMVR zum Ziel hat, die Musik des Benutzers in verschiedenen Formen und Farben darzustellen, werden Daten von diversen Quellen benötigt. Wir leben in einer wundervollen Zeit in Sachen

Datenvielfalt: Es gib viele Online-Services, welche zu einem Grossteil gratis sind, von denen man diverse Daten erhalten kann.

Es folgt eine kurze Zusammenstellung von untersuchten Datenquellen.

Name	Daten	API-Limite
Last.fm	Bilder, Meta-Daten	5 Request / Sekunde
The Echo Nest	Bilder, Meta-Daten, Analyse-Daten	120 Requests / Minute
Spotify	Bilder, Meta-Daten, Playlisten, Streams	?
Amazon	Bilder, Beschreibungen	1 Request / Sekunde
Gracenote	Bilder, Fingerprinting, Meta-Daten	∼1000 Requests / Tag
MusicBrainz	Bilder, Meta-Daten	∼1 Request / Sekunde

Tabelle 4.2: Eine Übersicht von verfügbaren Online-Datenquellen.

Im Falle von IMVR kommen die Daten grundsätzlich von drei verschiedenen Quellen:

- ID3 Tags der Musik-Dateien
- The Echo Nest
- Last.fm

Diese wurden gewählt aufgrund der durchsichtigen API-Limite und den verfügbaren Daten. The Echo Nest aggregiert zudem die Daten von anderen Seiten wie Spotify und MusicBrainz, und enthält wertvolle Analyse-Daten, die in eine zentrale Position in IMVR haben.

In einem ersten Schritt werden grundlegende Daten wie der Titel des Liedes, der Name des Artisten, usw. aus der Datei selbst entnommen. Wenn möglich wird auch gleich geprüft, ob ein Album-Cover hinterlegt ist.

In einem zweiten Schritt wird über eine .NET Bibliothek [2], welche im Rahmen des Projektes geforkt und erweitert wurde, zur API von The Echo Nest verbunden und diverse Meta-Daten zur Musik heruntergeladen.

Was bei der Originalimplementation leider fehlt, sind neuere Features wie Genres und Ortsdaten, sowie ein intelligenter Bremsalgorithmus, der dafür sorgt, dass die Datenlimite eingehalten wird. Deshalb wurde ein Fork erstellt, der diese Daten und Funktionalitäten ergänzt[1].

4.3 Datenstruktur

Für die Abspeicherung der Daten wurde zuerst ein Ansatz gewählt, der auf einer SQLite Datenbank basierte. Diese wurde in einem Prototyp auch erfolgreich implementiert. Es stellte sich jedoch heraus, dass diese Abhängigkeit das Programm unnötig verkomplizieren würde und eine simple In-Memory Datenstruktur völlig ausreicht.

[1] `https://github.com/EusthEnoptEron/echonest-sharp/tree/additional_apis`

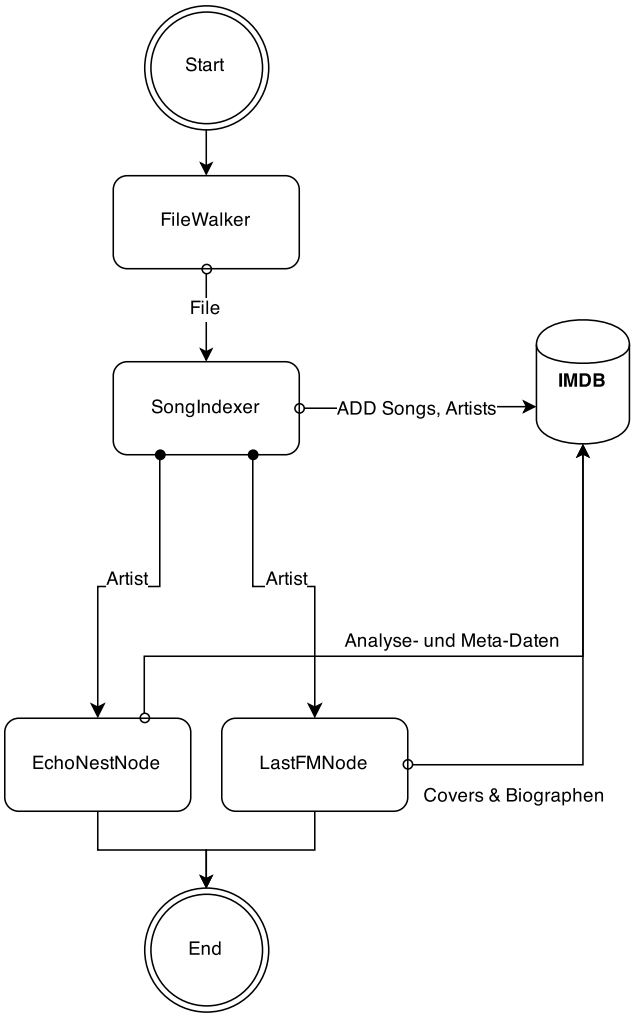

Abbildung 4.1: Der Datenfluss, den die Files beim Indexieren nehmen.

