Hochschule für Angewandte Wissenschaften Hamburg
Hamburg University of Applied Sciences

Bachelorarbeit

Ilyuza Mingazova

Automatische Verknüpfung von Blender-3D-Animationsobjekten mit den virtuellen Schallquellen einer WFS-Anlage.

Fakultät Technik und Informatik
Studiendepartment Informatik

Faculty of Engineering and Computer Science
Department of Computer Science

Ilyuza Mingazova

Automatische Verknüpfung von Blender-3D-Animationsobjekten mit den virtuellen Schallquellen einer WFS-Anlage.

Bachelorarbeit eingereicht im Rahmen der Bachelorprüfung

im Studiengang Bachelor of Science Angewandte Informatik
am Department Informatik
der Fakultät Technik und Informatik
der Hochschule für Angewandte Wissenschaften Hamburg

Betreuender Prüfer: Prof. Dr. Wolfgang Fohl
Zweitgutachter: Prof. Dr.-Ing. Andreas Meisel

Eingereicht am: 5. Juli 2016

Ilyuza Mingazova

Thema der Arbeit
Automatische Verknüpfung von Blender-3D-Animationsobjekten mit den virtuellen Schallquellen einer WFS-Anlage.

Stichworte
Blender, COLLADA, Wellenfeldsynthese, WFS, Open Sound Control, OSC

Kurzzusammenfassung
Diese Arbeit beschreibt eine Konzeptstudie zur Umsetzung einer automatischen Verknüpfung von 3D-Grafikobjekten an die virtuellen Sound-Quellen einer Wellenfeldsynthese-Anlage. Es wurde ein Java-Programm entwickelt, welches durch das Einlesen einer Blender-Szene als COLLADA-Datei die Animationsobjektkoordinaten auf die virtuellen Quellen überträgt. Das Ansteuern der WFS-Anlage, erfolgt über das Netzwerk durch das Verschicken von OpenSound Control-Nachrichten. Im Anhang befindet sich eine Benutzereinleitung für die Arbeit mit der Software.

Ilyuza Mingazova

Title of the paper
Automatic mapping of Blender 3D animation objects to the virtual sound sources of a WFS system.

Keywords
Blender, COLLADA, wave-field-synthesis, WFS, Open Sound Control, OSC

Abstract
This thesis describes a concept study of the implementation of an automatic mapping of 3D graphic objects to the virtual sound sources of a wave field synthesis system. In this thesis, a Java Programm will be developed, which transfers the animation object coordinates to the virtual sources, from a COLLADA file of a Blender scene. The WFS system is controlled by OpenSound Control messages, that are sent via the network. A user manual for the developed Software is attached to this thesis.

Inhaltsverzeichnis

1. Einleitung **1**
- 1.1. Die Grundidee und Motivation . . . 1
- 1.2. Ziel der Arbeit . . . 3
- 1.3. Aufbau der Arbeit . . . 3

2. Grundlagen der 3D-Video- und Audio-Modellierung **5**
- 2.1. Die Grundbausteine von Blender . . . 5
  - 2.1.1. Koordinatenraum . . . 5
  - 2.1.2. Mesh . . . 6
  - 2.1.3. Armature . . . 7
  - 2.1.4. Scene . . . 8
  - 2.1.5. Container . . . 8
  - 2.1.6. Import/Export Möglichkeiten . . . 8
- 2.2. COLLADA-Format . . . 8
- 2.3. Das Wellenfeldsynthese-Labor der HAW Hamburg . . . 13

3. Anforderungsanalyse **18**
- 3.1. Zielsetzung . . . 18
- 3.2. Quelldatenanalyse . . . 18
  - 3.2.1. Verschiedene Koordinatensysteme . . . 21
- 3.3. Verbindung zu dem Sound in der Wellenfeldsynthese-Anlage . . . 22
- 3.4. Zusammenfassung der Analyse . . . 23

4. Konzept der Software **24**
- 4.1. Übersicht der Anforderungen . . . 24
- 4.2. Definition der Arbeitssequenz . . . 25
- 4.3. Programmarchitekturkonzept . . . 25
- 4.4. Ablauf in einem GUI-Kozept . . . 28

5. Implementierung der Software **32**
- 5.1. Animationsobjekte als Klassen . . . 32
- 5.2. Objektdetektor der Software . . . 36
- 5.3. Die Speicherfunktionen der Software . . . 39
- 5.4. Das package *senderFamily* . . . 42

6. Testen der Software **48**
- 6.1. Evaluierungskriterien . . . 48

6.2. Evaluierung unter diversen Betriebsystemen 51
6.3. Validierung der Software 52
6.3.1. Black-Box-Test der Software 52
6.3.2. White-Box-Test der Software 53
6.4. Zusammenfassung der Ergebnisse 53

7. Zusammenfassung und Ausblick **54**
7.1. Ausblick auf die Erweiterung der Software 55
7.2. Fazit 56

A. Benutzerhandbuch **57**

B. Klassendiagramme

1. Einleitung

Die HAW Hamburg wurde im Jahr 2011 mit einer Wellenfeldsynthese-Anlage ausgestattet (vgl. Fohl, 2013). Mit Hilfe so einer Anlage ist es möglich die realen Schallquellen durch virtuelle zu simulieren. Seit der Installation der Anlage wurden in diesem Rahmen mehrere Projekte verschiedener Anwendungsziele erfolgreich umgesetzt. Unter anderem wurden auch visuelle Medien eingesetzt. In einigen Projekten wurden Videos mit Sound-Spuren, welche über die Wellenfeldsynthese-Anlage räumlich wiedergegeben werden konnten, erstellt.

1.1. Die Grundidee und Motivation

Die Entertainment-Industrie breitet sich über die Grenzen eines zweidimensionalen Raumes aus. Dazu zählen unzählige Filme und Animation-Projekte, welche in den letzten Jahren vorgestellt worden sind. Mit jedem neuen Jahr werden die Filme realistischer. Es beginnt mit der Einführung der visuellen 3D-Sicht und erweitert sich auf den dreidimensionalen Klangteppich. 3D-Filme sind schon seit längerem zur Film- und Fernsehkultur und bieten ein großes Filmerlebnis für alle Filmfreunde. Im Audio-Bereich wurden die Höhen-Informationen eingeführt. Die Zusammensetzung von Audio und Video resultiert in einem 3D Film, der äußerst realistisch wirkt.
Die Verknüpfung von Video und Audio geschieht händisch und ist enorm zeit- und somit kostenaufwändig. Diesen Vorgang zu automatisieren würde bei 3D-Videoproduktionen den Workflow verbessern, die Produktionszeit verringern und das Positionieren der Sound-Quellen präzisieren. Diese Arbeit widmet sich den Möglichkeiten zur Automatisierung dieser Prozesse und soll im Ergebnis zu einer Lösung bzw. Lösungsansatz führen, welche in der Filmindustrie einen unglaublichen Mehrwert schafft. Dafür werden in den einzelnen Bereichen verschiedene Möglichkeiten analysiert und umgesetzt.

Objekterkennung in Videos

Bei der visuellen Darstellung unterscheidet man zwischen einem Film und einem Animationsfilm. Ein Animationsfilm besteht aus mehreren Objekten, welche entworfen werden. Jedes

Detail wird per Hand an einem bestimmten Ort platziert. Solche Objekte werden mit Hilfe einer Animationssoftware (wie z. B. Blender (vgl. Blender Foundation, c)) modelliert und animiert. Mit einer Kamera aufgenommenes Material enthält keine Objekte sondern nur eine Reihe von Bildern, welche nacheinander folgen. Aus diesem Grund muss bei einem Film eine Objekterkennung durchgeführt werden. Dazu wird häufig Deep Learning eingesetzt. Die Funktion von Deep Learning liegt darin, die Bildbereiche in einem Bild zu analysieren und mit Musterbildern aus der vorhandenen Datenbank zu vergleichen. Das Konzept wurde von Google (vgl. Google Inc., b) weiterentwickelt und im Jahr 2012 stand ein Algorithmus zur Objekterkennung in Videos bereit (vgl. Google Inc., a). Somit konnten die Nutzer nachdem Hochladen von Videos die einzelne Tags aus den vom Algorithmus vorgeschlagenen Tags auswählen.
Da die Positionsdaten der Objekte bereits vorhanden sind und nicht berechnet oder geschätzt werden müssen, begrenzt sich diese Arbeit zur Vereinfachung auf die Animationsfilme.

Objektorientiertes Surround-Audio

Für ein realistisches Filmerlebnis ist das Klangerlebnis entscheidend. Sowohl Großkino- als auch Heimkinobereich entwickeln sich weiter. Der aktuelle Stand der Technik bietet eine dreidimensionale Klangkulisse. Neben aufwändigen und teueren Wellenfeldsynthese-Anlagen, stehen Dolby Atmos (vgl. Dolby Laboratories Inc.) und Auro 3D (vgl. Auro Technologies) zur Verfügung. Auro 3D (vgl. Auro Technologies) ist wie eine Wellenfeldsynthese-Anlage kanalbasiert, wobei Dolby Atmos (vgl. Dolby Laboratories Inc.) auf dynamische Audio-Objekte mit Metadaten setzt. Das bedeutet, während der Filmwiedergabe errechnet ein AV-Verstärker, auf welchem Lautsprecher mit welchem Anteil er welches Klangobjekt wiedergeben soll, damit der Sound im Raum dreidimensional geortet wird.

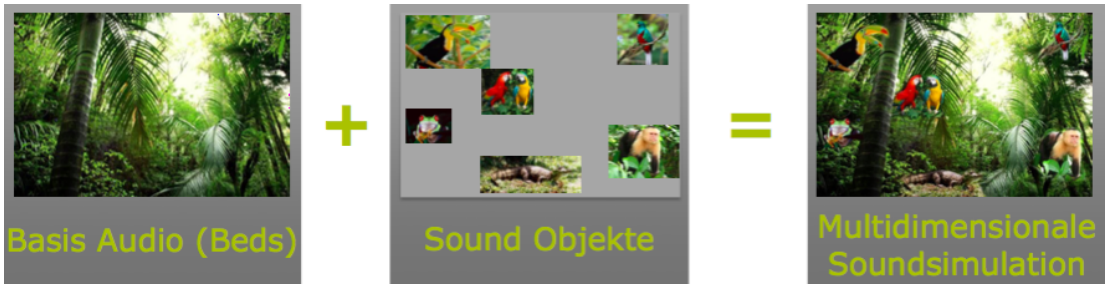


Abbildung 1.1.: Dolby Atmos: Objekt-orientiertes Audio (vgl. Paulsen, 2014)

In der Abbildung 1.1 ist ein Ansatz, für die Tonmischung und Wiedergabe von Dolby Atmos, vorgestellt. Aus der Mischung von statischen Elementen (Basis Sound) und dynamischen

Sounds (Sound Objekte), welche sich im Raum bewegen, entsteht eine multidimensionale Soundsimulation. Diese Kombination lässt den Hörer in eine real-erscheinende Szene eintauchen. Dabei müssen die Soundobjekte nur einmal gemixt werden, um später wiedergegeben werden zu können (vgl. Paulsen, 2014).
Die in der Abbildung 1.1 dargestellten Sound-Objekte sind Audioclips, welche mit Objekt-Metadaten angereichert sind. Diese Metadaten enthalten eine Nummer (ID) und Position (x, y, z), eine Grösse (Divergenz, Ausdehnung) und eine automatisierbare „Flugbahn“ (vgl. Paulsen, 2014). Durch diese Daten wird den Sound-Designern ermöglicht, Töne exakt im Zuschauerraum zu platzieren. Bei der Wiedergabe im Kino werden die Audiospuren auf die vorhandene Audiokonfiguration gerendert. Eine feste Kanalkonfiguration ist in diesem Fall für die Objekte nicht erforderlich (vgl. Watson).

1.2. Ziel der Arbeit

Die aktuelle Situation in dem Wellenfeldsynthesenanlage-Labor benötigt die manuelle Eingabe aller Quellen-Bewegungen. Das bedeutet, für einen 3D-Sound des Films muss man nach dem Erstellen des Films zuerst die Objektkoordinaten in die Wellenfeldsynthese-Koordinaten umrechnen. Danach werden diese Koordinaten per Hand in einer OpenSound Control-Software (vgl. OpenSound Control) wie SuperCollider (vgl. SuperCollider, a) eingetragen. Am Ende kann man den Vorgang der Bewegung der Quellen aus der OpenSound Control-Software (vgl. OpenSound Control) starten, wodurch die virtuellen Schallquellenpositionen in der Wellenfeldsynthese-Anlage über die Zeit gesteuert werden.
Das Ziel der Arbeit besteht in der Entwicklung einer Software zur automatischen Verknüpfung der (3D-)Grafikobjekte mit den virtuellen Schallquellen einer Wellenfeldsynthese-Anlage. Dieses Programm soll die Möglichkeit bieten, die Koordinaten eines Animationsobjekts (erstellt mit Hilfe von einer 3D-Grafiksoftware) zu extrahieren und anhand dieser, die virtuellen Sound-Quellen der Wellenfeldsynthese-Anlage automatisch zu manipulieren. Hierzu wurde eine Konzeptstudie durchgeführt, auf deren Basis die bestmöglichen Wege umgesetzt wurden.

1.3. Aufbau der Arbeit

Die Arbeit beginnt mit der Erläuterung von den Grundlagen der Video- und Audio-Modellierung. Das Kapitel vermittelt das Grundwissen der 3D-Grafiksoftware Blender (vgl. Blender Foundation, c) sowie die nötigen Aspekte für die Arbeit mit dem Programm. Der Aufbau der Wellenfeldsynthese-Anlage der HAW Hamburg ist ebenfalls in diesem Kapitel enthalten.

Darauf folgt die Analyse der Anforderungen der Automatisierung und der Quelldaten. Hier werden die allgemeinen Erwartungen an die zu entwickelnde Software gestellt und dafür benötigte Informationen ergründet. Im darauf folgendem Kapitel wird die durchgeführte Analyse in einem Konzept zusammengefasst.
Als nächstes folgt die Beschreibung der Implementierung der Software, welche auf dem vorher aufgebauten Konzept basiert. Hier werden ausführlich die Funktionen und die Zusammenhänge der einzelnen Klassen der entworfenen Software vorgestellt. Um die Funktionalität der entwickelten Software zu bestätigen wird in dem nächsten Kapitel die Validierung und die Testläufe durchgeführt.
Am Ende wird die Arbeit kurz zusammengefasst und es wird ein Ausblick gegeben.

2. Grundlagen der 3D-Video- und Audio-Modellierung

In diesem Kapitel werden die nötigen Grundlagen der Video- und Audio-Modellierung zum Verständnis der Arbeit erläutert. Am Anfang wird eine Übersicht zu Blender gegeben und die für das entwickelte Programm wichtigen Aspekte (wie z. B. das COLLADA-Format) beschrieben. Danach wird ein Überblick über das Wellenfeldsynthese-Labor geschaffen, das für den 3D-Sound als Ausgabe dient. Im Anschluss wird das OpenSound Control-Protokoll angeschnitten.

2.1. Die Grundbausteine von Blender

Blender ist eine freie 3D-Grafiksoftware der Blender Foundation (vgl. Blender Foundation, c). Es ist ein vollfunktionsfähiges Programm, das bei der Modellierung, Simulation, (3D-)Animation, Rendering, Spieleentwicklung und weiteren ähnlichen Gebieten zum Einsatz kommt. Das ursprünglich von dem Animationshaus „NeoGeo" im Jahr 1998 entwickelte Programm Blender wurde seit 13. Oktober 2002 unter der GNU General Public License (abk. GPL) veröffentlicht (vgl. Wartmann, 2011). Seit dem wurden mehrere Projekte erstellt (vgl. Wartmann, 2011). Zu diesen zählt auch ein open-source Film von „Project Peach" mit dem Titel „Big Buck Bunny" (vgl. Project Peach, b), der als Grundlage für diese Arbeit dient.
Um das Wissen über 3D-Animation zu vertiefen, werden im Folgenden die grundlegenden Elemente einer Blender-Umgebung erläutert.

2.1.1. Koordinatenraum

Die Weltdarstellung in Blender ist dreidimensional. Um einen Punkt im Raum zu beschreiben, wird ein Startpunkt ausgewählt. Bezüglich dieses Startpunktes ist es möglich die Koordinaten des Objektes zu bestimmen (vgl. Wartmann, 2011). Für die Objekte in Blender ist es das Zentrum des Blender-Raumes, der gleichzeitig auch der Nullpunkt ist (s. Abbildung 2.1).
Oft kommt es vor, dass ein Mesh mit einer Armature (s. Kapiteln 2.1.2 und 2.1.3) zu komplex wird. In diesem Fall ist es möglich, die Armature in einer Datei zu speichern und in die andere

Umgebung zu importieren. Bei der Übertragung wird die Information über die Position des importierten Meshs nicht übertragen. Das bedeutet, auch wenn sich ein Objekt in dieser Szene bewegt, bleiben die Koordinaten des importierten Meshs gleich. Was sich bei der Animation bewegt, ist die Armature des Skeletons. Auf die Koordinaten der einzelnen Knochen des Skeletts kann man jedoch zugreifen (näher dazu im Kapitel 2.2).

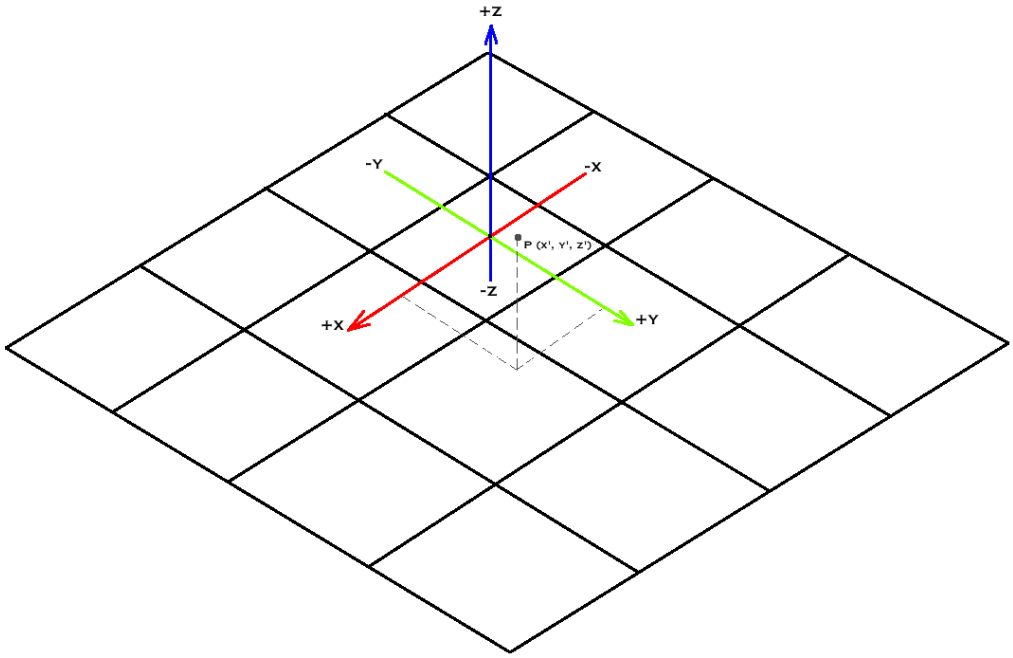


Abbildung 2.1.: Schematische Darstellung eines Punktes in Blender. Der Startpunkt ist der Nullpunkt.

2.1.2. Mesh

Das kleinste Element der polygonalen Modellierung in Blender ist ein Raumpunkt (*Vertex*). Zwischen zwei Vertices entsteht eine Kante (*Edge*), zwischen drei Kanten bildet sich eine Fläche. Zusammen formen sie ein Netz (*Mesh*), welches die Form eines Objektes bestimmt (z. B. einen Würfel, s. Abbildung 3.1) (vgl. Wartmann, 2011). Der Mesh hat keine innere Struktur und erhält die Information nur über die Hülle des Objektes.

2.1.3. Armature

Ein Skelettsystem (*Armature*) ermöglicht die Kinematik und kann nach Bedarf das umliegende Mesh verformen (vgl. Wartmann, 2011). Das System besteht aus mehreren Knochen (*Bone*), die im Bild als Pyramiden, mit Kugeln an den Spitzen, zu sehen sind (s. Abbildung 2.3). Auf der linken Seite ist ein einfaches Mesh zu sehen, welches einen Menschen darstellt. Die Knochen sind in den Armen und Beinen zu sehen. Auf der rechten Seite der Abbildung 2.3 ist das Modellieren eines Arms dargestellt. Im Vergleich zu der linken Armature enthält dieser eine komplexere Struktur.