Das finale Produkt verwendet also eine simple, objektorientiert Datenstruktur, die serialisiert und so zwischen den zwei Projekten (Indexer und IMVR) geteilt werden kann. Das Schema ist in Abbildung 4.2 zu sehen. Damit beide Projekte Zugriff auf die genutzten Klassen haben, wurde das Schema in eine separate DLL ausgelagert, die ebenfalls als Abhängigkeit in beiden Projekten referenziert wird.

Anzumerken ist, dass mehrere Einstiegspunkte auf die Daten existieren und so eine gewisse Redundanz geschaffen wird. Diese Redundanz ist hilfreich, weil in IMVR mehrere Modi eben diese Einstiegspunkte benötigen. Aufgrund der gewählten Serialisierungsmethode entsteht jedoch kein grosser Speicher-Overhead.

Die Serialisierung wird durch sogenannte Protocol Buffers [4] realisiert, wofür eine Open-Source Bibliothek namens *protobuf-net* [2] existiert. Protocol Buffers ist ein binäres Datenformat, welches von Google Inc. entwickelt wurde und bekannt ist für seine Kompaktheit und Simplizität. Gewählt wurde es, weil die herkömmliche Serialisierung mit .NETs `BinaryFormatter` zum Teil zu Problemen mit Unity führen kann.

[2] `https://code.google.com/p/protobuf-net/`

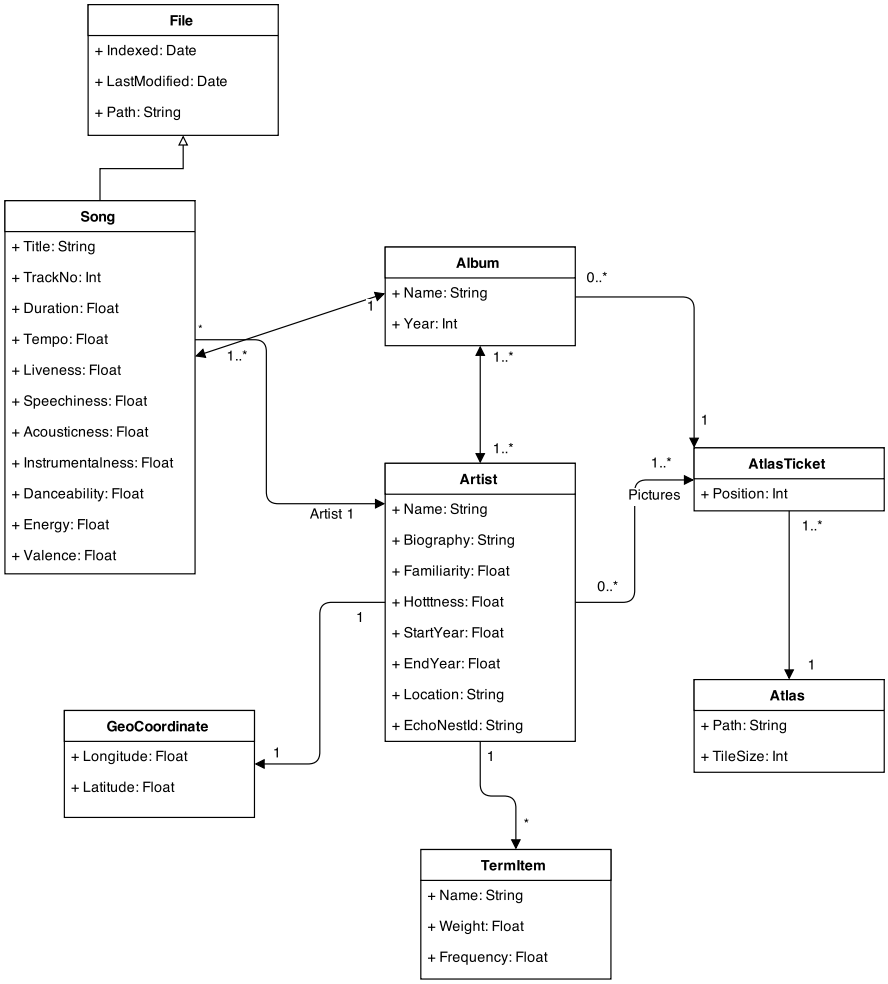

Abbildung 4.2: Klassendiagramm für die Klassen im IMVR.Commons Package.