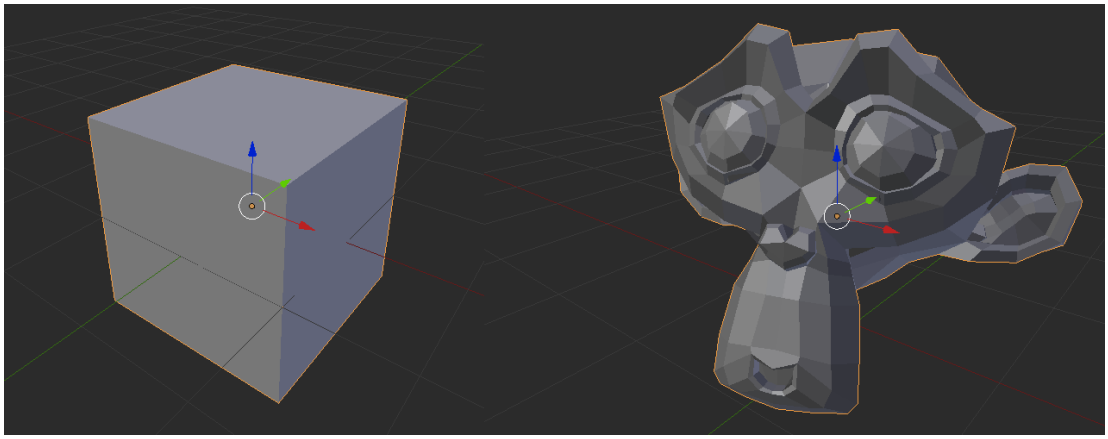


Abbildung 2.2.: Mesh in Blender (links: ein einfacher Würfel, rechts: komplexes Beispielpolygon - ein Affenkopf)

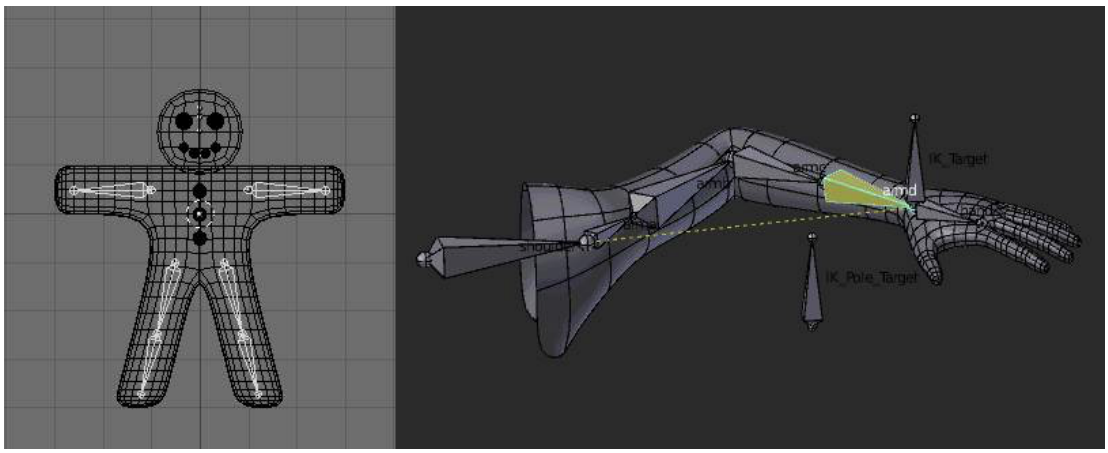


Abbildung 2.3.: Eine Armature in Blender (vgl. Blender Foundation, b)

2.1.4. Scene

Die Welt in Blender stellt ein dreidimensionalen Raum dar, in dem verschiedene Film-Objekte (Meshs, Kameras, Lichter usw.) enthalten sind. Eine Szene (*Scene*) enthält eine Ansicht der Welt, aufgenommen von einem definierten Blickwinkel der Kamera (eine Blender-Datei kann mehrere Szenen enthalten). Ein Film setzt sich aus einer oder mehreren Szenen zusammen, der sich in verschiedenen Containerformaten speichern lässt.

2.1.5. Container

Eine Containerdatei (auch Container genannt) speichert die unterschiedlich kodierten Audio- und Videospuren in einer gemeinsamen Struktur. Die Containerformate legen die Struktur der Speicherung fest. Zu solchen Formaten gehören z. B. die unabhängigen MPEG-Formate (vgl. Kersken, 2013).

Ein Container ist eine komprimierte Datei und somit nicht für weiteres detailliertes Bearbeiten geeignet. Daher kann man unkomprimierte Formate der Blender-Umgebung nutzen, um die Objekte in die Umgebung zu importieren oder zum Export bereitzustellen.

2.1.6. Import/Export Möglichkeiten

Blender unterstützt das Importieren und das Exportieren von mehreren Dateiformaten, z. B. Wavefront (.obj), Autodesk (.3ds), COLLADA (.dae). Diese Dateiformate sind nicht Blender-spezifisch und können auch von anderen Grafikprogrammen erstellt und genutzt werden. Die am häufigsten genutzten Formate sind in der Standard-Konfiguration enthalten, darüberhinaus werden spezielle Formate durch Add-ons unterstützt (vgl. Blender Foundation, b). Das Wavefront-Format definiert die Geometrie und andere Eigenschaften für Objekte. In diesem Format werden die Daten im ASCII-Format gespeichert. Das .3ds-Format gibt die Möglichkeit die Blender-Elemente in der Autodesk-Software (vgl. Autodesk) zu benutzen. Eine .3ds-Datei ist eine binäre Datei und ist somit für die weitere Bearbeitung nicht einsetzbar.

2.2. COLLADA-Format

COLLADA ist ein universelles XML-basiertes Schema. Es ermöglicht den Transport von 3D-Inhalten zwischen den Animations-Anwendungen, bietet umfassende Codierung der visuellen Szenen einschließlich der Geometrie, der Schatten und Effekte, der Physik, der Animation und der Kinematik.

Mit Hilfe von COLLADA können vielfältige 3D-Redaktions- und Verarbeitungstools in eine leistungsfähige Produktionskette kombiniert werden. Die Spezifikation von dem COLLADA-Format wird von Khronos Group betreut (vgl. Khronos Group).
Das Exportieren einer Grafikszene in das COLLADA-Format erzeugt eine .dae-Datei (Abkürzung von Digital Asset Exchange). Die Struktur des neuen XML-Dokumentes ist von der Komplexität der Szene abhängig. Der Körper einer .dae-Datei enthält folgende Tags: *asset*, mehrere *libraries* und *scene*, welche im folgenden genauer erläutert werden (vgl. Khronos Group).

asset Definiert die Informationen im Bezug auf das Parent-Element und enthält Punkte wie den Autor und das Erstellungs-/Veränderungsdatum.

library_cameras Bibliothek der Kamera-Elemente. Eine Kamera verkörpert den Augenpunkt des Betrachters auf die visuelle Szene, hat eine definierte Position und die eingestellte Drehung in einer Szene.

library_lights Bibliothek der Lichter-Elemente. Ein Licht verkörpert eine Lichtquelle (Umgebungslichtquelle, Punktlichtquelle).

library_images Bibliothek für die Ablage der Bilder-Inhalte. Das Element repräsentiert entweder ein Bild der Daten oder darstellbarer Objekte, die ihre Daten später empfangen.

llibrary_effects Bibliothek der Effekt-Elemente. Ein Effekt definiert die Gleichungen, die das optische Erscheinungsbild verarbeiten.

library_materials Bibliothek zur Lagerung der Material-Elemente. Ein Material instanziert ein Effekt und füllt dessen Argument mit Werten.

library_geometries Bibliothek zur Lagerung der visuellen Formen und des Aussehens eines Objektes in einer Szene.

library_animations Bibliothek der *animation*-Tags (s. Unterkapitel ”animations” - 2.2).

library_controllers Bibliothek der allgemeinen Steuerinformationen für dynamische Inhalte. Ein Controller enthält Elemente, welche die Manipulation der Daten beschreiben. Die tatsächliche Art der Daten wird durch die untergeordneten Elemente detailliert dargestellt.

library_visual_scenes Bibliothek der visuellen Szenen. Eine visuelle Szene verkörpert den gesamten Satz von Informationen, die aus dem Inhalt einer COLLADA-Datei sichtbar werden kann. Eine Szene enthält alle Objekte der Szene.

scene Deklariert die Basis der Szenenhierarchie. Das *scene* ist nur einmal in der COLLADA-Datei enthalten.

Um ein Objekt in dem Koordinatenraum darzustellen, wird die Rotation, die Position und die

Skalierung in mehreren verschiedenen *animation*-Tags festgehalten. Für jede Koordinatenachse existiert ein *animation*-Tag.

animation-Tag

Ein animation-Tag der COLLADA-Datei kategorisiert die Deklaration der Animationsinformation. Die hierarchisch aufgebaute Struktur einer Animation enthält durch Sampler-Funktionen beschriebene Elemente und ist zu den gruppierten Animationen, die zusammen ausgeführt werden sollen, geordnet. Die Animation beschreibt die Transformation eines Objektes oder die Veränderung des Wertes über die Zeit (vgl. Khronos Group).
Eine gemeinsame Animationstechnik ist eine key-frame (Schlüssel-Bild) Animation. Ein key-frame ist das zweidimensionale Abtasten von Daten. Die erste Dimension ist *input* und repräsentiert in der Regel die Zeit. Die zweite Dimension ist *output* und enthält die zu animierenden Werte. Mit einer Reihe von key-frames und einem Interpolationsalgorithmus werden die Zwischenwerte für die Zeiten zwischen den key-frames berechnet (vgl. Khronos Group).
Jedes Objekt, beschrieben in einem *animation*-Tag, hat ein Attribut *id* mit folgendem Aufbau: <Name des Objektes>_<Verformungsart>_<Koordinatenachse>. Name des Objektes wird bei der Modellierung festgelegt. Die Verfomungsart kann *location* für die Position, *rotation* für die Rotation und *scale* für die Skalierung sein. Bei der Koordinatenachse werden alle drei Richtungen - x, y, z - betrachtet.
Der *animation*-Tag kann folgende Kinder-Elemente enthalten: mehrere *source*-Tags, welche die Daten entsprechend der Semantik des Elternelementes bietet, und einen *sampler*-Tag, welcher die Interpolation-Funktion für die Animation erklärt. Zu den Kinder-Elementen des *source*-Tags unter anderen kann ein Array-Element (z. B. *float_array*, *int_array*, *Name_array*) und *technique_common* gehören. *technique_common* gibt in den Kinder-Elementen die Information für dieses Element wie der Name und der Typ, die Argumente von *float_array* sind die dazugehörige Werte des Elementes (vgl. Khronos Group).

```
1  <library_animations>
2    <animation id="Cube_location_X">
3      <source id="Cube_location_X-input">
4        <float_array id="Cube_location_X-input-array" count="2"
           >0.04166662 1.291667</float_array>
5        <technique_common>
6          <accessor source="#Cube_location_X-input-array" count="2"
             stride="1">
7            <param name="TIME" type="float"/>
```

```
8           </accessor>
9         </technique_common>
10      </source>
11      <source id="Cube_location_X-output">
12        <float_array id="Cube_location_X-output-array" count="2"
              >3.18133 -1.1516</float_array>
13        <technique_common>
14          <accessor source="#Cube_location_X-output-array" count="2"
                stride="1">
15            <param name="X" type="float"/>
16          </accessor>
17        </technique_common>
18      </source>
19      ...
```

Listing 2.1: Ein Cube-Mesh-Datenausschnitt erzeugt mit Blender und exportiert im COLLADA-Format. Der Cube bewegt sich von einem Punkt zu dem anderen Punkt.

Im Listing 2.1 kann man einen Ausschnitt einer .dae-Datei betrachten. Das Attribut *id* in der Zeile zwei enthält den Namen „Cube", die Bezeichnung *location* für die Position und *X* für die x-Koordinaten-Achse. In den Zeilen drei, 11 und 19 sind die Kinder-Elemente, die *source*-Tags zu sehen. Der erste *source*-Tag enthält zwei Werte für die Zeit als Argument des *float_array*-Tags. Der zweite *source*-Tag gibt die Information über die x-Position des Objektes zu diesen Zeitpunkten. Allgemein betrachtet kann man von diesem Beispiel ablesen, dass das Objekt "Cube" zwei Zeitpunkte hat, in dem sich das Objekt von einem zu dem anderen Punkt bewegt.

Translationsmatrix

Das Listing 2.1 zeigt den Ausschnitt der Darstellung eines Meshs im COLLADA-Format. Die Bones eines Skeletons werden jedoch in der Form einer Matrix dargestellt. Die Matrix wird als Argument von einem *array*-Element wiedergegeben und enthält alle Transformationen des Bones. Der Aufbau einer Matrix wird in dem *accessor*-Tag beschrieben, der ein Kind-Element des *technique_common*-Tags sein kann. In dem *count*-Attribut wird die Anzahl der Zeilen und in dem *stride*-Attribut wird die Anzahl der Spalten festgelegt. Die Ausgabe des *accessor*-Tags definiert die Anzahl und die Reihenfolge der *param*-Kinder-Elemente (vgl. Khronos Group). Ein *param*-Tag enthält ein *name*-Attribut und ein *type*-Attribut. Ist das *name*-Attribut nicht vorhanden, werden die Werte in der Ausgabe ignoriert. Das *type*-Attribut enthält die Information über den Typ der Matrix. Einer der Formate ist float4x4 und steht für eine Matrix

in Form von einem Quaternion. Mit Hilfe von einem Quaternion ist es möglich, die Drehungen in alle Richtungen und eine Verschiebung darzustellen. Ein Quaternion besteht aus zwei Teilen. Der obere Teil enthält die x-, y- und z-Komponenten, welche die Achsen darstellen, um welche eine Drehung oder die Verschiebung passieren soll. Der untere Teil ist eine w-Komponente, die den Betrag der Drehung um dieser Achse darstellt.

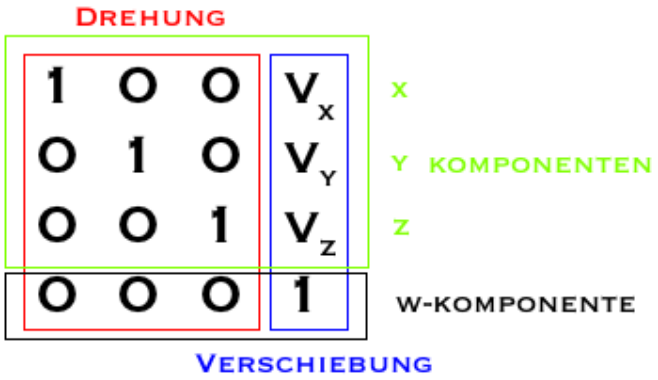


Abbildung 2.4.: Eine Matrix in Form von einem Quaternion.

In der Abbildung 2.4 wurde jede der Komponenten farblich hervorgehoben. Der rot markierte Bereich enthält die Beträge zur Drehung und der blaue Bereich enthält die Einheiten zur Verschiebung des Objektes. Der grüne Anteil stellt die Achsen-Komponente dar, der schwarze Bereich repräsentiert die w-Komponente. Die letzte Spalte enthält den Vektor mit Beträgen, um die das Objekt verschoben werden soll. Somit überführt diese Matrix v_x Einheiten entlang der x-Achse, v_y Einheiten entlang der y-Achse und v_z Einheiten entlang der z-Achse. Wenn die Einheiten v_x, v_y und v_z dem Nullvektor entsprächen, passiert keine Verschiebung.

Globale und objektbezogene Positionierung des Blender-Objektes

Die Darstellung eines Objektes an einer Position in Blender setzt sich aus zwei Sichten zusammen. Diese Sichten sind als zwei Matrizen dargestellt. Die Multiplikation dieser zwei Matrizen miteinander ergibt die Position des Objektes im Blender-Raum. Die erste Matrix ist die *matrix_world* und enthält die Transformationsmatrix des gesamten Raumes der Szene. Die zweite Matrix ist die *PoseBone.matrix*, welche eine endgültige Transformation des Objektes im Raum beschreibt (vgl. Blender Foundation, a).

2.3. Das Wellenfeldsynthese-Labor der HAW Hamburg

Die Wellenfeldsynthese (abk. WFS) ist ein Audiowiedergabeverfahren bei dem es möglich ist, die virtuellen Schallquellen innerhalb und außerhalb des reellen Hörraumes frei zu platzieren. Die physikalische Grundlage der WFS bildet das Huygenssche Prinzip (s. Abbildung 2.5). Es erklärt solche grundlegende Phänomene wie Brechung, Reflexion und Wellenausbreitung. Nach dem Huygensschen Prinzip wird jeder Punkt einer Wellenfront als der Ausgangspunkt einer neuen Elementarwelle, die phasengleich zur ursprünglichen Welle ist, betrachtet (vgl. Slavik und Weinzierl, 2008). Durch die Überlagerung von dem Lautsprecher-Array mit Tonsignalen wird das gewünschte Schallfeld erzeugt (vgl. Fohl, 2013).

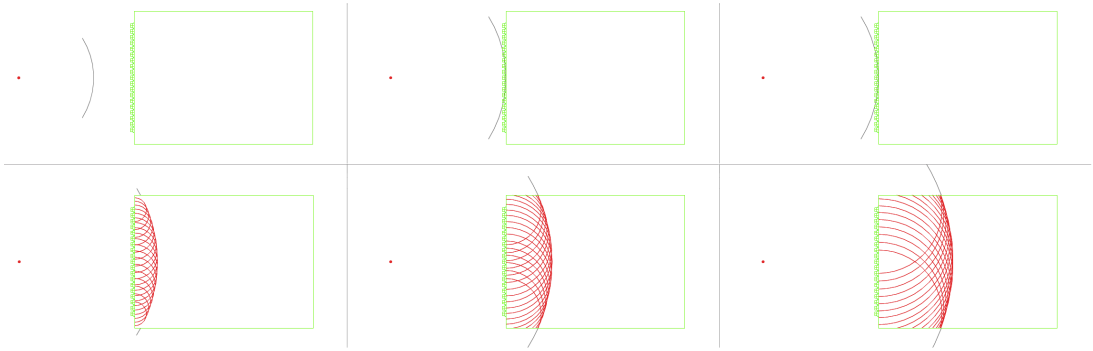


Abbildung 2.5.: Ausbreitung der Welle nach dem Huygensschem Prinzip (vgl. Oellers)

Der Laborraum der HAW Hamburg wurde im Jahr 2011 mit einem WFS-System ausgestattet, um eine Augmented-Reality-Audio (abk. ARA) Umgebung zu schaffen. Diese besteht aus 26 einzelnen Lautsprecher-Modulen, die in der Höhe von zwei Metern in einem Rechteck aufgehängt sind. Jedes Modul (s. Abbildung 2.6) hat acht Lautsprecherkanäle, welche 10 cm Abstand zwischen einander besitzen, und zwei Woofer (vgl. Fohl, 2013).

Abbildung 2.6.: Lautsprecher-Modul (vgl. FourAudio)

Netzwerke des Labors

Die Systemkommunikation erfolgt über zwei Netzwerke. Die Ansteuerung der einzelnen Kanäle geschieht über ein Dante Audionetzwerk der Firma Audinate und ist ein IP-basiertes Netzwerk (vgl. Audinate). Das ermöglicht die Verwendung einer Software-Soundkarte (Audinate Dante DVS, 64 Kanäle) und reduziert die Verkabelung auf ein einfaches Standard-Ethernet (vgl. Makarski u. a.). Durch die Kombination der Software mit Dante-Netzwerk wird die Verwaltung 128 separate Kanäle ermöglicht (vgl. Fohl, 2013). Das andere Netzwerk dient zur Kommunikation der WONDER-Komponenten (vgl. Johns, 2014). Das gesamte Netzwerk ist in der Abbildung 2.7 zu sehen.

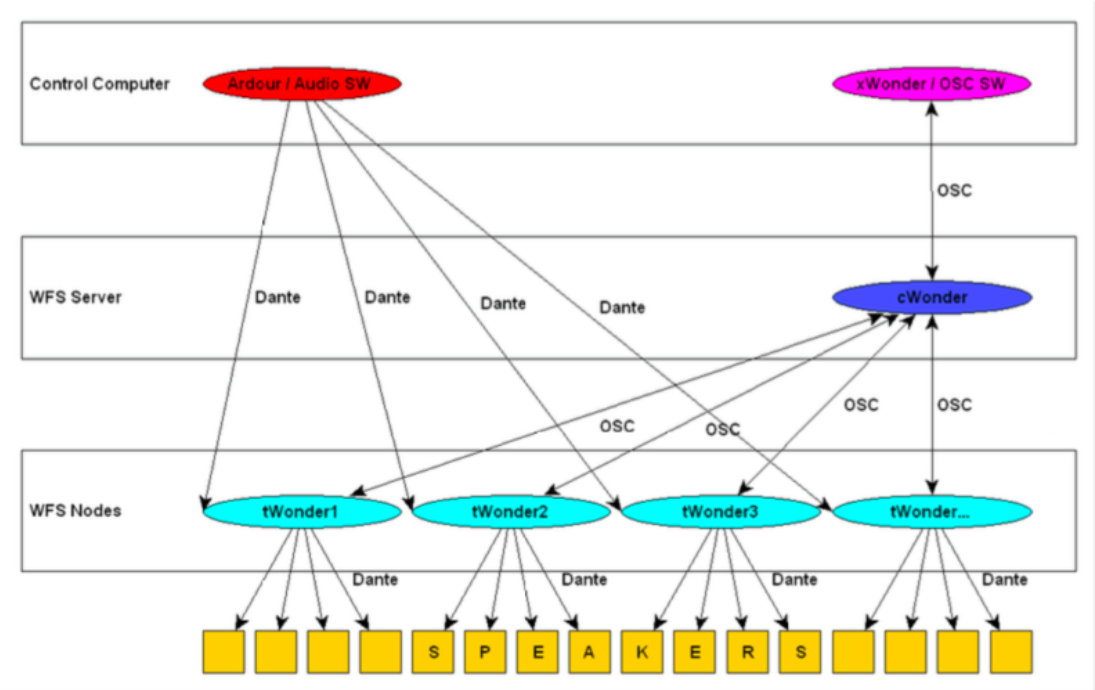


Abbildung 2.7.: Netzwerkstruktur einer WFS-Anlage (vgl. Makarski u. a.)

JACK Audio Connection Kit

Aufgrund des Aufbaus des Dante-Netzwerks erscheint die Sound-Karte mit 128 Eingängen und 128 Ausgängen im System. Jeder Eingang oder Ausgang repräsentiert eine Sound-Quelle der WFS und somit ist die maximale Anzahl der Quellen auf 128 limitiert. Das Routing zwischen den Dante Ein-/Ausgängen und der Audio-Software ermöglicht JACK Audio Connection Kit (abk. JACK) (vgl. Fohl, 2013). JACK ist ein virtueller Audio-Server, welcher das Audio-Routing sowohl zwischen den Anwendungen als auch zwischen den Geräten ermöglicht. Es kann

mehrere Anwendungen zu einem Audiogerät verbinden. JACK wird unter Linux, OS X, Solaris, FreeBSD und Windows unterstützt (vgl. Davis und Letz). Zur Steuerung von JACK wird die Software Qjackctl (vgl. QjackCtl) genutzt. Dadurch ist es möglich, die Ausgangskanäle einer Anwendung an die Kanäle der WFS-Anlage zu legen.

OpenSound Control

Die Sound-Quellen der WFS-Anlage kann man über OpenSound Control-Nachrichten (abk. OSC) fernsteuern (vgl. OpenSound Control). OSC ist ein Protokoll für die Kommunikation zwischen den Computern und Multimedia-Geräten und ist für die moderne Netzwerktechnologie optimiert. In der Tabelle 2.1 sind ein paar Beispiele der Nachrichten zur Steuerung der Anlage aufgelistet. Die ID wird als eine Integer-Zahl und weitere Koeffizienten werden als Float-Zahlen verschickt.

OSC-Nachricht	**Bedeutung**
/WONDER/source/activate(id)	Aktiviert die virtuelle Sound-Quelle mit dem übergebenem ID
/WONDER/source/position(id, x, y)	Positioniert die virtuelle Sound-Quelle mit dem übergebenem ID an die Position (x, y)
/WONDER/source/position(id, x, y, t)	Bewegt die virtuelle Sound-Quelle mit dem übergebenem ID an die Position (x, y) in der übergebenen Zeit t (in Sekunden)
/WONDER/source/deactivate(id)	Deaktiviert die virtuelle Sound-Quelle mit dem übergebenem ID

Tabelle 2.1.: Beispiele der OSC-Nachrichten zur Steuerung der WFS-Anlage

Die OSC-Nachrichten können nicht nur zur Steuerung der WFS-Anlage genutzt werden, sondern auch zu der Steuerung der Audio-Software. Diese wird auch benötigt, um ein Sound in der Anlage wiederzugeben. Je nachdem, wie viele Kanäle die Software unterstützt, kann man die Audiospuren auf eine oder mehrere Sound-Quellen verlegen. Ein mächtiges Programm in der Reihe von anderen Programmen, die in der Verbindung mit der Anlage genutzt werden, ist Ardour (vgl. Ardour). Ardour ist eine digitale Workstation zum Erstellen und Bearbeiten von Audiodateien. Die Software unterstützt mehrere Kanäle und kann außerdem per OSC-Nachrichten gesteuert werden.

SuperCollider

Eine andere Software, welche sich auch mit den OSC-Nachrichten beschäftigt, ist SuperCollider (vgl. SuperCollider, a) . SuperCollider (vgl. SuperCollider, a) ist eine objektorientierte Programmierumgebung für Echtzeit-Audio- und Videoverarbeitung. Es ist eine vielseitige Umgebung für die Signalverarbeitung und für die Erstellung von Musik-Anwendungen (vgl. SuperCollider, b). In dieser Vielfalt der Möglichkeiten kann man auch aus der Software die OSC-Nachrichten verschicken. Ein Beispiel hierfür ist in dem Listing 2.2 dargestellt. Die erste Zeile in dem Beispiel zeigt den allgemeinen Aufbau einer Nachricht in SuperCollider.

```
1 NetAddr(hostname, port).sendMsg(args);
2
3 NetAddr("192.168.14.100", 58100).sendMsg("/WONDER/source/position",
    0, 1, 1);
4 NetAddr("192.168.14.101", 3819).sendMsg("/ardour/transport_stop");
```

Listing 2.2: Ein Beispiel SuperCollider-Datei.

Eine Nachricht benötigt den Host und den Port des Empfängers. Danach folgt die OSC-Nachricht und eventuelle Parameter zur Spezifizierung. In der Zeile 3 in dem Listing 2.2 wird der Aufbau einer OSC-Nachricht zur Kommunikation mit der WFS-Anlage der HAW Hamburg dargestellt. Diese verschickt an die Adresse 192.168.14.100 und den Port 58100 die Nachricht mit dem Inhalt zur Positionierung einer Schallquelle mit der Source-ID 0 an der Position (1, 1). In der Zeile 4 in dem Listing 2.2 wird eine OSC-Nachricht zum Stoppen der Wiedergabe in Ardour (vgl. Ardour) verschickt.

Koordinatensystem im Labor

Die OSC-Nachrichten mit der Positionsangabe ermöglichen die Positionierung einer Sound-Quelle an den bestimmten Koordinaten. Das Koordinatensystem in dem WFS-Labor ist aufgrund des Aufbaus zweidimensional (ohne z-Achse). Das bedeutet, die vertikale Bewegung im Raum ist für den Sound irrelevant. Die Belegung der x- und y-Achsen ist in der Abbildung 2.8 dargestellt.

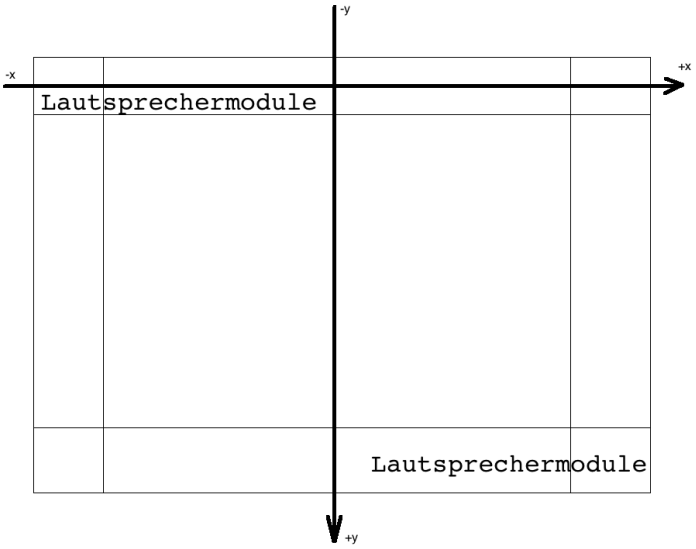


Abbildung 2.8.: Schematische Darstellung des Koordinatensystems in dem WFS-Labor.

3. Anforderungsanalyse

In diesem Kapitel wird eine Analyse der Ausgangslage durchgeführt. Als erstes werden die Anforderungen geschildert. Danach wird ein Überblick über die Quelldaten des Projektes „Big Buck Bunny" erschaffen. Im Weiteren wird die Möglichkeit der Herstellung einer Verbindung zu der WFS-Anlage erläutert. Am Ende werden die Ergebnisse zusammengefasst.

3.1. Zielsetzung

Es soll eine Software entwickelt werden, mit der es möglich ist, die Objekte eines Films mit den virtuellen Sound-Quellen einer WFS-Anlage zu verknüpfen. Dazu sollen die Koordinaten des Objektes im Film automatisch extrahiert werden. Als Grundlage wurde das Projekt „Big Buck Bunny" ausgewählt, da der gesamte Film zum größten Teil mit einer freien Software hergestellt wurde (mehr dazu im Kapitel 3.2). Außerdem soll es ermöglicht werden, diese Daten an die WFS-Anlage weiterzugeben, welche für die Ausgabe des Sounds genutzt werden soll. Dafür muss bestimmt werden, welche Daten sowohl auf der Video-Seite als auch auf der Audio-Seite benötigt werden.

3.2. Quelldatenanalyse

Der kurze Animationsfilm „Big Buck Bunny" ist ein im Jahr 2008 veröffentlichtes open-source Projekt der Gruppe „Projekt Peach" (vgl. Project Peach, b).
Die Erstellung und die Animierung der Tiere wurde hauptsächlich in Blender durchgeführt. Die Audiospuren wurden mit Hilfe von Audacity (vgl. Audacity) zusammengelegt. Die einzige kommerzielle Software ist Reaktor von Native Instruments (vgl. Native Instruments). Das Programm wurde zum Erstellen der Audio-Effekte des Projektes genutzt. Alle freien Quelldaten sind im Netz zum Download bereitgestellt (Project Peach, a).

Abbildung 3.1.: Der Hase Big Bunny und drei Waldtiere: links - ein Eichörnchen und ein Chinchilla, rechts - ein fliegendes Eichörnchen (vgl. Project Peach, b)

Der fertige Film ist in verschiedenen Video-Formaten verfügbar. Diese sind als Containerdateien bereitgestellt und enthalten somit keine Information über innere Objekte, da diese bei der Komprimierung wegoptimiert werden. Mit Hilfe der bereitgestellten Ressourcen der Video-Daten kann jede interessierte Person den Film nachproduzieren oder sogar den Verlauf der Geschichte neu schreiben. Die kompletten Szenen sind in 13 Ordner unterteilt. Jeder Ordner enthält einen Filmabschnitt jeweils als .blend-Datei. Jede Blender-Datei dieses Projektes enthält eine Szene. Die einzelne Kreaturen sind als Skeletons (ein Mesh mit einer Armature) in getrennten Dateien verfügbar. Die weiteren Eigenschaften wie die Umgebung, die Texturen, die Eigenschaften usw. sind ebenfalls als .blend-Dateien vorhanden.
Eine .blend-Datei ist eine Binär-Datei und ist außerhalb von Blender nicht zum Lesen geeignet. Es gibt mehrere Möglichkeiten, eine Szene in Blender in ein anderes Format zu exportieren und somit in einer lesbaren Form darzustellen (mehr dazu in Kapitel 2.1.6). Eine der umsetzbaren Lösungen dafür ist der Export in eine .dae-Datei. Aufgrund der XML-basierten Struktur ist es möglich eine .dae-Datei zu parsen und somit an die nötigen Komponenten zu gelangen.
Der Aufbau der COLLADA-Datei für die Animationsobjekte ist sehr allgemein und umfangreich, um das Format möglichst universell zu halten. Allerdings werden nicht alle verfügbare Informationen benötigt, um die Positionen der Objekte der .blend-Datei als Sound-Quellen der WFS-Anlage darzustellen. Aufgrund dessen kann man sich auf den Teil der .dae-Datei fokussieren, welche die Koordinaten und die Zeitpunkte darstellt. Anhand der *animation*-Tags in der

library_animation kann man nach in der Szene enthaltenden Objekten suchen. Die Anzahl der Tags ist in der Library verdreifacht, da für jedes Objekt drei Koordinatenachsen-Informationen exportiert werden. Die Suche wird aber durch folgende Merkmale verfeinert:

- Der aktuelle Aufbau der Anlage ermöglicht die Bewegung der Sound-Quellen nur in einer horizontalen Ebene. Somit entfällt der Informationsbedarf an der z-Achse. Daher können die Tags mit den z-Koordinaten ignoriert werden.
- Ein *source*-Tag von einem *animation*-Tag eines Meshs sollte nur die Bezeichnung *location* enthalten, da diese für die Position des Objektes zuständig ist. Die weiteren möglichen Bezeichnungen wie *scale* und *rotation* können ignoriert werden.
- Die Armature ist in Form einer 4×4-Matrix bereitgestellt. Da die Rotationsinformation keinen Einfluss auf die Position hat, sollten nur die Positionskoordinaten aus der Translationsmatrix (mehr dazu im Kapitel 2.2) herausgefiltert werden. Das bedeutet, die Auswertung verfeinert sich auf einen Translationsvektor, welcher sich aus dem vierten und achten Element des Translationsmatrix-Arrays zusammensetzt.

Die importierten Skeletons können aufgrund der verlorenen Mesh-Informationen anhand deren Armaturen identifiziert werden. Die Armature kann mehrere Knochen enthalten. Daher ist es wichtig, die Knochen, welche für den Sound relevant sind, mit einem Identifikationsmerkmal zu versehen. Im allgemeinen Fall sollte dieser Knochen die Position des Objektes in der Welt bestimmen.
Mit der Annahme der Vergabe der Bezeichnung könnten nicht nur die Hauptknochen, sondern auch andere Knochen für die Verknüpfung mit dem Sound exportiert werden. Zum Beispiel wenn der Sound sich nicht an das Objekt, sondern nur auf ein Teil des Objektes beziehen soll, wird die Positions dieses Teils relevant. In der Abbildung 3.2 ist ein Beispiel dargestellt, anhand welchen sich der Fall erklären lässt. Es ist eine Armature, welche aus mehreren Knochen besteht. Die Armature selbst bleibt an einer Stelle, jedoch erfolgt eine Bewegung von der Hand a (siehe Abbildung 3.2), an der sich eine Sound-Quelle befindet. Für die Position des gesamten Objektes in der Welt würde die Position der Wirbelsäule ausreichen. Aber da sich die Wirbelsäule nun nicht bewegt, muss die Position der Hand a bestimmt werden. Damit beim Finden der Positionsdaten keine Probleme auftreten, muss bei der Modellierung des Knochens, welcher die Hand a repräsentiert, eine Kennung vergeben werden.

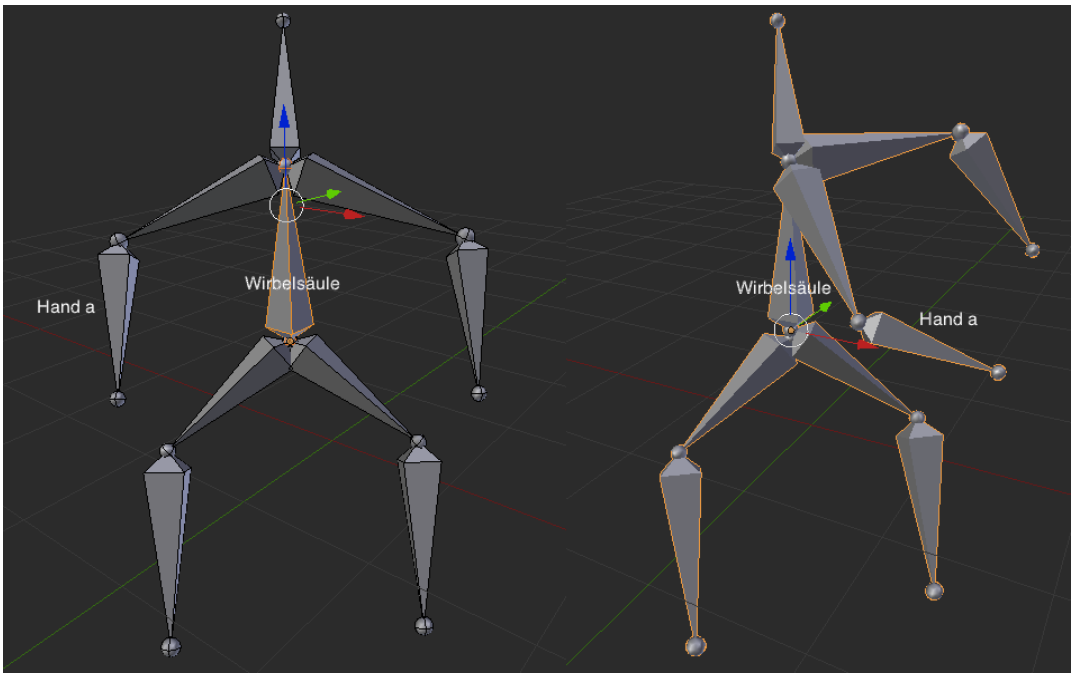


Abbildung 3.2.: Eine Handbewegung von einer Armature

3.2.1. Verschiedene Koordinatensysteme

Das Koordinatensystem von Blender ist dreidimensional. Außerdem ist dem User die Modellierung und Aufnahme von verschiedenen Seiten nicht unterbunden. Das bedeutet, die Welt in Blender könnte von allen Winkeln aus aufgenommen werden. Im Vergleich zu diesem Aspekt ist das WFS-Labor fest an die zweidimensionale Ebene gebunden.
Angenommen, man beachte die Höhe der Blender-Objekte nicht, dann kann man den Raum in Blender zweidimensional darstellen. Trotzdem würde das Problem, bei der Übertragung der Koordinaten an die WFS-Anlage, an zwei Stellen bestehen. Das erste Problem ist der Startpunkt des jeweiligen Koordinatensystem. In Blender befindet der sich in der Mitte des Raumes. Der Nullpunkt des Koordinatensystems in dem Raum des WFS-Labors liegt am Rande (s. Abbildung 3.3). Der zweite Unterschied im Vergleich der x-Achsen deutlich. Diese sind um die y-Achse gespiegelt. Die x-Achse im Blender-Raum zeigt in die entgegengesetzte Richtung der x-Achse des anderen Raumes.

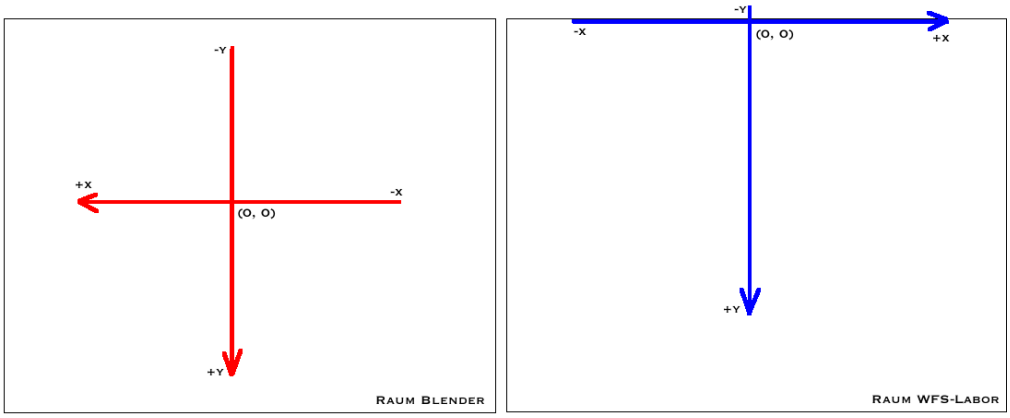


Abbildung 3.3.: Die Koordinatensysteme im Vergleich

Der Richtungsunterschied der beiden x-Achsen lässt sich durch die Multiplikation ausgleichen:

$$x_{\text{Blender}} * (-1) = x_{\text{WFS}}$$

Im Gegensatz dazu stellt die Umrechnung des Startpunktes sich komplizierter dar. Aufgrund der Freiheit der Modellierung der Objekte in Blender ist es möglich, die Richtungen der Bewegungen frei im Raum zu definieren. Das bedeutet, dass sowohl die x-Achse als auch die y-Achse die Bewegung „vorne/hinten“ oder „links/rechts“ repräsentieren können. Und somit ist in jedem Projekt anders. Aufgrund dieser Umstände muss eine Lösung in der zu entwickelnden Software gefunden werden.

3.3. Verbindung zu dem Sound in der Wellenfeldsynthese-Anlage

Um eine Verbindung zwischen einer Audiospur und einem grafischen Objekt herzustellen ist es ausreichend, die Audiospur und das Objekt mit der gleichen Quellen-ID zu erstellen. Um einer Quelle den Sound zuzuweisen, wird eine Audio-Software benötigt. Die Ausgangskanäle dieser Software entsprechen einer Sound-Quelle der WFS-Anlage. Je nachdem, wie die virtuelle Verkabelung in JACK verknüpft wurde, entspricht die eine oder die andere Spur einer virtuellen Sound-Quelle.
Die Zuweisung einer Quellen-ID zu einem Animationsobjekt ist nicht vorhanden. Dieses soll

also in der Software behandelt und vermerkt werden. Die Verknüpfung eines Objektes an eine virtuelle Sound-Quelle kann beim Verschicken der OSC-Nachrichten passieren. Wenn also ein Animationsobjekt eine Quellen-ID zugewiesen bekommt, darf diese ID gleichzeitig nur von diesem Objekt genutzt werden. Mit Hilfe der in der Tabelle 2.1 beschriebenen OSC-Befehle, kann man eine virtuelle Sound-Quelle aktivieren und an einem bestimmten Ort platzieren. Der Ort der Quelle wird durch die Angabe der x- und y-Koordinaten bestimmt. Da die Nachrichten in dem lokalen Netz der WFS-Anlage verschickt werden, wird die IP-Adresse und der Port des Empfängers benötigt.
Die Verknüpfung der Audiospuren und der Animationsobjekte unter solchen Umständen setzt voraus, dass der Sound für den Film fertiggestellt ist. Das bedeutet, die Audiospuren müssen in der richtigen zeitlichen Reihenfolge in einer Audio-Software geordnet sein. Dabei ist es nicht unterbunden, dass eine Spur von verschiedenen Objektsounds genutzt werden kann. Aber wenn dieser Fall auftritt, muss auch das richtige Objekt zum Einsatz kommen. Dafür muss die Verknüpfung der Audio-Software-Ausgänge an die Kanäle der WFS-Anlage bekannt sein.

3.4. Zusammenfassung der Analyse

Die durch die Analyse erschaffenen Ergebnisse bilden eine grundlegende Basis für die zu entwickelnde Software. Es wurde festgestellt, dass die meisten Entwicklungsarbeiten sich der Video-Seite widmen werden. Es müssen die Koordinaten der Objekte exportiert und umgerechnet werden. Es soll auch ermöglicht werden, diesen Objekten eine Quellen-ID zu vergeben. Somit wäre die Verbindung mit den virtuellen Quellen der WFS-Anlage sichergestellt. Außerdem ist durch die Analyse festgestellt worden, dass ein Vermerken von relevanten Knochen für die Wiederfindung in der Software notwendig ist. Somit könnten die Anforderungen an die Software erfüllt werden und die Analyse der Ausgangslage war erfolgreich.

4. Konzept der Software

In diesem Kapitel wird anhand der durchgeführten Analyse das Konzept der Software aufgebaut. Am Anfang wird ein Überblick über die Anforderungen an die Software gegeben. Im Weiteren wird der grobe Arbeitsverlauf vorgestellt, der sich in der Übersicht der packages widerspiegelt. Danach wird das Konzept in einem schematischen GUI-Konzept erweitert und zusammengefasst. Am Ende wird auf die Lösungsansätze eingegangen, welche die Fragen, die im Laufe des Entwurfs entstanden sind, beantworten.