4.4 Herausforderungen

Während der Entwicklung des Indexers kristallisierten sich diverse Schwierigkeiten, welche grundsätzlich in drei Kategorien einordnen lassen: organisatorische und datentechnische.

Die Organisation, oder Planung, war deshalb problematisch, weil einer der wichtigsten Faktoren für den Aufbau des Indexers die Vielfalt der Datentypen war. Da anfangs geplant war, Musik *und* Bilder zu indexieren und darzustellen, machte eine parallele Verarbeitung der Daten Sinn und war auch notwendig.

Es wurden parallel mehrere Bilder auf mehreren CPU-Kernen analysiert, und gleichzeitig wurde auch die Musikdatenbank erweitert. Mit dem Wegfall des Bilder-Parts wird die parallele Verarbeitung also um einiges unwichtiger. Dies gilt besonders, weil bei den Datenquellen der Musikindexierung jeweils nur ein Thread Sinn macht, da die APIs mit Limiten ausgestattet sind.

Datentechnisch stellte es sich als problematisch heraus, geeignete Datenquellen zu wählen. Von Anfang an war klar, dass der Service von The Echo Nest verwendet würde, doch dieser alleine ist nicht ausreichend.

Bevor schliesslich Last.fm als Quelle für Album-Covers gewählt wurde, wurden besonders zwei APIs untersucht: Amazon und Gracenote.

Im Falle von Amazon führten zwei Probleme zum Ausschluss: das Fehlen einer guten C#-Bibliothek und die schwerfällige Registrierung. Momentan sieht die Situation so aus, dass recht viel manuell gemacht werden muss, bzw. eine umfangreiche Service-Description importiert werden muss. [3] In Sachen Registrierung wird erwartet, dass man eine Webseite registriert, Kontaktdaten angibt und dann geprüft wird.

Bei Gracenote beläuft sich das Problem hauptsächlich auf die Daten-Limite. Diese ist nirgends öffentlich ersichtlich, und sobald diese überschritten wird, sind keine Requests mehr möglich für den ganzen Tag. In einer Applikation wie IMVR, wo in einem Schritt alles indexiert werden soll, ist die ein Killer-Kriterium.

5 Implementation von IMVR

5.1 Aufbau

5.2 Konzept

5.3 Interaktionskonzept

5.4 Visual Design

5.5 Herausforderungen

6 Testergebnisse

7 Projektmanagement

7.1 Soll-Ist

7.2 Zeitplan

8 Schlussbetrachtung

Zum Schluss soll kurz ein Blick auf die Gegenwart, die Vergangenheit und die Zukunft geworfen werden. War das Projekt erfolgreich? Wo lagen die *Pitfalls*? Wie geht es weiter? Diese Fragen sollen an dieser Stelle beantwortet werden.

8.1 Ergebnis

8.2 Reflexion

Grundsätzlich ist das Ergebnis zufriedenstellend, doch gibt es trotzdem zahlreiche Punkte, welche hätten besser gelöst werden können.

Teilweise aufgrund der experimentellen Natur des Projektes war die Planung nicht so ausgearbeitet und zuverlässig, wie sie es in einem herkömmlichen Projekt wäre. Durch das mehrfache Verändern von Funktionalität, dem wiederholten Refactoring, und der geänderten Zielsetzung entstand viel Komplexität und "Code-Smell" [5].

8.3 Ausblick

Selbständigkeitserklärung

Ich/wir bestätige/n, dass ich/wir die vorliegende Arbeit selbstständig und ohne Benutzung anderer als der im Literaturverzeichnis angegebenen Quellen und Hilfsmittel angefertigt habe/n. Sämtliche Textstellen, die nicht von mir/uns stammen, sind als Zitate gekennzeichnet und mit dem genauen Hinweis auf ihre Herkunft versehen.

Ort, Datum: Biel, 08.06.2015

Name Vorname: Meer Simon

Unterschrift: .....

Literaturverzeichnis

[1] "Leap motion - 3d motion and gesture control for windows & mac." [Online]. Available: https://www.leapmotion.com/product/vr

[2] T. Auensen, "echonest-sharp." [Online]. Available: https://github.com/torshy/echonest-sharp

[3] O. Trutner, "Signing amazon product advertising api requests ? the missing c# wcf sample," 2009. [Online]. Available: https://flyingpies.wordpress.com/2009/08/01/17/

[4] Wikipedia, "Protocol buffers," 2015, [Online; accessed 24-May-2015]. [Online]. Available: http://en.wikipedia.org/w/index.php?title=Protocol_Buffers&oldid=662187371

[5] ——, "Smell (programmierung)," 2015, [Online; Stand 1. Juni 2015]. [Online]. Available: http://de.wikipedia.org/w/index.php?title=Smell_(Programmierung)&oldid=137769083

Abbildungsverzeichnis

Tabellenverzeichnis