4.1. Übersicht der Anforderungen

Es wird eine Software entwickelt, mit deren Hilfe man die Animationsobjekte eines Films mit dem Sound in einer WFS-Anlage verknüpfen kann. Die visuelle Verfolgung des Films soll somit mit der Position der Sound-Quelle übereinstimmen. Das bedeutet, dass die Position des Wiedergabegeräts, auf dem der Film abgespielt wird, ortsunabhängig sein soll. Außerdem muss in dem zu entwerfendem Programm die Lösung für den Export der Objektkoordinaten der Animationsobjekte des Films gefunden werden.

Konzept Allgemein

Die zu entwickelnde Software wird die Animationsobjekte eines Films auf die virtuellen Sound-Quellen einer WFS-Anlage mappen. Der Film soll aus mehreren Szenen zusammengesetzt werden. Die Szenen werden als .dae-Dateien geladen, da diese die Positionen der Objekte eines Films beinhalten. Wichtig ist, dass die zeitliche Anordnung einer Szene des Films mit dem Verlauf in dem Film übereinstimmt. Jede Szene als .dae-Datei soll von dem Programm geparst werden, wodurch die Animationsobjekte dieser Szene herausgefiltert werden. Die Animationsobjekte werden die Zeitpunkte mit den Objektkoordinaten im Raum festhalten, wobei die einzelnen Koordinaten individuell anpassbar sein sollten. Zum Schluss sollen die Positionen der Animationsobjekte in eine Reihe von OSC-Nachrichten umgewandelt werden. Diese OSC-Nachrichten werden anschließend von dem Programm an die WFS-Anlage zum Abspielen gesendet. Dieses wird voraussetzen, dass die Sound-Spuren in einer benutzerdefinierten

Software in der richtigen zeitlichen Anordnung befinden. Die Verknüpfung der Ausgangskanäle der Audiosoftware mit den Eingangskanälen der WFS-Anlage muss ebenfalls bekannt sein. Es wird außerdem ermöglicht, das Projekt zur weiteren Bearbeitung zu speichern.

4.2. Definition der Arbeitssequenz

Die Idee ist es eine Szene als .dae-Datei einzulesen und die Objekte dieser Szene für die Verknüpfung mit den virtuellen Schallquellen vorzubereiten. Dafür ist eine Software mit zwei Ansichten zu entwickeln. Die erste Ansicht soll die Übersicht über alle Szenen enthalten, die zweite Ansicht wird sich mit den Einstellungen für die Objekte der jeweiligen Szenen beschäftigen. Der grobe Ablauf der Schritte eines Nutzers (im Weiteren User genannt) ist wie folgt:

- User wählt eine Szene als .dae-Datei in der ersten Ansicht aus.
- User gelangt per Knopfdruck in die zweite Ansicht, wo er die Objekte der ausgewählten Szene zu sehen bekommt.
- User wählt ein Animationsobjekt aus und vergibt diesem eine Quellen-ID.
- User wählt die gewünschten Koordinaten des Objektes aus.
- User startet die Szene.

Somit stellen sich zwei konkrete Komponenten der Software dar. Eine Komponente verwaltet die Szenen und die Animationsobjekte jeder dieser Szenen. Die nächste Komponente ist für das Versenden der OSC-Nachrichten zuständig, womit der Vorgang der Bewegung der Quellen in der WFS-Umgebung angestoßen wird. Außer dieser zwei Komponenten benötigt die Software nach den Anforderungen eine Speicherungsfunktion. Somit könnte zum Beispiel der Versandvorgang aufgezeichnet und wiedergegeben werden.

4.3. Programmarchitekturkonzept

In welche Arbeitskomponenten sich die zu entwickelnde Software aufteilen soll, zeigte das Diagramm der packages, welches in der Abbildung 4.1 vorgestellt ist. Nun wird zuerst die Funktionalität von jedem kurz angesprochen und im Weiteren werden die wichtigsten packages näher erläutert.

vidaer bezeichnet das Hauptpackage. Der Name ist von dem Namen der Software abgeleitet. *vidaer* enthält außerdem zwei controller-Klassen, welche für die Funktionalität der GUI sorgen.

fileStructure versammelt die Klassen, die sich mit den Animationsobjekten auseinandersetzen.

loaderStructure enthält die Funktionsklassen des Speicher,- Lade- und Versandvorgangs in einer Datei.

utilityFamily ist eine Gruppe von Hilfsklassen.

oscFamily wird für das Versenden der OSC-Nachrichten benötigt.

senderFamily beinhaltet die Gruppe der Thread-Klassen, die sich mit dem Sendevorgang beschäftigen.

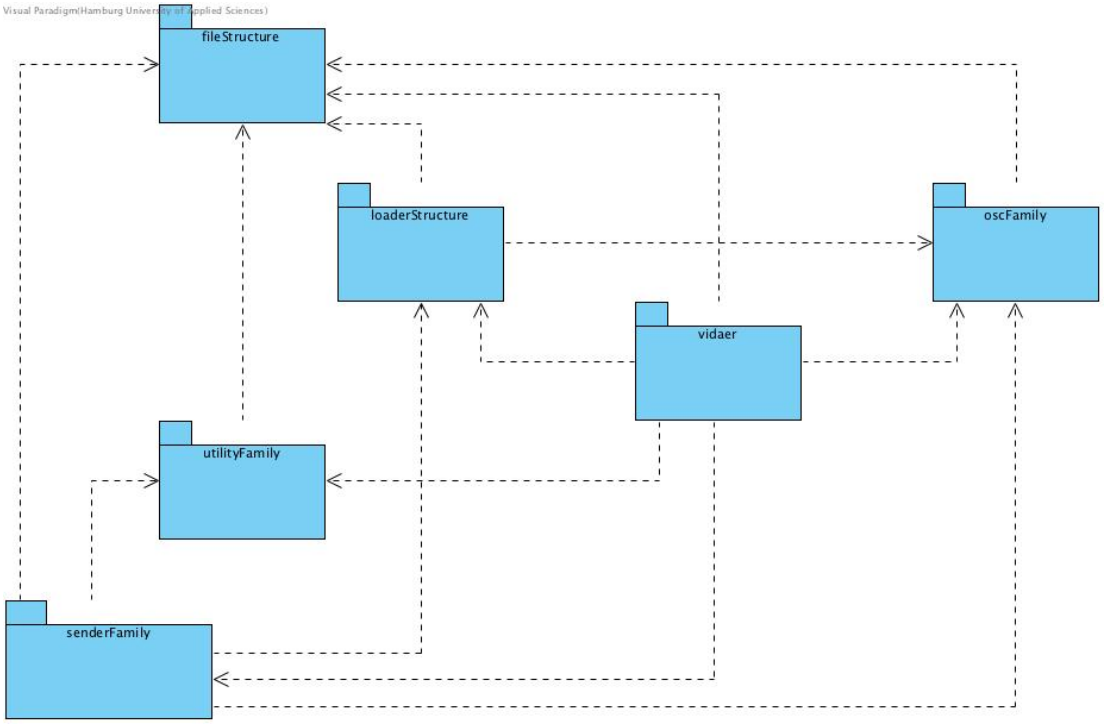


Abbildung 4.1.: Das package-Diagramm der entworfenen Software.

Die meiste Aufmerksamkeit widmet sich dem package *fileStructure*, weil dieses für die Verwaltung der Animationsobjekte zuständig sein soll. Die Idee ist es, die geladene Szene nach den Objekten zu durchsuchen und in der Software aufzulisten. Dafür ist es wichtig zu wissen, welche Szenen geladen sind. Somit entsteht eine *FileDispatcher*-Klasse, die eine Übersicht über die Liste der Szenen behält. Aus dem Grund, dass eine Szene mehrmals benutzt werden könnte, braucht die Software nicht das vielfache Laden der selben Szene unterbinden. Jede Szene kann

einen zeitlichen Abstand zu den anderen Szenen bzw. dem Anfang haben. Somit entsteht eine andere Klasse zur Kontrolle je einer Szene - *AnimationFile*. Diese Klasse soll nur zwei Attribute haben: den zeitlichen Abstand und die Container-Klasse *Animation*. Die *Animation*-Klasse enthält eine Liste der Animationsobjekte. Für die Speicherung der einzelnen Animationen ist die *AnimStruct*-Klasse verantwortlich. In dieser Klasse werden die einzelnen Animationsobjekte der Szene mit eigenem Namen und der zugewiesenen Quellen-ID festgehalten. Da jedes Animationsobjekt Daten wie Zeitpunkt und dazugehörigen Koordinaten enthält, werden diese in eine *AnimationData*-Klasse verpackt. Die Abbildung 4.2 gibt eine Übersicht über die Hierarchie des packages und zeigt die Zusammenhänge zwischen den einzelnen Klassen. Zum Beispiel, ein *FileDispatcher* hat eine Liste von *AnimationFile*-Objekten. Oder ein *AnimationFile* hat ein *Animation*-Objekt.

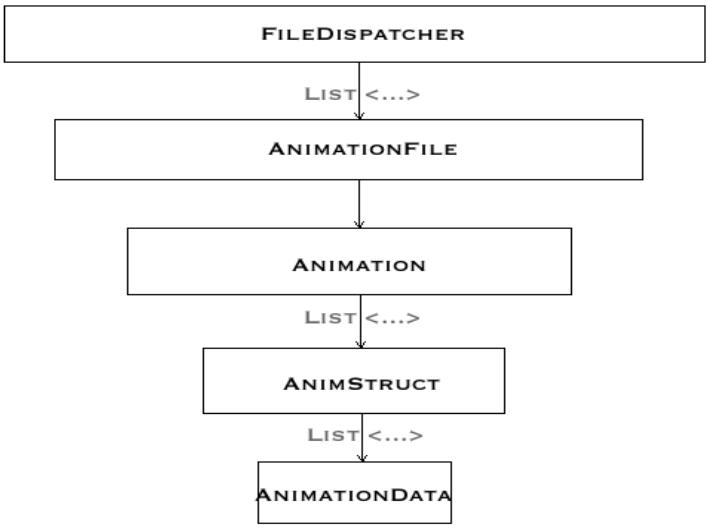


Abbildung 4.2.: Hierarchischer Überblick über das *fileStructure*-package.

Das package *loaderStructure* soll sich den Aktionen, mit einer in der Software ausführbaren Datei, widmen. Dafür werden zwei Klassen benötigt. Die Klasse *RunnerFile* soll die Informationen der Datei in der Software festhalten. Die Klasse *XMLController* soll die Erstellung, die Speicherung und das Laden verwalten. Die Datei wird in einem eigenem Format *.vida* gespeichert. Diese hat eine XML-Struktur. Für das Erstellen der Datei ist die zweite Klasse des packages zuständig.
In dem package *utilityFamily* werden die Klassen versammelt, die bei dem Aufbau der Software helfen sollen. Unter anderen soll hier die Klasse *ObjectDetector* implementiert werden. Diese Klasse entspricht einem XML-Parser, in welchem eine .dae-Datei geparst wird. Aus den in der

Datei enthaltenden Animationen werden die Objekte des *fileStructure*-packages gefüllt. Der *ObjectDetector* muss nach einem festen Muster vorgehen und die Animationen anhand der folgenden Merkmale herausfiltern.

- Ein Animationsobjekt darf keinen Verweis auf das Licht (*light*) in seinem Namen enthalten, damit es einen Unterschied zu den Lichtobjekten geben kann.
- Die .dae-Datei wird sowohl nach Skeletons als auch nach Meshs durchsucht. Enthält ein Skeleton einen Knochen, der später eine Sound-Quelle zugewiesen bekommen soll, muss dieser die Bezeichnung *Root* in dem Namen tragen. Alle andere Knochen (nicht Root-Knochen) werden ignoriert.
- Die Position wird entweder anhand der Bezeichnung *_location* oder *_matrix* identifiziert, je nachdem, ob das Objekt ein Skeleton oder ein einfaches Mesh ist.
- Nach der Identifizierung eines Animationsobjektes werden die Zeit- und die Positionsinformationen exportiert. Die jeweiligen Daten befinden sich in dem float_array-Tag der .dae-Datei. Für die Zeit wird die *input*-Bezeichnung und für die Koordinaten die *output*-Bezeichnung festgelegt.

Nach dem sequentiellem Vorgang der Analyse sind die Animationsobjekte aller geladenen Szenen mit den zugehörigen Informationen bereit zum Darstellen.

4.4. Ablauf in einem GUI-Kozept

Wie schon in Kapitel 4.2 erwähnt, soll die Software aus zwei Ansichten bestehen. Die erste Ansicht (s. Abbildung 4.3) soll globale Informationen und Einstellungen festhalten. In der linken Seite werden die Szenen als .dae-Dateien über den Auswahl-Button der Leiste 1 geladen und in der Liste der geladenen Szenen aufgelistet. In der rechten Seite werden die benötigten Einstellungen für die ausgewählte Szene getroffen. Man könnte also die Szene in eine Richtung verschieben oder drehen. Außerdem soll es in diesem Teil ein Feld für den zeitlichen Versatz der Szene geben. Es wird deswegen benötigt, weil jede Szene bei dem Zeitpunkt 0.00 Sek. beginnt. Es ist unmöglich von der .dae-Datei herauszufinden, in welchem zeitlichen Zusammenhang eine Szene zu der anderen steht. Mit der Eingabe einer Sekundenzahl in diesem Feld der Ansicht, kann der User die gewünschte Sequenz des Filmes aufbauen. Das Video für die parallele visuelle Begleitung ist ebenfalls in dieser Ansicht zu laden. In dem Bereich für das geladene Video wird sich auch ein Button zum Laden des Videos befinden.

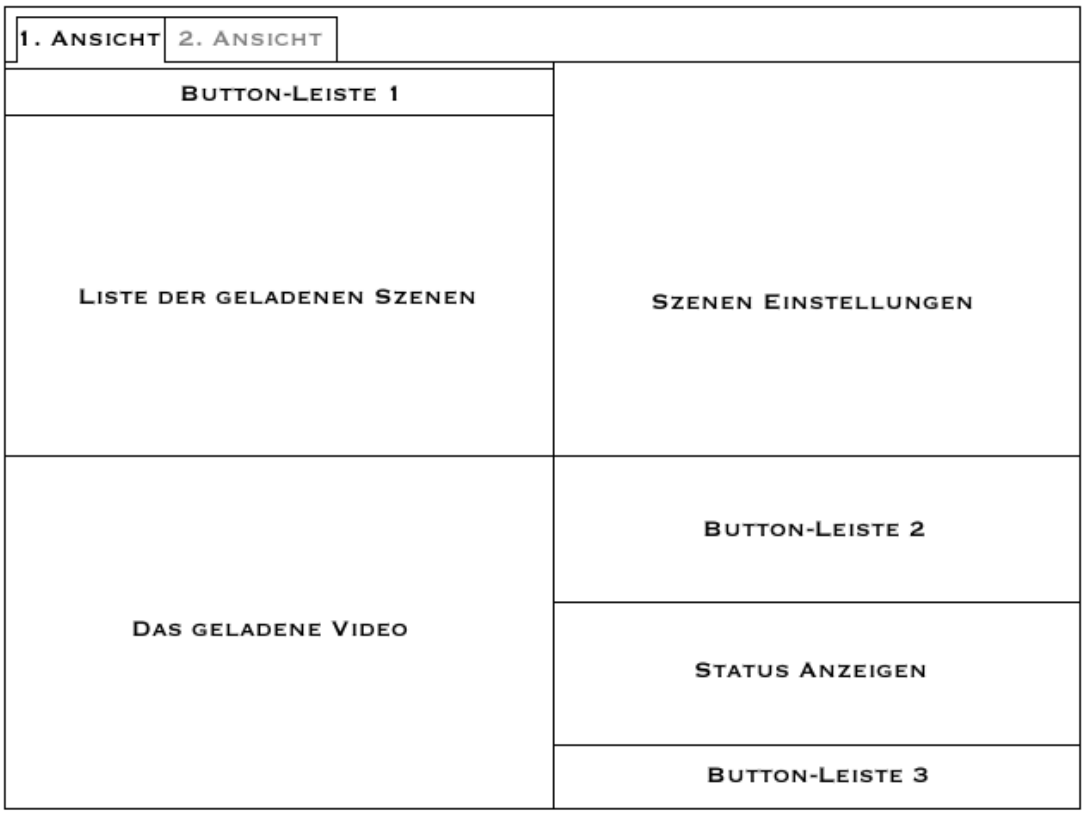


Abbildung 4.3.: GUI - Der erster Tab.

Die Button-Leiste 1 soll sich mit der Auswahl der Szenen beschäftigen. Das bedeutet, zu dieser Leiste gehören z. B. Buttons zum Auswählen und Löschen einer Szene. Außerdem soll ein Button für das Zurücksetzen der ganzen Liste sorgen. Ebenfalls wird sich hier der Button, mit dem man zu den OSC-Einstellungen gelangt, befinden. Über diesen würde der User in eine externe Ansicht gelangen. Von hier aus bekommt der User die Möglichkeit, die Struktur der OSC-Nachrichten oder die IP-Adresse und den Port der WFS-Anlage zu verändern. In die Button-Leiste 2 sollen sich die Aktionsbuttons für das Speichern, das Laden und das Ausführen der Vidaer-Dateien verlagern. In der untersten Button-Leiste 3 werden sich zwei Buttons zum Starten und zum Stoppen des Versand-Vorgangs und ein Button zum Verschicken der OSC-Nachrichten an den Empfänger befinden. Somit kann man alle Objekte der Szenen starten, welche ausgewählt worden sind.
Die zweite Ansicht (s. Abbildung 4.4) dient dem detaillierten Betrachten der Szene. Die Ansicht *Liste der Objekte* dient dazu, alle Objekte der in Ansicht 1 ausgewählten Szene aufzulisten. In der *Liste der Position* sollen alle Zeitpunkte mit den dazugehörigen Koordinaten zur Auswahl

bereitgestellt werden. Das *Fenster für Befehle* ist eine Ansicht, in welche die OSC-Befehle über die Buttons hinzugefügt und hinterher in einer anderen Software bearbeitet werden können.

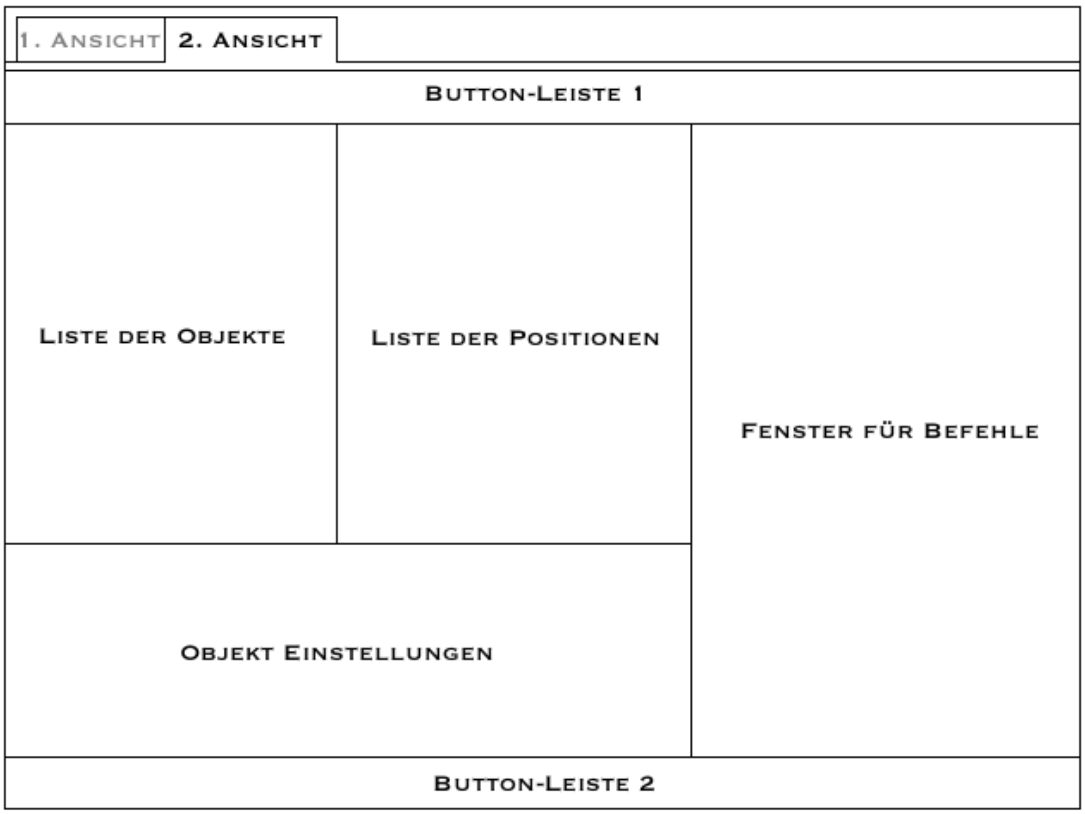


Abbildung 4.4.: GUI - Die zweite Ansicht .

Im Bereich *Objekt Einstellungen* kann man die Einstellungen für das ausgewählte Objekt der Liste verändern. Zu diesen Einstellungen gehört das Verschieben oder das Drehen des Objektes in Uhrzeigerrichtung. Außerdem enthält diese Ansicht zwei Bedienungsleisten mit Funktionen, welche im Folgenden genauer beschrieben werden. Die obere Leiste hat weitere Elemente:

- Button zum Auswählen aller Koordinaten des Objektes
- Button zum Abwählen aller Koordinaten des Objektes
- Feld zum Eingeben einer Zahl als Quellen-ID
- Button zum Bestätigen der Quellen-ID
- Button zum Einfügen des Aktivierungs-Befehls

- Button zum Einfügen des Positions-Befehls

In der unteren Leiste befinden sich die Elemente:

- Button zum Starten der Szene
- Button zum Speichern des Textfeldes als eine SuperCollider-Datei
- Button zum Leeren des Textfeldes
- Button zum Zurücksetzen aller Parameter

5. Implementierung der Software

In diesem Kapitel wird anhand des aufgestellten Konzeptes und mit den Erkenntnissen der durchgeführten Analyse die Softwareimplementierung vorgestellt. Das Kapitel beschreibt die Implementierung von dem entwickelten Programm Vidaer. Der Name der Software ist eine Mischung aus dem Wort „Video“ und dem Format „.dae“. Es wird auf die relevantesten Klassen der Software eingegangen. Als erstes werden die Klassen des *fileStructure*-packages näher veranschaulicht. Als Nächstes wird die *ObjectDetector*-Klasse erläutert, welche für die Analyse der .dae-Datei existiert. Folgend wird der Entwurf der Speicherfunktionen der Software präsentiert. Im Anschluss wird der Versand-Vorgang der OSC-Nachrichten angeschnitten. Eine Übersicht aller Klassen als Klassendiagramm befindet sich in Anhang B.

5.1. Animationsobjekte als Klassen

Ein Animationsobjekt einer Szene ist ein Objekt mit einem Namen und einer Source ID, welche für die Verknüpfung mit einer WFS Anlage benötigt wird. In den definierten Zeitpunkten befindet sich dieses Objekt an festgesetzten Koordinaten. Diese Informationen können aus einer .dae-Datei exportiert werden. Die Methode, welche sich diesem Export-Prozess widmet, ist in Kapitel 5.2 beschrieben. Die Zuweisung einer Source-ID erfolgt durch den User. In diesem Kapitel wird die Struktur der Speicherung der Animationsobjekte in der Software erläutert. Zur Übersicht ist in der Abbildung 5.1 der hierarchische Überblick über das *fileStructure*-package aus dem Konzept gegeben. Im Folgenden werden die Klassen fileDispatcher, AnimationFile und AnimStruct näher erläutert. Auf die Klassen Animation und AnimationData wird nicht genauer eingegangen, da diese nur zur Datenführung dienen.

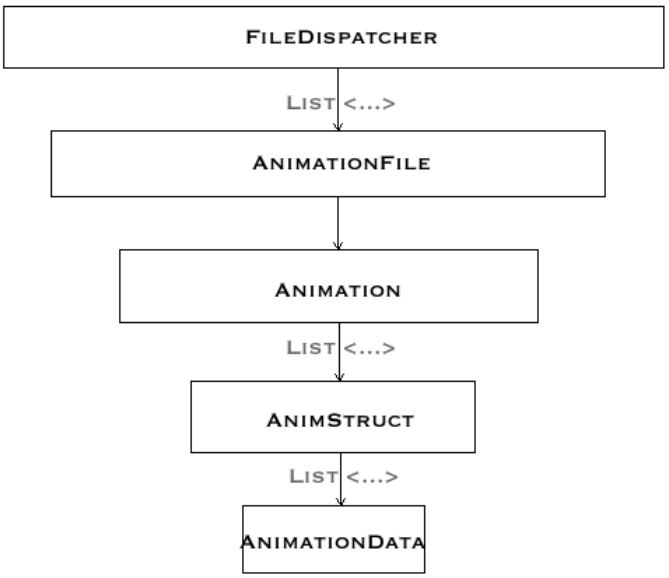
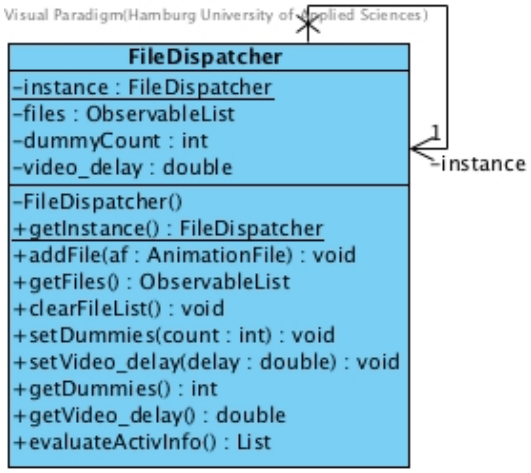


Abbildung 5.1.: Hierarchischer Überblick über das *fileStructure*-package.

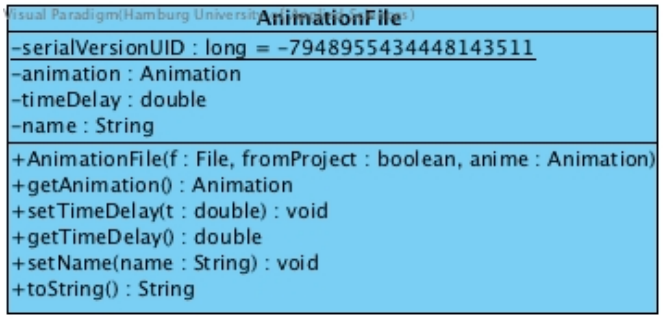
FileDispatcher



Um alle Objekte der Szene im Blick zu behalten, existiert die Klasse *FileDispatcher*. Die Klasse ist als eine Singleton-Klasse implementiert und enthält eine private Liste *files* mit den *AnimationFile*-Objekten. Außerdem hat *FileDispatcher* eine private Variable *dummyCount*, welche bei der Initialisierung auf 0 gesetzt wird. *dummyCount* existiert für die Erstellung der Sound-Quellen, die keinem der Animationsobjekte zugewiesen werden sollen. Diese Quellen können zum Beispiel für die Background-Spuren genutzt werden. Die *evaluateActivInfo*-Methode der

FileDispatcher-Klasse kürzt die gesamte Liste der Animationsobjekte auf eine Liste der *AnimStruct*-Objekte, welche für den Versand ausgewählt worden sind.

AnimationFile



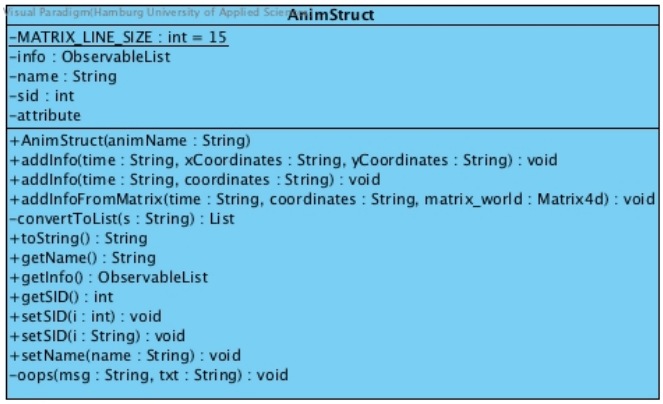
In der Klasse *AnimationFile* hat man die Möglichkeit, ein *Animation*-Objekt als ein *File*-Objekt zu erzeugen, da *AnimationFile* von der Klasse *File* erbt. Mit dem Aufruf des Konstruktors wird ein Objekt der *ObjectDetector*-Klasse (s. Kapitel 5.2) erstellt, wobei eine Unterscheidung getroffen wird. Je nachdem, ob der Aufruf des Konstruktors bei dem Parsen einer .dae-Datei oder einer .vidap-Datei (Vidaer Projekt Datei) geschieht, wird der dritte Parameter benötigt (s. Listing 5.1). Somit wird die private Variable *animation* initialisiert.

```
1   public AnimationFile(File f, boolean fromProject, Animation
        anime) {
2       super(f.getPath());
3       if (!fromProject) {
4           try {
5               ObjectDetector od = new ObjectDetector(f);
6               animation = od.searchXML();
7               name = f.getName();
8           } catch (SAXException | IOException |
                ParserConfigurationException ex) {
9               Logger.getLogger(AnimationFile.class.getName()).log(
                    Level.SEVERE, null, ex);
10          }
11      } else {
12          animation = anime;
13      }
14  }
```

Listing 5.1: Konstruktor der Klasse *AnimationFile*

Außerdem verwaltet *AnimationFile* die zeitliche Verzögerung der Animationsobjekte. Über die Methode *setTimeDelay()* kann man alle Zeitpunkte aller *AnimStruct*-Objekte um den gewünschten Betrag zeitlich verschieben. Die Methode arbeitet mit den Parametern größer oder gleich null. Somit werden die Zeitpunkte auf einen späteren Zeitpunkt gesetzt oder auf den ursprünglichen Wert zurückgesetzt.

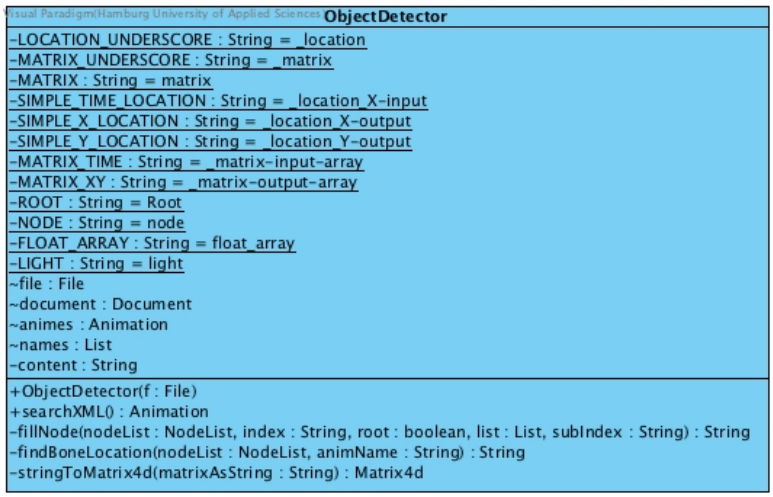
AnimStruct



Die Klasse *AnimStruct* beschreibt ein Animationsobjekt. Ein Objekt der Klasse *AnimStruct* hat einen Namen *name*, eine Source-ID *sid* und eine Liste *info*, welche die Zeitpunkte als *AnimationData*-Objekte enthält. Bei dem Aufruf des Konstruktors wird lediglich der Name der Animation gefragt, die Source-ID ist zu diesem Punkt auf -1 gesetzt und die Liste *info* ist leer. Die Vergabe einer Source-ID ist dem User überlassen. Der Bereich der Eingabe gehört zu der Menge $[-1, 63]$. Die Software wertet die Eingabe einer negativen Zahl als das Abwählen des Animationsobjektes. Wenn also ein Animationsobjekt eine negative Source-ID hat, wird dieser im Versandvorgang nicht berücksichtigt. Außerdem ist dem User nicht unterbunden, mehreren *AnimStruct*-Objekten die selbe Source-ID zu vergeben. Der Grund dafür ist die Möglichkeit der mehrfachen Benutzung der Sound-Spur. Dadurch könnten zwei verschiedene Objekte gemeinsam eine Audiospur nutzen. Um die besten Ergebnisse zu erzielen, muss durch den User sichergestellt werden, dass diese Sound-Spuren nicht zu der selben Zeit angeordnet sind. Die Liste *info* wird von der Software gefüllt. Es gibt drei Funktionen, mit deren Hilfe die Liste eines *AnimStruct*-Objektes die *AnimationData*-Objekte zugewiesen bekommt. Die Erstellung eines neuen *AnimationData*-Objektes übernehmen die drei *addInfo()*-Methoden. Die *addInfo*-Methoden werden für das Erstellen der Informationen eines Meshs genutzt. Die *addInfoFromMatrix(String, String, Matrix4d)*-Methode füllt die Liste *info* des *AnimStruct*-Objektes, welches einen Skeleton (oder ein Bone) repräsentiert. Die Methode erwartet als Parameter die Zeiten und die Koordinaten jeweils als String-Objekt. Der String der Koordinaten entspricht

einer 4×4-Translationmatrix (s. Kapitel 2.2). Außerdem benötigt die Methode als dritten Parameter eine Matrix vom Typ *Matrix4d*. Diese soll die globale Matrix darstellen. In der Methode erfolgt die Umrechnung der Objektkoordinaten in die Raumkoordinaten durch die Multiplikation mit der globalen Matrix.

5.2. Objektdetektor der Software



Die Software soll die Animationsobjekte aus einer .dae-Datei herausfiltern. Dafür wird ein Parser benötigt, welcher diese Funktion übernimmt. Da eine .dae-Datei eine XML-Struktur hat, müssen die benötigten Tags festgelegt werden. Danach wird nach diesen Tags in der Szenen-Datei gesucht. Die Informationen der gesuchten Tags werden in den entsprechenden Objekten gespeichert.
Für diesen Zweck existiert die Klasse *ObjectDetector.java*. Diese Klasse enthält mehrere final-Strings, welche für die Suche nach Tags nötig sind:

LOCATION_UNDERSCORE, MATRIX_UNDERSCORE werden als Index für die Identifizierung eines Tags mit Koordinaten-Informationen genutzt.

MATRIX wird benötigt um die globale Position zu identifizieren.

SIMPLE_TIME_LOCATION, SIMPLE_X_LOCATION, SIMPLE_Y_LOCATION sind die Subindizes. Wenn ein Tag mit der Bezeichnung *_location* gefunden wird, müssen weitere Subtags untersucht werden.

MATRIX_TIME, MATRIX_XY stehen für Indizes zur Findung der Zeit- und Koordinaten-Informationen innerhalb der Mesh-Daten.

ROOT steht für ein festgelegtes Merkmal zur Identifizierung eines Bones, der betrachtet werden soll.

NODE steht für ein Tag der .dae-Szene, in welchem die globalen Positionen enthalten sind.

FLOAT_ARRAY wird benötigt, um das Attribut eines Tags zu finden, welche die Zahlen enthält. Je nachdem sind es Zeiten oder die Koordinaten des entsprechenden Animationsobjektes.

LIGHT ist das Merkmal zur Identifizierung eines Lichtobjektes.

Der Konstruktor der Klasse *ObjectDetector* benötigt ein Objekt vom Typ *File*. Die Richtigkeit des Formats der Datei ist vor dem Aufrufen des Konstruktors zu überprüfen. Hier wird die Datei in einer lokalen Variable *file* gespeichert und die anderen Variablen werden initialisiert. Nach dem Erstellen eines *ObjectDetector*-Objektes steht die Methode *searchXML()* zur Verfügung. Diese Methode sorgt für das Füllen der Objekte der Animation-Klassen mit den entsprechenden Werten. Die komplette Methode ist in dem Listing 5.2 aufgelistet.

```
1   public Animation searchXML() throws SAXException, IOException,
        ParserConfigurationException {
2
3       NodeList nodeList = document.getDocumentElement().
            getChildNodes();
4       AnimStruct animation;
5       //create temporary lists
6       List<String> tempT = new ArrayList<>();
7       List<String> tempX = new ArrayList<>();
8       List<String> tempY = new ArrayList<>();
9       List<String> tempTime = new ArrayList<>();
10      List<String> tempCoord = new ArrayList<>();
11
12      //fill the temporary lists with information from meshs
13      fillNode(nodeList, SIMPLE_TIME_LOCATION, false, tempT,
            LOCATION_UNDERSCORE);
14      fillNode(nodeList, SIMPLE_X_LOCATION, false, tempX,
            LOCATION_UNDERSCORE);
15      fillNode(nodeList, SIMPLE_Y_LOCATION, false, tempY,
            LOCATION_UNDERSCORE);
16      //fill the temporary lists with information from bones
17      fillNode(nodeList, MATRIX_TIME, true, tempTime,
            MATRIX_UNDERSCORE);
18      fillNode(nodeList, MATRIX_XY, true, tempCoord,
            MATRIX_UNDERSCORE);
```

```
19
20          //create AnimStruct-objects from temporary lists with mesh
                informations
21          for (int i = 0; i < tempT.size(); i++) {
22              animation = new AnimStruct(tempT.get(i));
23              if (tempY.size() == tempT.size()) {
24                  animation.addInfo(tempT.get(i + 1), tempX.get(i + 1)
                        , tempY.get(i + 1));
25                  animes.addAnime(animation);
26                  if (!names.contains(animation.getName())) {
27                      names.add(animation.getName());
28                  }
29              }
30              i++;
31          }
32
33          //create AnimStruct-objects from temporary lists with bone
                informations
34          for (int i = 0; i < tempTime.size(); i++) {
35              animation = new AnimStruct(tempTime.get(i));
36              findBoneLocation(nodeList, tempCoord.get(i));
37              animation.addInfoFromMatrix(tempTime.get(i + 1),
                    tempCoord.get(i + 1), stringToMatrix4d(content));
38              animes.addAnime(animation);
39              if (!names.contains(animation.getName())) {
40                  names.add(animation.getName());
41              }
42              i++;
43          }
44          return animes;
45      }
```

Listing 5.2: *searchXML()*-Methode der Klasse *ObjectDetector*

Am Anfang der Methode werden die temporären Listen erstellt, in welche die Informationen vorübergehend gespeichert werden sollen. Danach folgen die Aufrufe der Methode *fillNode()*, welche für das Finden der Informationen in den Tags verantwortlich ist. Am Ende werden aus den temporären Listen die Animation-Objekte erzeugt. Dafür werden zwei for-Schleifen benutzt. Eine for-Schleife sorgt für das Erstellen der Objekte aus den einfachen Meshs. Die zweite for-Schleife erstellt die Animation-Objekte aus den Listen, welche mit den Informationen

der Skeletons gefüllt sind. Beide for-Schleifen haben die Länge der entsprechenden Zeit-Listen als Grenze, weil jeder Zeitpunkt eigene Koordinaten haben muss.

5.3. Die Speicherfunktionen der Software

In der entwickelten Software Vidaer existieren drei Speichermöglichkeiten: die Speicherung des Projektes, die Speicherung des Versand-Vorgangs und die Speicherung benutzerdefinierter Befehle als SuperCollider-Datei (vgl. SuperCollider, a). Im Weiteren findet eine ausführliche Übersicht der Implementierung dieser drei Funktionen statt.

.vida-Format

Es wurde ein eigenes Format für das Speichern der in der Software ausführbaren Dateien implementiert. Eine .vida-Datei hat eine XML-Struktur und speichert nur die ausgewählten Objekte mit den ausgewählten Zeitpunkten. Somit wird der gesamte Versand-Vorgang gespeichert und kann zum späteren Zeitpunkt aus Vidaer abgespielt werden. Eine .vida-Datei ist ausschließlich zum Abspielen gedacht und nicht für die Weiterverarbeitung geeignet. In dem Listing 5.3 betrachtet man eine Beispieldatei im .vida-Format.

```
1  <?xml version="1.0" encoding="UTF-8" standalone="no"?>
2  <vidae>
3      <wfs_settings>
4          <ip>192.168.14.100</ip>
5          <port>58100</port>
6          <dummy>0</dummy>
7          <ardour_port>3819</ardour_port>
8      </wfs_settings>
9      <osc_commands>
10         <activate>/WONDER/source/activate</activate>
11         <deactivate>/WONDER/source/deactivate</deactivate>
12         <position>/WONDER/source/position</position>
13     </osc_commands>
14     <animations>
15         <anim_object name="Suzanne" sourceID="2">
16             <time>0.04 2.5 5.0 </time>
17             <coordinates>-2.77 -0.12 1.77 -0.01 -0.5 -4.71 </
                   coordinates>
18         </anim_object>
```

```
19      </animations>
20  </vidae>
```

Listing 5.3: Eine .vida-Datei

In dieser Struktur ist es möglich, alle Einstellungen des Users zu speichern, welche für den Versand-Vorgang nötig sind. Am Anfang befinden sich die WFS-Frontend-PC Informationen wie die IP-Adresse, der Port, die Anzahl der Dummies und der Ardour-Port (nähere Informationen dazu im Kapitel 5.4). Weiter folgt das Tag *osc_commands*, welches die eingestellte OSC-Struktur der Nachrichten beschreibt. Am Ende kommt das *animations*-Tag. Dieses enthält alle Objekte, welche in der Software mit einer Source-ID vermerkt worden sind. Es wird der Name des Animationsobjektes und die Source-ID als Attribute des Tags übergeben. In dem Beispiel von Listing 5.3 hat das Animationsobjekt den Namen „Suzanne“ und die Source-ID 2. *time*- und *coordinates*-Tags sind die childnodes des *animations*-Tags und enthalten entsprechend die Zeiten und die Koordinaten, welche der User zum Verschicken ausgewählt hat. *coordinates*-Tag beinhaltet die x- und die y-Koordinaten. Zum Beispiel wurden in dem Listing 5.3 bei dem Animationsobjekte „Suzanne“ drei Punkte ausgewählt: (-2.77,-0.12), (1.77,-0.01) und (-0.5,-4.71).

.vidap-Format

Vidaer unterstützt die Speicherung des Projektes als eine .vidap-Datei. Genau wie .vida ist .vidap eine XML-basierte Datei. Im Vergleich zu dem .vida-Format werden hier alle Objekte gespeichert. Das bedeutet, alle geladenen Szenen werden mit allen Animationsobjekten exportiert. Es werden die vom User ausgewählten Zeitpunkte der Objekte und die ursprünglichen Zeiten notiert. Somit können alle Objekte nach dem Importieren einer Projekt-Datei wiedergefunden werden. Das Listing 5.4 präsentiert ein Projekt mit zwei geladenen Szenen. Als Szenen wurde die selbe .dae-Datei doppelt geladen und verschiedene Zeitpunkte wurden ausgewählt.

```
1  <?xml version="1.0" encoding="UTF-8" standalone="no"?>
2  <project_vidaer>
3      <default_set>
4          <wfs_ip>192.168.14.100</wfs_ip>
5          <wfs_port>58100</wfs_port>
6          <ardour_port>3819</ardour_port>
7          <dummy>0</dummy>
8          <set_video_time>0.0</set_video_time>
9      </default_set>
```

```
10      <osc_set>
11          <activate>/WONDER/source/activate</activate>
12          <deactivate>/WONDER/source/deactivate</deactivate>
13          <position>/WONDER/source/position</position>
14      </osc_set>
15      <files>
16          <anim_file name="test.dae">
17              <anim_object name="Suzanne">
18                  <sid>-1</sid>
19                  <default_time>0.04 2.5 5.0 </default_time>
20                  <set_time>0.04 2.5 5.0 </set_time>
21                  <coordinates>-2.77 -0.12 1.77 -0.01 -0.5 -4.71 </
                        coordinates>
22                  <selected>true false true </selected>
23              </anim_object>
24          </anim_file>
25          <anim_file name="test.dae">
26              <anim_object name="Suzanne">
27                  <sid>-1</sid>
28                  <default_time>0.04 2.5 5.0 </default_time>
29                  <set_time>5.04 7.5 10.0 </set_time>
30                  <coordinates>-3.12 1.77 -3.01 2.77 -7.71 0.5 </
                        coordinates>
31                  <selected>false true false </selected>
32              </anim_object>
33          </anim_file>
34      </files>
35  </project_vidaer>
```

Listing 5.4: Eine .vidap-Datei

Auf den ersten Blick sehen Listing 5.3 und Listing 5.4 ziemlich gleich aus. Aber eine .vidap-Datei enthält mehr Tags als eine .vida-Tag. Das liegt daran, dass die Struktur einiger Tags ein wenig ausführlicher ist. Somit somit ist das Tag *anim_objekt* dem *anim_file*-Tag untergeordnet , welches eine Szene darstellt. Außerdem beinhaltet ein *anim_object*-Tag childnodes wie *default_time*, *setted_time* und *selected*. Ein *selected*-Tag entspricht einem boolean-Array, in welchem die Auswahl der entsprechenden Zeitpunkte vermerkt ist. Die *default_time*- und ein *set_time*-Tags bestehen aus den Zeitpunkten. Der Unterschied besteht darin, dass das *default_time*-Tag die ursprünglichen Zeitpunkte festhält und *setted_time*-Tag die Zeitpunkte inklusive den zeitlichen Versatz beinhaltet. In dem Listing 5.4 hat die erste „test.dae“-Szene

keinen Delay zugewiesen bekommen. Das resultiert aus dem Vergleich der Tags in den Zeilen 19 und 20. Beide Tags sind gleich. Im Gegensatz dazu wurde die zweite „test.dae“-Szene um 5 Sekunden später versetzt. Die Zeit wird in Millisekunden ausgegeben.

Das Speichern als .scd-Datei

Ein zusätzliches Feature ist der Export bestimmter OSC-Befehle in eine .scd-Datei. Die Dateien in diesem Format werden aus dem SuperCollider (vgl. SuperCollider, a) ausgeführt. Über drei Buttons (s. Abbildung 6.3) hat der User die Möglichkeit die OSC-Befehle *activate* und *position* in ein integriertes Textfeld hinzuzufügen.

Abbildung 5.2.: Die Buttons zum Hinzufügen der OSC-Befehle. Ein Ausschnitt aus Vidaer.

Über den Button *Add activate* kann man ein OSC-activate-Befehl in das Textfeld einfügen. Wurde kein Objekt aus der Animationsobjektenliste ausgewählt oder verfügt das ausgewählte Animationsobjekt noch keine Source-ID, wird die Eingabe ignoriert und der User wird über ein Pop-up-Fenster benachrichtigt. Eine ähnliche Reaktion ergibt das Drücken auf die Buttons mit den Positionsbefehlen. Über *Add selected position* kann nur die ausgewählte Position dem Textfeld hinzugefügt werden. Im Vergleich dazu fügt der Befehl *Add selected positions* alle mit dem Häkchen markierten Positionen dem Text hinzu. Dabei werden die zeitlichen Abstände zwischen den Zeitpunkten berücksichtigt. Das Textfeld bleibt für alle Szenen gleichgefüllt, was gemischte Eingaben der Positionierungsbefehle bietet. Wenn alle nötigen Daten exportiert wurden, kann man die Liste der Befehle aus dem Textfeld als eine .scd-Datei speichern.

5.4. Das package *senderFamily*

Nachdem die gewünschten Animationsobjekte der Szenen durch den User ausgewählt worden sind, besteht die Möglichkeit, den Versand-Vorgang zu starten. Dieser Vorgang entspricht dem Verschicken der OSC-Nachrichten zu den bestimmten Zeitpunkten. Das Versenden der OSC-Nachrichten übernehmen die Threads des packages *senderFamily*. Die Übersicht über den allgemeinen Vorgang kann aus der Abbildung 5.3 entnommen werden. Darauf folgend werden die Klassen ausführlich vorgestellt.

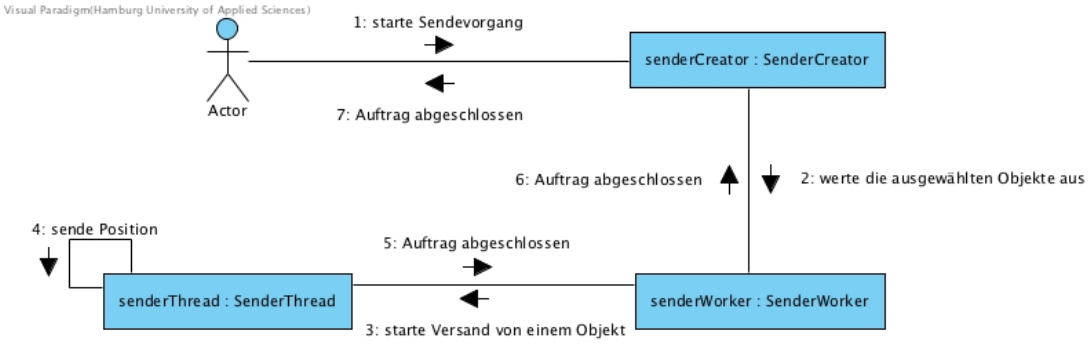
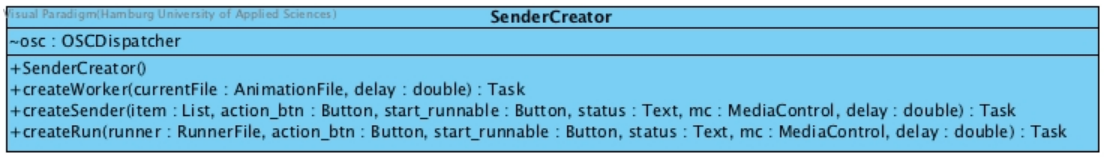


Abbildung 5.3.: Kommunikationsdiagramm der Klassen des *senderFamily*-packages.

Das in der Abbildung 5.3 vorgestellte Diagramm beschreibt das Starten des Sendevorgangs. Der User stößt den Versand über den Button in der GUI an. Somit wird der *senderCreator* erstellt und gestartet. Der *senderCreator* erstellt einen *senderWorker* mit den vorhandenen Informationen. Der *senderWorker* wertet die Informationen aus und erstellt für jedes Objekt einen *senderThread*. In dem Beispiel aus der Abbildung 5.3 soll für ein Objekt der Sendevorgang gestartet werden. Nach dem Erstellen von dem *senderThread* übernimmt der *senderThread* das Versenden der OSC-Nachrichten mit den Positionsdaten des Objektes. Nach der abgeschlossenen Arbeit wird der *senderWorker* über das Ende des Sendevorgangs benachrichtigt. Danach weist der *senderWorker* den *senderCreator* auf die Fertigstellung des Auftrags hin. Der *senderCreator* aktualisiert sein Status und benachrichtigt den User.

SenderCreator



Die Klasse *SenderCreator* übernimmt die Kreation der eigentlichen Sender. Die Sender teilen sich in zwei verschiedene Varianten auf: *SenderWorker* und *SenderThread*. Die Klasse *SenderWorker* baut ein Sendevorgang aus einer .vida-Datei oder aus dem erstellten oder geladenem Projekt auf. Das Verschicken der Informationen eines Animationsobjektes einer Szene des Projektes übernimmt jeweils ein *SenderThread*. Dieser wird in der Klasse *SenderWorker* erstellt. Ein Beispiel ist als Sequenzdiagramm in der Abbildung 5.4 vorgestellt. In diesem Diagramm ist ein Projekt dargestellt, in welchem zwei Animationsobjekte zum Versand ausgewählt worden sind. Der User erstellt ein Objekt der Klasse *SenderCreator* und erstellt einen Sender über die Me-

thode *createWorker()*. Daraufhin erstellt der *SenderCreator* ein Objekt der Klasse *SenderWorker*. Nach dem Starten des *SenderCreator* wird das Starten von dem *SenderWorker* angestoßen. Der *SenderWorker* erstellt und startet eine Anzahl von *SenderThread*s für jedes ausgewählte Animationsobjekt. Jeder *SenderThread* verschickt die Positionsinformationen in einer for-Schleife. Die *run()*-Methode eines *SenderThreads* ist fertig, wenn alle Zeitpunkte verschickt worden sind. Wenn alle *SenderThread*s die Arbeit beendet haben, ist der Versand-Vorgang erfolgreich abgeschlossen und die *run()*-Methode des *SenderWorker*s zu Ende.

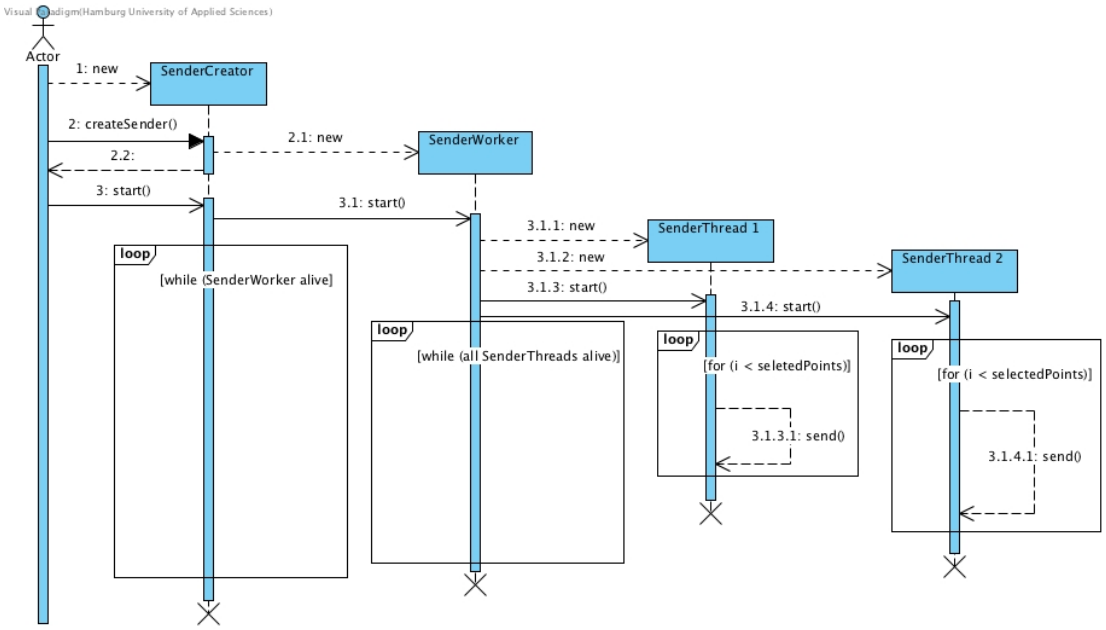
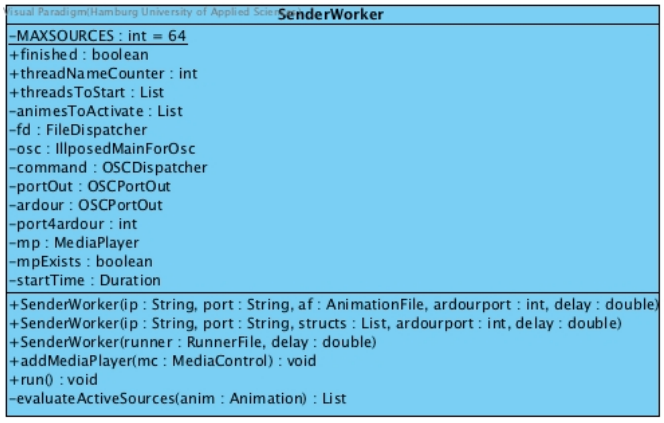


Abbildung 5.4.: Das Sequenzdiagramm eines Versand-Vorgangs.

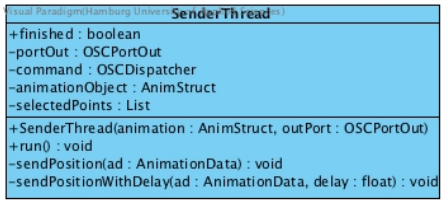
SenderWorker



Die Erstellung eines Objekts der Klasse *SenderWorker* kann über drei Konstruktoren erfolgen. Über die Konstruktoren kann man den Sende-Vorgang auf drei verschiedene Arten In einem Konstruktor muss ein *AnimationFile*-Objekt übergeben werden, in dem zweiten wird eine Liste der *AnimStruct*-Objekte erwartet, der dritte Konstruktor benötigt ein *RunnerFile*-Objekt als Parameter. Anschließend wird in allen Konstruktoren die lokale Variable *animesToActivate* initialisiert. Außerdem werden IP-Adresse und der Port des WFS-Frontend-PCs, der Ardour-Port und das Delay für das Video gefragt. Vidaer enthält die Funktion, mit dem Starten der Bewegungen gleichzeitig die Audiowiedergabe in Ardour zu starten. Hierzu wird der Ardour-Port benötigt. Die IP-Adresse für Ardour wird nicht abgefragt, weil davon auszugehen ist, dass Ardour ebenfalls auf dem WFS-Frontend-PC läuft. Die ersten zwei Konstruktoren bekommen die oben genannten Informationen übergeben. Ein *RunnerFile*-Objekt enthält diese Daten. Daher werden in dem dritten Konstruktor keine weiteren Parameter benötigt. Zusätzlich zum automatischen Start des Sounds, existiert in Vidaer das gleichzeitige Starten des geladenen Videos. Soll der integrierte Mediaplayer nicht mit dem Anfang des Videos starten, wird das nötige Delay in dem Konstruktor übergeben.
Die *run()*-Methode der *SenderWorker*-Klasse arbeitet mit der Liste *animesToActivate*, welche Objekte vom Typ *AnimStruct* benötigt. Daher wird in dem einen Konstruktor das *AnimationFile*-Objekt mit Hilfe von der *evaluateActivSources()*-Methode in eine kompatible Liste umgewandelt. Am Anfang der *run()*-Methode werden alle virtuellen Quellen der WFS-Anlage zurückgesetzt. Danach folgt die Aktivierung aller nötigen Quellen anhand der Source-ID der Animationsobjekte und der Anzahl der dummy Quellen. Diese Aktion erfolgt in einer for-Schleife durch das Versenden einer *activate*-OSC-Nachricht. Gleichzeitig wird in der Schleife die Liste *threadsToStart* gefüllt. Diese wird für das Erstellen eines *SenderThread*s für jedes *AnimStruct*-Objekt aus *animesToActivate* gebraucht. Als Nächstes wird Ardour, das Video und

die *SenderThread*s gestartet.
Ein *SenderWorker* existiert so lange bis der letzte *SenderThread* seine Arbeit beendet hat. Am Ende setzt *SenderWorker* den lokalen *finished*-Flag auf true und stoppt die Wiedergabe des Videos und Ardour.

SenderThread



Der *SenderThread* verwaltet den Versand-Vorgang der Informationen einer Szene und wird durch den *SenderWorker* erzeugt. In dem Konstruktor werden mit Hilfe der übergebenen Parameter die lokalen Variablen initialisiert. Im Anschluss wird anhand der *selected*-Variable jedes *AnimationData*-Objekt aus dem übergebenen *AnimStruct*-Objekt die *selectedPoints*-Liste gefüllt. Die *run()*-Methode besteht aus einer for-Schleife, welche über alle Elemente der *selectedPoints*-Liste iteriert (s. Listing 5.5).

```
1    public void run() {
2        for (int i = 0; i < selectedPoints.size(); i++) {
3            try {
4                if (!isInterrupted()) {
5                    if (selectedPoints.get(i).equals(selectedPoints.
                         get(0))) {
6                        Thread.sleep((long) (selectedPoints.get(i).
                             getTime() * 1000));
7                        sendPosition(selectedPoints.get(i));
8                    } else {
9                        float delta = selectedPoints.get(i).getTime
                             () - selectedPoints.get(i - 1).getTime();
10                       sendPositionWithDelay(selectedPoints.get(i),
                              delta);
11                       Thread.sleep((long) (delta * 1000));
12                   }
13               }
14           } catch (InterruptedException ex) {
15               finished = true;
16               interrupt();
```

```
17                  return;
18              }
19          }
20          finished = true;
21      }
```

Listing 5.5: *run()*-Methode der Klasse *SenderThread*

Das erste Element in der Liste beschreibt die Ausgangsposition des Animationsobjektes. Die Koordinaten werden mit einer Verzögerung verschickt, welche gleich dem ersten Zeitpunkt des Objektes ist. Alle darauf folgenden Koordinaten werden mit Delay verschickt. Der Versand so einer OSC-Nachricht erzeugt einen Sound-Verlauf, vom letzten zum neuen Punkt in der WFS-Anlage. Danach wird eine Zeit lang (gleich Delay *delta*) gewartet, bis die nächsten Koordinaten verschickt werden. Die übergebene Zeit in den Zeilen 6 und 11 in dem Listing 5.5 sind mit 1000 multipliziert, da alle Zeitpunkte der *AnimationData*-Objekte in Millisekunden gespeichert werden und 1000 dem Umrechnungsfaktor entspricht.
Die *run()*-Methode arbeitet mit interrupted exceptions, welche durch den User ausgelöst werden können. In der grafischen Oberfläche des Vidaer gibt es die Möglichkeit, den Versand-Vorgang zu stoppen. Hiermit wird eine InterruptedException ausgelöst, welche in der Zeile 14 abgefangen wird. Somit wäre der Versand-Vorgang zu Ende. Daher wird auch hier das lokale *finished*-Flag auf true gesetzt. In der Abbildung 5.5 ist ein Sequenzdiagramm vorgestellt, wobei der Versand-Vorgang durch den User unterbrochen wird.

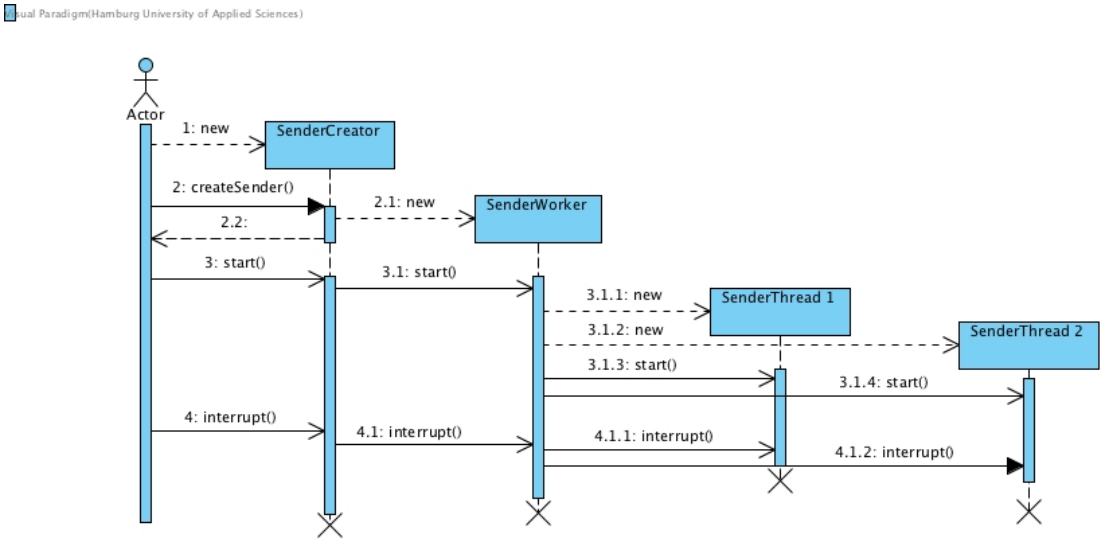


Abbildung 5.5.: Das Sequenzdiagramm eines abgebrochenen Versand-Vorgangs.

6. Testen der Software

In diesem Kapitel wird die entworfene Software evaluiert. Zuerst werden die Evaluierungskriterien festgelegt. Im Folgenden wird die Unterstützung verschiedener Betriebsysteme nach diesen Kriterien getestet. Im Anschluss steht ein Überblick über den Black-Box-Test und den White-Box-Test bereit.

6.1. Evaluierungskriterien

Die Software Vidaer wurde unter Java 8 entwickelt. Somit ist sichergestellt, dass das Programm unter den verschiedenen Betriebsystemen gestartet werden kann. Außerdem wurde die JavaOSC-Library (vgl. Ramakrishnan) genutzt. Daher sind das vorinstallierte Java Runtime Environment 8 und vorhandene JavaOSC-Library (vgl. Ramakrishnan) vorauszusetzen. Mit dieser Annahme soll die Funktionsfähigkeit der Software getestet werden. Dazu werden folgende Punkte überprüft.

- Wird die GUI richtig dargestellt?
- Wurde eine Test-COLLADA-Datei richtig eingelesen?
- Kann der Versand-Vorgang gestartet und abgebrochen werden?
- Stimmt die Bewegung der angehörten Schallquelle mit den Erwartungen überein?
- Werden die Koordinaten bei einer Rotation an die richtige Position überführt?
- Kann eine geladene .mp4-Videodatei richtig dargestellt werden?
- Funktioniert die Speicherung der Projekt- und Runner-Dateien?
- Wurden die Projekt- und Runner-Datei richtig erstellt?
- Können die Projekt- und Runner-Dateien korrekt eingelesen werden?

Für eine Test-Szene wurde eine Blender-Szene entworfen. In dieser wandert ein Affenkopf (erstellt in Blender (vgl. Blender Foundation, c) als ein Mesh) zuerst von links nach rechts und im Anschluss nach hinten (s. Abbildungen 6.1 und 6.2). Anschliessend wurde die Szene als .dae-Datei **test.dae** exportiert. Da diese Test-Szene keine komplexe Struktur aufweist, wurden außerdem auch die Szenen des „Big Buck Bunny"-Projektes (vgl. Project Peach, b) beim Testen eingesetzt.
Der Test soll unter realistischen Bedingungen im WFS-Labor der HAW Hamburg durchgeführt werden. Ein schematischer Versuchsaufbau ist in den Abbildungen 6.1 und 6.2 dargestellt. Auf diesen Abbildungen ist das WFS-Labor der HAW Hamburg schematisch zu betrachten. Der schwarze Punkt in der Mitte der Abbildungen soll den User darstellen. Sein Blickwinkel zeigt in eine Richtung, welche durch einen Strich vor dem User abgebildet ist. Somit schaut der User in dem linken Teil der Abbildung 6.1 zum Nullpunkt des Koordinatensystems der WFS-Anlage und der Fernseher befindet sich rechts von dem User. In dem rechten Teil der Abbildung 6.1 steht der User mit dem Rücken zu dem Nullpunkt des Koordinatensystems der WFS-Anlage und der Fernseher ist links von dem User. In der Abbildung 6.2 schaut der User zum Fernseher. Die Idee hinter diesen verschiedenen Drehungen besteht darin, die in der entwickelten Software vergebene Sound-Position korrekt im Raum zu lokalisieren. Der Sound-Verlauf wird von dem Zeitpunkt T1 zu dem Zeitpunkt T3 über den Zeitpunkt T2 erwartet. Die Anfangspositionen der Punkte sind folgende:

- T1: 0.04 Sekunden. Koordinaten: (-2.77, -0.12)
- T2: 2.5 Sekunden. Koordinaten: (1.77, -0.01)
- T3: 5.0 Sekunden. Koordinaten: (-0.5, -4.71)

Diese Aufstellung ist dem linken Teil der Abbildung 6.1 zu erwarten. Dem entsprechend sind in der rechten Seite der Abbildung 6.1 die Koordinaten um 180 Grad und in der Abbildung 6.2 um 90 Grad im Uhrzeigersinn gedreht.

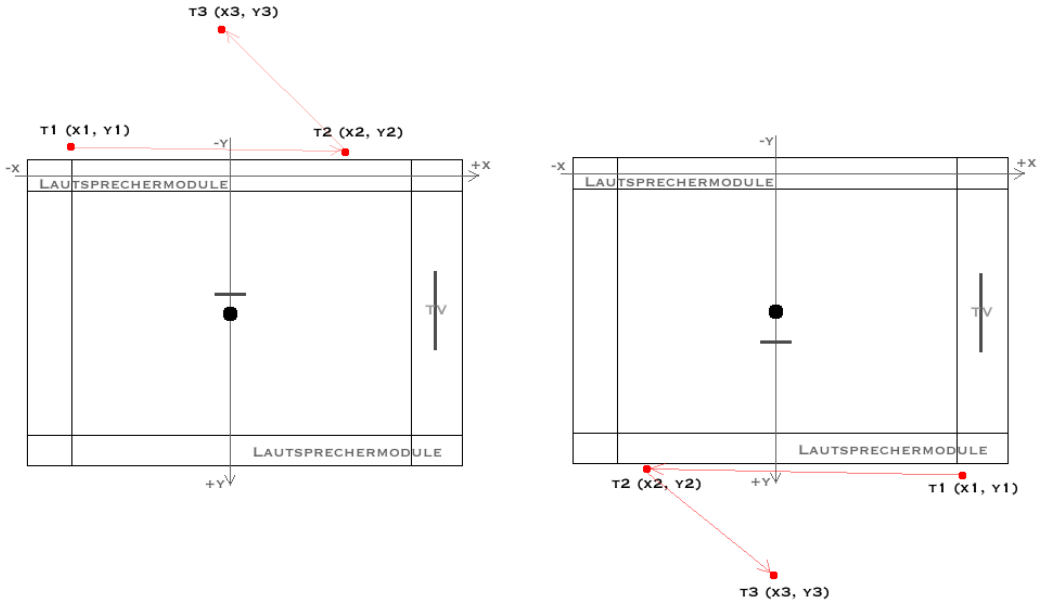


Abbildung 6.1.: Der Aufbau der Test-Versuche 1. Links: der Verlauf von dem Objekt (rot) unverändert. Rechts: der Verlauf von dem Objekt (rot) rotiert um 180 Grad.

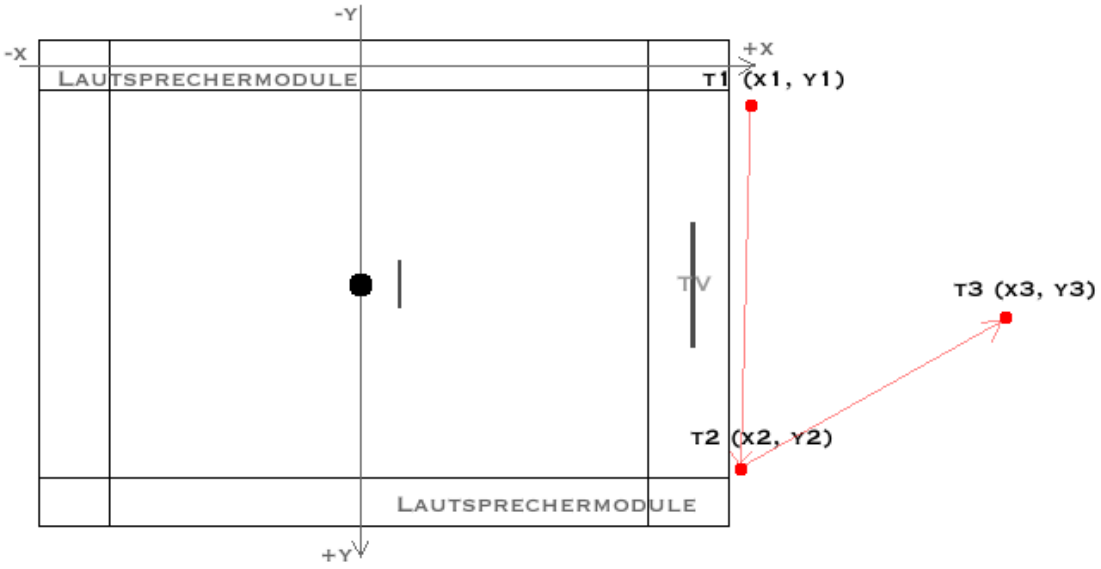


Abbildung 6.2.: Der Aufbau der Test-Versuche 2. Die Koordinaten wurden um 90 Grad im Uhrzeigersinn gedreht.

6.2. Evaluierung unter diversen Betriebsystemen

Nach den aufgestellten Testkriterien wurden drei Rechner mit verschiedenen Betriebsystemen ausgewählt.

Rechner 1:

- Betriebsystem: Windows 7 Professional
- Prozessor: Intel(R) Xeon(R) CPU E5-2630 v2 2,6 GHz 2,6 GHz
- Arbeitsspeicher: 32 GB RAM
- Festplatte: NTFS 920 GB
- Ausgewähltes Netzwerkinterface: Ethernet

Rechner 2:

- Betriebsystem: OS X El Capitan 10.11.6 Beta
- Prozessor: Intel Core i5 1,3 GHz
- Arbeitsspeicher: 4 GB
- Festplatte: SSD 128 GB
- Ausgewähltes Netzwerkinterface: WLAN

Rechner 3:

- Betriebsystem: Linux Mint 17.3 Cinnamon 64-Bit
- Prozessor: Intel Core i5 2,5 GHz
- Arbeitsspeicher: 8 GB
- Festplatte: SSD 80 GB
- Ausgewähltes Netzwerkinterface: WLAN

Jeder der verwendeten Rechner konnte den Test nach den Evaluierungskriterien erfolgreich abschließen. Die Versuche wurden wie in den Abbildungen 6.1 und 6.2 aufgebaut. Die Tabelle 6.1 zeigt die getesteten Anforderungen.

Anforderung	**Rechner 1 Windows 7**	**Rechner 2 OS X**	**Rechner 3 Linux Mint**
Wird die GUI richtig dargestellt?	✔	✔	✔
Wurde eine Test-COLLADA-Datei richtig eingelesen?	✔	✔	✔
Kann der Versand-Vorgang gestartet und abgebrochen werden?	✔	✔	✔
Stimmt die Bewegung der angehörten Schallquelle mit den Erwartungen überein?	✔	✔	✔
Werden die Koordinaten bei einer Rotation in die richtige Position überführt?	✔	✔	✔
Kann eine geladene .mp4-Videodatei richtig dargestellt werden?	✔	✔	✔
Funktioniert die Speicherung der Projekt- und Runner-Dateien?	✔	✔	✔
Wurden die Projekt- und Runner-Datei richtig erstellt?	✔	✔	✔
Können die Projekt- und Runner-Dateien korrekt eingelesen werden?	✔	✔	✔

Tabelle 6.1.: Übersicht der getesteten Anforderungen.

6.3. Validierung der Software

Außer der Evaluierung der Funktionalität von Vidaer unter diversen Betriebsystemen wurde die Software durch die Software-Tests validiert. So konnte die entworfene Software mit den aufgestellten Anforderungen (s. 3.1) abgeglichen werden. Hierfür wurde die Funktionalität der relevanten Methoden in einem Black-Box-Test überprüft. Außerdem wurde ein detaillierter White-Box-Test durchgeführt.

6.3.1. Black-Box-Test der Software

Der Black-Box-Test wurde mit Hilfe von JUnit 4 (vgl. JUnit) erstellt. Es wurden vier Klassen dem Test unterworfen: *MathCalculator.java*, *ObjectDetector.java*, *ProjectFile.java* und *XMLController.java*. Jede dieser Klassen liest eine Test-Datei ein, welche auf der **test.dae** basiert. Die *MathCalculatorTest*-Klasse überprüft die korrekte Umrechnung der Verschiebung und der Ro-

tation der Koordinaten. Die *ObjectDetectorTest*-Klasse kontrolliert die Korrektheit der aus der **test.dae** erstellten *fileStructure*-package-Objekten. Die *ProjectFileTest*-Klasse und die *XMLControllerTest*-Klasse überprüfen die geladenen Vidaer-Dateien. Dafür wurden die Dateien 5.3 und 5.4 genutzt. Außerdem wurden in diesen Klassen zwei Dateien eingelesen, in welchen gezielt ein Fehler erzeugt wurde. Die Mängel wurden erfolgreich identifiziert. Somit konnte unter den angegebenen Kriterien bewiesen werden, dass auch im Falle inkorrekter Benutzung die Fehler abgefangen werden konnten.

6.3.2. White-Box-Test der Software

Für einen Test auf Basis der Spezifikation wurde ein Code Coverage mit Hilfe von EclEmma (vgl. Roubtsov) durchgeführt. Der Test wurde mehrmals ausgeführt. Im Laufe der Tests wurde das Refactoring vorgenommen. Somit hat sich das Ergebnis um 20 % verbessert und liegt am Ende bei 96%. Die fehlende 4% weisen auf die nicht aufgerufene Sicherheitsmethoden, welche aber bei der Weiterentwicklung der Software benötigt werden.

Element	Missed Instructions	Cov.	Missed Branches	Cov.	Missed	Cxty	Missed	Lines	Missed	Methods	Missed	Classes
vidaer		96%		83%	76	447	57	1.599	6	228	0	27
Total	286 of 7.434	96%	74 of 438	83%	76	447	57	1.599	6	228	0	27

Abbildung 6.3.: VidaerCode Coverage Ergebnis.

6.4. Zusammenfassung der Ergebnisse

Die entwickelte Software wurde ausführlich getestet. Dazu wurde die Unterstützung der diversen Betriebsysteme überprüft und die Softwaretests wurden in dem WFS-Labor durchgeführt. Im Verlauf konnten einige Gedanken- und Sicherheitslücken durch das Refactoring gefüllt werden. Insgesamt wurden alle Testfälle erfolgreich abgeschlossen.

7. Zusammenfassung und Ausblick

Das Ziel dieser Arbeit war die Erstellung einer Software zur automatischen Verknüpfung der Blender-Animationsobjekte mit den virtuellen Schallquellen einer Wellenfeldsynthese-Anlage. Als Basis wurde das Projekt „Big Buck Bunny“ von Project Peach (vgl. Project Peach, b) verwendet. Als Resultat ist die Software Vidaer entstanden, welche die Funktion der Verknüpfung übernimmt. Der Name Vidaer ist die Verschmelzung von den Wörtern „Video“ und „.dae"(ein Format der COLLADA).
Am Anfang der Arbeit wurden die Grundlagen erläutert, welche für das Verständnis der Arbeit benötigt werden. Dazu zählt das Basiswissen über die Blender-Umgebung, das COLLADA-Format und das Wellenfeldsynthese-Labor der HAW Hamburg. Als Nächstes wurde eine Analyse der Anforderungen durchgeführt. In dieser wurden die erwarteten Vorstellungen geschildert. Um diese Anforderungen zu erfüllen, wurde im weiteren Verlauf eine Analyse der Quelldateien des Projektes „Big Buck Bunny“ (vgl. Project Peach, b) ausgearbeitet. Daraus wurden einige Merkmale der Weiterverarbeitung der Datei klar definiert. Zum Beispiel wurde der XML-Tag einer Blender-Datei für die Darstellung der Koordinaten identifiziert. Aufgrund der Komplexität so einer .blend-Datei wurden außerdem Annahmen getroffen, durch welche die Software präzisiert werden konnte. Ebenfalls wurde die Art der Verbindung mit den Schallquellen der WFS-Anlage beschrieben.
Basierend auf den durchgeführten Analysen ist im nächsten Kapitel das Konzept der Software entstanden. In diesem Teil der Arbeit wurden die geschilderten Anforderungen in einer Arbeitssequenz zusammengefasst. Die Architektur der Software wurde ebenfalls vorgestellt. Im Anschluss folgte das Konzept der GUI.
Die Implementierung der Software befasst sich mit der Entwicklung von Vidaer. In diesem Kapitel wird der Aufbau der Klassen und die Methoden ausführlich vorgestellt. Der Umfang der Implementierung umfasst die relevanten Klassen wie zum Beispiel die Klassen aus dem *fileStructure*-package und *senderFamily*-package. Auf diesen zwei basiert die innere Struktur der Software zur Aufbewahrung der Informationen von Animationsobjekten und zum Versenden der OSC-Nachrichten. Außerdem wurden in diesem Kapitel die Speicherfunktionen der Software erläutert.

Im Anschluss folgt das Kapitel der Evaluierung der Software. Hier werden die durchgeführten Tests beschrieben und dokumentiert. Zum Anfang des Kapitels wurden die Kriterien aufgestellt, nach welchen der Erfolg des Testvorgangs beurteilt werden konnte. Folgend wurde Vidaer nach diesen Kriterien auf drei Rechnern mit verschiedenen Betriebsystemen getestet. Alle Tests wurden erfolgreich beendet. Am Ende folgt die erfolgreiche Validierung der Software durch Softwaretests wie Black-Box- und White-Box-Test.

7.1. Ausblick auf die Erweiterung der Software

Die entworfene Software ist das erste zu diesem Thema entwickelte Programm in dem Wellenfeldsynthese-Labor der HAW Hamburg. Vidaer bietet außer der grundlegenden Funktion der Verknüpfung von Animationsobjekten mit virtuellen Schallquellen auch mehrere Möglichkeiten zur individuellen Anpassung der Parameter. Hierzu könnte die mögliche Weiterentwicklung an der entworfenen Software wie folgt aussehen.

- Vidaer nutzt die Objekt-Koordinaten der Animationsobjekte, ohne die Position der Kamera zu berücksichtigen. Daher ist es notwendig bei der Modellierung einer Szene in Blender die Achsenrichtungen zu beachten. Die Einführung der Umrechnung der Objekt-Koordinaten bezüglich der Kameraposition wäre der nächste Schritt in der Entwicklung der Software.

- Der momentane Aufbau des Labors der Wellenfeldsynthese-Anlage der HAW Hamburg ermöglicht die Bewegung der Schallquellen nur in der horizontalen Ebene. Somit wurde die Höhe in der Implementierung nicht beachtet. Das bedeutet mit der Erweiterung des Labors um die z-Ebene, könnte auch Vidaer um die z-Koordinaten erweitert werden.

- Der aktuelle Stand der Software bietet die Möglichkeit zur Verschiebung oder zur Rotation der Koordinaten in eine beliebige zweidimensionale Richtung. Ein weiteres Feature wäre die Skalierung der Objekt-Koordinaten. Außerdem werden die Startpositionen der Dummy-Quellen auf den Ursprung gelegt. Es wäre flexibler, diese festen Quellen an den beliebigen Raum-Koordinaten platzieren zu können.

- Die Koordinaten der Animationsobjekte in Vidaer hängen mit den hinzugefügten Locations in der Blender-Frameleiste zusammen. Somit ist die Platzierung der virtuellen

Quellen einer WFS-Anlage nur auf diese Koordinaten begrenzt. Für eine eventuelle Änderung nach der Blender-Modellierung wäre das Einfügen der benutzerdefinierten Position einem Objekt der Szene in Vidaer sinnvoll.

7.2. Fazit

Zum Schluss kann gesagt werden, dass das Ziel der Arbeit erreicht werden konnte. Die entworfene Software Vidaer exportiert die Objektkoordinaten einer Blender-Szene in eine benutzerfreundliche Umgebung. In dieser kann der User persönliche Einstellungen treffen und die Animationsobjekte mit den virtuellen Quellen einer WFS-Anlage verbinden. Aus der Vidaer-Umgebung lässt sich der Versand-Vorgang über ein Netzwerk starten, womit die Schallquellen der WFS-Anlage bewegt werden. Zusätzlich dazu kann eine Videodatei geladen werden, um den Vorgang visuell zu verfolgen. Die Sound-Spuren in Ardour (vgl. Ardour) können ebenfalls mit dem Starten des Versand-Vorgangs gestartet werden. Somit betrachtet der User am Ende das geladene Video mit dem dreidimensionalen Sound der WFS-Anlagenumgebung.

A. Benutzerhandbuch

Die Bedienungsanleitung zu dem entwickelten Programm befindet sich in diesem Anhang. Das wurde für den Endnutzer geschrieben und enthält sowohl die komplette Übersicht der Software als auch die schrittweise Anleitung.

Vidaer

Bedienungsanleitung

Ilyuza Mingazova
Haw Hamburg

Contents

1 Einleitung **3**

2 Installation **3**

3 Vorbereitung **3**

4 Übersicht **4**

5 Erste Schritte **7**

1. Einleitung

Das Programm Vidaer wurde für die automatische Verknüpfung von Blender-3D-Animationsobjekten mit den virtuellen Schallquellen einer WFS-Anlage entwickelt. Der Name der Software stammt von "Video" und dem COLLADA-Format[1] ".dae". Mit Hilfe von Vidaer kann man die Animationsobjekte eines 3D-Films an die virtuellen Quellen einer Wellenfeldsynthese-Anlage (abk. WFS-Anlage) über OpenSoundControl-Nachrichten (abk. OSC) übertragen. Entworfen und getestet wurde die Software für die WFS-Anlage der HAW Hamburg.

2. Installation

Vidaer ist ein Java-basiertes Programm und wurde mit Hilfe von Java 8 erstellt. Daher ist sicherzustellen, dass auf dem zu benutzendem Rechner Java-Platform ab der Version 8 installiert ist. AuSSerdem benutzt die Software die JavaOSC[2] Bibliothek von Illposed. Wenn es Probleme bei dem Versand-Vorgang der OpenSoundControl-Nachrichten auftreten, ist es empfohlen, die JavaOSC-Bibliothek zu dem Ordner hinzuzufügen, wo sich Vidaer befindet.

3. Vorbereitung

Vidaer arbeitet mit COLLADA-Dateien[3]. Es ist notwendig die benötigten Szenen als .dae-Datei zu exportieren. Dabei ist es wichtig zu beachten, dass die Objekte mit Skelett einen Hauptknochen haben, der auch ”Root”-Bezeichnung haben muss. Hat ein Skeleton keinen Hauptknochen mit der Bezeichnung, wird dieser von dem Vidaer ignoriert.
Manchmal tritt ein Export-Problem, wobei keine oder nicht alle Daten der Objekte in die neue .dae-Datei mitexportiert werden. Ob das der Fall ist, kann überprüft werden, in dem man die Datei mit einfachem Texteditor öffnet und prüft, ob die Objekte mit den zuvor eingegebenen Namen und der Erweiterung ”_location” existieren. Z.B. wenn ein Objekt ”Cube” heisst, kann man mit der Texteditor-Suche den Ausdruck ”Cube_location_X” finden. Wenn die Suche keine Ergebnisse ergab, müssen die Export-Einstellungen des Animationsprogramms überprüft werden.
Es ist ebenfalls möglich das fertige Video zu laden, um den Fortschritt visuell zu verfolgen. Das Programm unterstützt .mp4-Format. AuSSerdem startet Vidaer die Wiedergabe in Ardour[4], wenn Ardour-Port richtig unter den Vidaer-Einstellungen

[1] https://www.khronos.org/collada/
[2] http://www.illposed.com/software/javaosc.html
[3] https://www.khronos.org/collada/
[4] http://ardour.org

eingegeben wurde. Die IP-Adresse des Rechners, auf welchem Ardour gestartet werden soll, muss dann mit der WFS-IP-Adresse übereinstimmen.

4. Übersicht

Nach dem Öffnen des Programms erscheint das Hauptfenster - Bild 1.

1. Main Page - Das Hauptfenster zur Auswahl der Szenen.
   Export Objects - Das Fenster ist am Anfang deaktiviert.
2. Hier werden die ausgesuchte Szenen dargestellt. Mit dem Doppelklick auf eine Datei oder mit dem Drücken des Buttons "Continue with scene" wird man zu dem Fenster ”Export Objects” umgeleitet.
3. Das geladene Video erscheint hier.
4. Das Feld zum Eingeben des zeitlichen Versatzes der ausgewählten Szene (Eingabe in Sekunden). Das erste Feld versetzt die .dae-Szene, das zweite Feld versetzt den Startpunkt des Videos.
5. Wird in der Szene ein oder mehrere Hintergrund-Sounds benötigt, ist es möglich mit der Eingabe der Anzahl der dummy-Quellen in diesem Feld welche zu erzeugen.

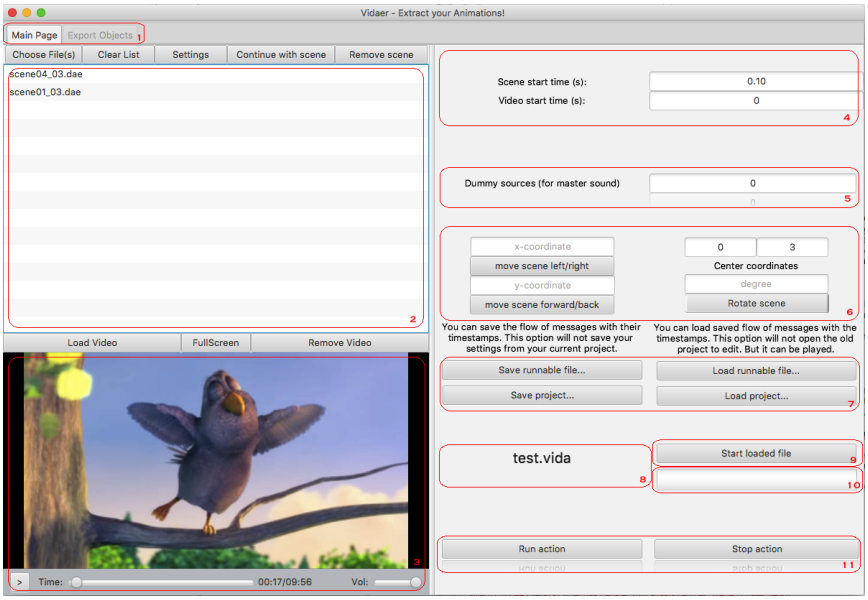


Figure 1: Das Hauptfenster des Programms

6. Muss eine Szene aus der Liste in dem Feld 2 in eine Richtung verschoben oder gedreht werden, kann man dieses hier einstellen. Eine Drehung um einen positiven Winkel bedeutet hier eine Drehung im Uhrzeigersinn. Die Veränderungen werden sich auf alle Objekte der Szene auswirken. Dafür muss natürlich die gewünschte Szene ausgewählt sein.

7. Save runnable file - Somit kann man den Versand-Vorgang als eine .vida-Datei speichern.
   Load runnable file - Ermöglicht das Laden einer .vida-Datei in das Programm. Ob eine Datei geladen ist, kann man im Feld 8 betrachten.
   Save project - Die gesamte Arbeit als eine .vidap-Datei speichern.
   Load project - Eine .vidap-Datei kann hiermit geladen werden.

8. Das Feld, wo der Name der geladenen .vida-Datei angezeigt wird. Erscheint hier ein leeres Feld, ist somit keine Datei geladen worden.

9. Start loaded file - Startet den Versand-Vorgang aus der geladenen Datei.

10. Dies ist ein Balken, welcher den Versand-Vorgang-Fortschritt anzeigt.

11. Run action - Startet das Versenden aller Nachrichten basierend auf den ausgewählten Objekten der Szenen. Wurde keinem der Objekte eine SourceID vergeben, passiert nichts.
    Stop action - Stoppt den Versand-Vorgang der Software

Die Buttons unter dem Feld 1 des Bilds 1 haben selbstsprechende Namen und werden somit hier nicht erläutert.
Das Export-Fenster - Bild 2 - gibt die Möglichkeiten die Objekte der Szene mit der WFS-Anlage zu verknüpfen.

1. Export Objekts - Erscheint beim Übergang von dem Main Page.

2. Hier erscheint der Name der ausgewählten auf der Main Page Szene.

3. Liste der Animationsobjekte der Szene. Mit einem Doppelklick auf einen Item kann der Name des Objektes geändert werden.

4. Die Liste der Koordinaten mit dem Zeitpunkt des ausgewählten Objektes (es muss ein Objekt in der Liste 3 ausgewählt werden). Soll ein Zeitpunkt exportiert werden, muss der Hacken bei diesem Zeitpunkt gesetzt werden.

5. Dieses Feld dient zur Beeinflussung von den Koordinaten eines Objektes der Liste 3. Muss der Objekt in eine Richtung verschoben oder gedreht werden,

ist dieses hier einzugeben. Dafür muss das gewünschte Objekt in der Liste 3 ausgewählt sein. Eine Drehung um einen positiven Winkel bedeutet hier eine Drehung im Uhrzeigersinn.

6. Select All - Setzt alle Zeitpunkte des Objektes als ausgewählt.
   Deselect All - Setzt alle Zeitpunkte des Objektes als nicht-ausgewählt.
   Confirm SID - Setzt das eingegebene Zahl im Feld links von dem Knopf als Quellen-ID des ausgewählten Objektes.

7. Add activate - Fügt dem Feld 8 eine Aktivierungs-OSC-Nachricht hinzu.
   Add selected position - Fügt den in der Liste 4 ausgewählten Zeitpunkt Feld 8 hinzu.
   Add all selected positions - Fügt alle in der Liste 4 mit Haken ausgewählten Zeitpunkte Feld 8 hinzu.

8. Das Feld zur weitere Bearbeitung der Objekte, kann per Hand bearbeitet werden. Achtung - die in dieses Feld eingegebenen Nachrichten haben keine Auswirkung auf das Verschicken der OSC-Nachrichten aus diesem Programm.

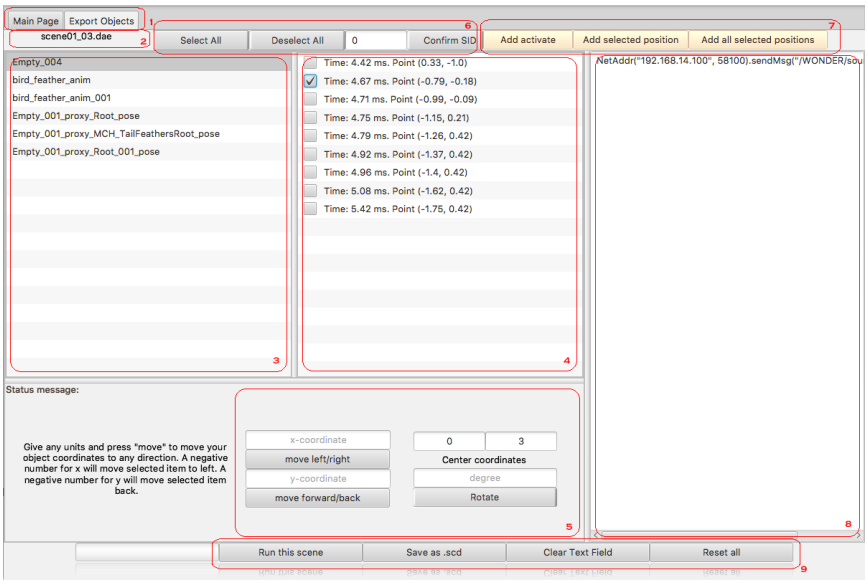


Figure 2: Das Export-Fenster des Programms

9. Run this scene - Startet das Versenden der Nachrichten nur für diese Szene.
   Save as .scd - Speichert den eingegebenen Text von Feld 8 als eine SuperCollider-Datei.

Clear Text Field - Leert das Feld 8.
Reset all - Setzt alle Einstellungen der Szene zurück.

5. Erste Schritte

Um eine Szene zu bearbeiten sind folgende Schritte auszuführen.

1. Drücken Sie den "'Choose File(s)"'-Button, um die .dae-Dateien auszuwählen. Es können mehrere Dateien gleichzeitig geladen werden und es lassen sich nachträglich Dateien zur Liste hinzufügen.

2. Wählen Sie eine Szene aus der geladenen Liste aus, in dem Sie einmal auf die Szene klicken.

3. Geben Sie den Zeitversatz für diese Szene ein. Bestätigen Sie diesen mit der "'Enter"'-Taste.

4. Mit einem Doppelklick auf die Szene oder per Knopfdruck auf das "'Continue with this scene"'-Button werden Sie zur Export-Ansicht weitergeleitet.

5. Wählen Sie ein Animationsobjekt aus der Liste aus, indem Sie einmal auf den Objektnamen klicken. Es erscheinen die Koordinaten des Objektes mit der dazugehörigen Zeit im mittleren Bereich. (Tipp: der Name des Objektes kann per Doppelklick auf das Objekt geändert werden)

6. Wählen Sie die Zeiten aus, die an die WFS-Anlage extrahiert werden sollen. (Tipp: mit dem Button "'Select All"' lassen sich alle Zeitpunkte des Objektes sofort aktivieren)

7. Geben Sie die Quellen-ID für dieses Objekt im Feld Source-ID ein und bestätigen Sie diese Ihre Eingabe mit dem "'Confirm SID"'-Button oder der "'Enter"'-Taste.

8. Sie können nun die Szene starten, indem Sie den "'Run this scene"'-Button drücken. Den Fortschritt kann man in dem Progress-Balken links von dem "'Run this scene"'-Button beobachten.

Die bearbeitete Szene oder Szenen lassen sich auch vom Hauptfenster starten. Dafür wechselt man zu dem Startbildschirm und startet den Versand-Vorgang über den "'Run action"'-Button. Wenn dazu ein Video geladen wurde, wird dieses mit dem Starten des Versand-Vorgangs auch gestartet.

B. Klassendiagramme

Im Folgenden sind zwei Varianten von dem Klassendiagramm der entworfenen Software dargestellt. Die hier abgebildeten Diagramme sind vollständig im Vergleich zu den, die in der Arbeit vorgestellt worden sind (s. Implementierung der Software). Das erste Diagramm zeigt die Abhängigkeit der packages. Das zweite Diagramm ist eine komplette Übersicht der Klassen.

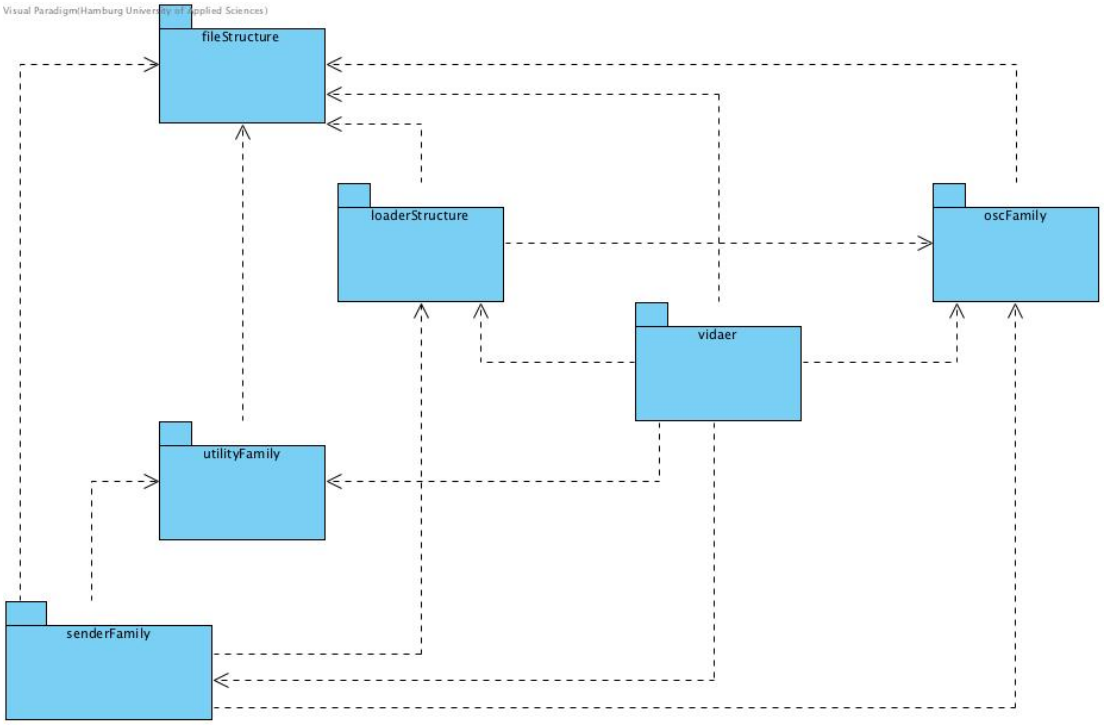
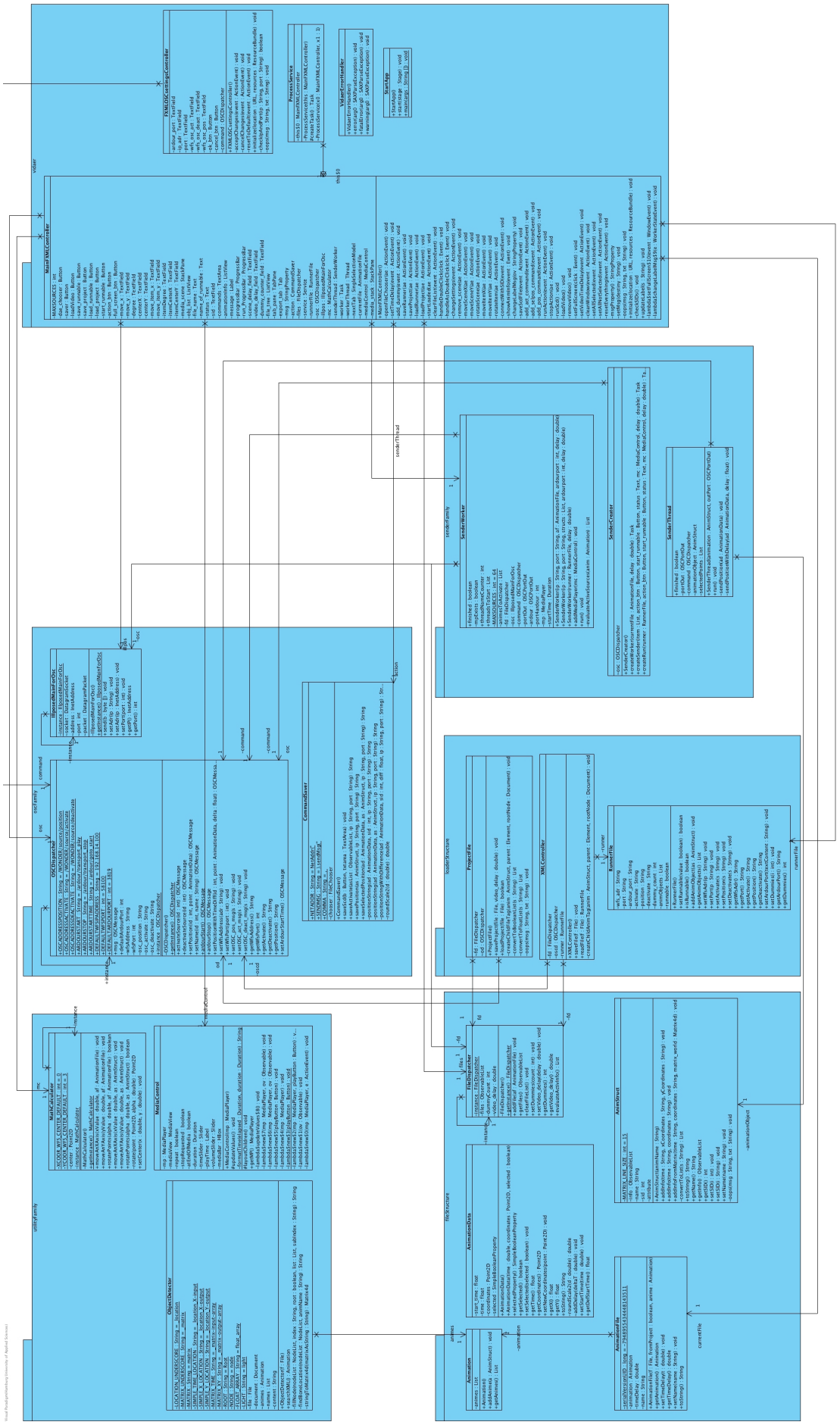


Abbildung B.1.: Das package-Diagramm der entworfenen Software.



Literaturverzeichnis

[Ardour] Ardour: *http://ardour.org*. – Abruf: 12.04.2016

[Audacity] Audacity: *http://www.audacityteam.org*. – Abruf: 05.04.2016

[Audinate] Audinate: *http://www.audinate.com*. – Abruf: 01.04.2016

[Auro Technologies] Auro Technologies : *http://www.auro-3d.com*. – Abruf: 22.06.2016

[Autodesk] Autodesk: *https://http://www.autodesk.com*. – Abruf: 05.04.2016

[Blender Foundation a] Blender Foundation: *Blender 2.68.5 - API documentation*. https://www.blender.org/api/blender_python_api_2_68_5/. – Abruf: 22.06.2016

[Blender Foundation b] Blender Foundation: *Blender Reference Manual*. https://www.blender.org/manual/. – Abruf: 30.05.2016

[Blender Foundation c] Blender Foundation: *www.blender.org*. – Abruf: 31.03.2016

[Davis und Letz] Davis, Paul ; Letz, Stéphane: *http://jackaudio.org*. – Abruf: 07.04.2016

[Dolby Laboratories Inc.] Dolby Laboratories Inc.: *http://www.dolby.com*. – Abruf: 22.06.2016

[Fohl 2013] Fohl, Wolfgang: *The Wave Field Synthesis Lab at the HAW Hamburg*. In: Bader, Rolf (Hrsg.): *Sound - Perception - Performance*. Hamburg : Springer, 2013. – ISBN 978-3-319-00106-7

[FourAudio] FourAudio: *http://www.fouraudio.com*. – Abruf: 01.04.2016

[Google Inc. a] Google Inc.: *Automatic large scale video object recognition*. http://www.freepatentsonline.com/8254699.html. – Abruf: 30.06.2016

[Google Inc. b] Google Inc.: *https://www.google.de*. – Abruf: 22.06.2016

[Johns 2014] Johns, Robert: *Klangquellen-Positionierung in verschiedenen Surround Formaten auf Basis von Positionsdaten einer Wellenfeldsynthese-Anlage*. 2014

[JUnit] JUnit: *http://junit.org/junit4/*. – Abruf: 6.06.2016

[Kersken 2013] Kersken, Sascha: *IT-Handbuch für Fachinformatiker - der Ausbildungsbegleiter*. 6. Aufl. Bonn : Galileo Press, 2013. – ISBN 978-3-8362-2234-1

[Khronos Group] Khronos Group: *COLLADA Specification 1.5.0*. https://www.khronos.org/collada/

[Makarski u. a.] Makarski, Michael ; Goertz, Anselm ; Thaden, Rainer ; Kleber, Jochen: *Entwicklung einer WFS-Anlage mit FIR- Entzerrung und Dante Audionetzwerk*. http://www.ifaa-akustik.de/files/tmt2012_makarski_fir-basierte-wfs-anlage-mit-audionetzwerk.pdf. – Abruf: 29.03.2016

[Native Instruments] Native Instruments: *http://www.native-instruments.com/de/products/komplete/synths/reaktor-6/*. – Abruf: 05.04.2016

[Oellers] Oellers, Helmut: *Wave Field Synthesis*. http://www.syntheticwave.de/Wavefieldsynthesis.htm. – Abruf: 21.03.2016

[OpenSound Control] OpenSound Control: *http://opensoundcontrol.org*. – Abruf: 01.04.2016

[Paulsen 2014] Paulsen, Arnd: *Audio-Entwicklungen im Heim- und Kinobereich. Strategien für eine noch bessere Unterhaltung*. https://www.irt.de/fileadmin/media/downloads/veranstaltungen/tmt2014/IRT_TT_6_Paulsen.pdf. November 2014. – Abruf: 22.06.2016

[Project Peach a] Project Peach: *http://bbb3d.renderfarming.net/explore.html*. – Abruf: 05.04.2016

[Project Peach b] Project Peach: *https://peach.blender.org*. – Abruf: 05.04.2016

[QjackCtl] QjackCtl: *http://qjackctl.sourceforge.net*. – Abruf: 07.04.2016

[Ramakrishnan] Ramakrishnan, Chandrasekhar: *Illposed Software*. http://www.illposed.com/software/javaosc.html. – Abruf: 6.06.2016

[Roubtsov] Roubtsov, Vlad: *EclEmma*

[Slavik und Weinzierl 2008] Slavik, Karl M. ; Weinzierl, Stefan: *Wiedergabeverfahren*. In: Weinzierl, Stefan (Hrsg.): *Handbuch der Audiotechnik*. Berlin : Springer, 2008. – ISBN 978-3-540-34300-4

[SuperCollider a] SuperCollider: *http://supercollider.github.io*. – Abruf: 22.06.2016

[SuperCollider b] SuperCollider: *http://swiki.hfbk-hamburg.de/MusicTechnology/13*. – Abruf: 22.06.2016

[Wartmann 2011] Wartmann, Carsten: *Das Blender-Buch*. 4. Aufl. Heidelberg : dpunkt.verlag, 2011. – ISBN 978-3-89864-610-9

[Watson] Watson, Nick: *Dolby Atmos erklärt*. http://www.professional-production.de/fileadmin/user_upload/img/fachbeitrag/13/13-10/pdf/DolbyAtmos.pdf. – Abruf: 23.06.2016

Hiermit versichere ich, dass ich die vorliegende Arbeit ohne fremde Hilfe selbständig verfasst und nur die angegebenen Hilfsmittel benutzt habe.

Hamburg, 5. Juli 2016 Ilyuza Mingazova