Deutsches Zentrum
für Luft- und Raumfahrt

Hochschule Emden/Leer
Fachbereich: Technik
Abteilung: E+I
Studiengang: Medientechnik

Bachelorarbeit

Realisierung einer Audio Engine in CosmoScout VR

Vorgelegt von Florian Saß
Matr. Nr.: 7013872
Abgabe am 29. Janaur 2024

Erstprüfer: Prof. Dr. Maria Rauschenberger
Zweitprüfer: M.Sc. Moritz Zeumer

Eidesstattliche Versicherung

Ich, der/die Unterzeichnende, erkläre hiermit an Eides statt, dass ich die vorliegende Arbeit selbständig verfasst habe und keine anderen als die angegebenen Quellen und Hilfsmittel benutzt habe. Alle Quellenangaben und Zitate sind richtig und vollständig wiedergegeben und in den jeweiligen Kapiteln und im Literaturverzeichnis wiedergegeben. Die vorliegende Arbeit wurde nicht in dieser oder einer ähnlichen Form ganz oder in Teilen zur Erlangung eines akademischen Abschlussgrades oder einer anderen Prüfungsleistung eingereicht.

Mir ist bekannt, dass falsche Angaben im Zusammenhang mit dieser Erklärung strafrechtlich verfolgt werden können.

Braunschweig, 29.01.2024

I Abstract

Die Darstellungen von Datensätzen sind ein wichtiger Bestandteil in der Analyse von Datensammlungen. Die klassische Darstellung eines Datensatzes ist die Visualisierung, welche durch ihre meist einfache Verständlichkeit ein optimales Werkzeug ist. Jedoch kann dieser Ansatz schnell an seine Grenzen stoßen, wenn vielschichtige und komplexe Zusammenhänge dargestellt werden sollen. In solch einem Fall kann es sich lohnen, eine Kombination aus unterschiedlichen Medien zu nutzen, um verschiedene Dimensionen in einer Datensammlung darzustellen.

Das Ziel dieser Arbeit ist es, die Software CosmoScout VR, ein virtuelles Sonnensystem zur visuellen Darstellung von astronomischen Datensätzen, um ein Audiosystem zu erweitern, damit eine auditive Darstellung von Daten, auch Sonifikation genannt, möglich ist. Das Audiosystem, auch als Audio Engine bezeichnet, baut auf die Audiobibliothek OpenAL-Soft auf, eine plattformübergreifende Softwareschnittstelle zur Audiohardware, um Audioquellen dreidimensional darzustellen. Die Audio Engine implementiert für die Anforderungen erforderliche Funktionalitäten, wie z. B. ein semimodulares Design, mit welchem eine einfache Anpassung an einen Anwendungsfall möglich ist und neue Funktionen leichter implementiert werden können. Des Weiteren gibt es ein automatisches Bufferverwaltungssystem und Techniken, um die Limitierungen der gleichzeitig spielenden Audioquellen in OpenAL zu erweitern.

Außerdem wird eine Möglichkeit angeschnitten, wie eine Sonifikation mittels der Audio Engine möglich ist. Es wurde ein funktionierender, wenn auch nicht fertiggestellter, Prototyp einer Sonifikationsmethode entwickelt, von der aus in Zukunft eine fundiertere Methode abgeleitet werden kann.

Inhaltsverzeichnis

1 Einleitung

In einer Welt, die zunehmend von digitalen Technologien geprägt ist, spielt die Kombination verschiedener Medien eine wichtige Rolle bei der Wahrnehmung und Verarbeitung von Informationen. Die Darstellung von wissenschaftlichen Daten hat bisher vorwiegend auf visuellen Darstellungen beruht. Dieser Fokus folgte oftmals aus technischen Einschränkungen, da viele Informationsmedien keine Audioausgabe unterstützen. Allerdings haben die Fortschritte in der Technologie dazu geführt, dass viele Aspekte des menschlichen Lebens in die digitale Welt verlagert wurden, in der die Konsumenten zunehmend auf die kombinierte Darstellung von Bild und Ton angewiesen sind. Heutzutage verfügt ein Großteil der Menschen über die technischen Möglichkeiten, sowohl bewegte Bilder als auch Töne wahrzunehmen.

Wie kann man diese Veränderung und die daraus resultierenden neuen Möglichkeiten optimal nutzen? In der Welt der Wissenschaft stehen bereits zahlreiche Werkzeuge zur Verfügung, um Daten visuell aufzuarbeiten. Um eine vergleichbare Unterstützung für auditive Informationen zu erreichen, benötigt es geeignete Werkzeuge zur Verarbeitung von Inhalten. Wie können solche Werkzeuge entwickelt und eingesetzt werden, um den Anforderungen an die Darstellung auditiver Informationen gerecht zu werden?

1.1 Motivation und Ziel der Arbeit

Am Institut für Softwaretechnologie des Deutschen Zentrums für Luft- und Raumfahrt wird die Software Cosmoscout VR entwickelt. Mit Hilfe dieser Software sind bereits komplexe Visualisierungen im astronomischen Kontext realisierbar. Es ist ein virtuelles Sonnensystem, mit welchem astronomische Datensätze in Echtzeit präsentiert und analysiert werden können. [20] Bisher ist jedoch nur eine Visualisierung der Daten möglich und es existiert bisher keine Unterstützung für auditive Inhalte.

Das Ziel dieser Arbeit ist es, in CosmoScout VR ein Audiosystem zu implementieren. Dieses Audiosystem, auch Audio Engine genannt, hat das Ziel, sowohl eine allgemeine Unterstützung von Audio in der Software zu ermöglichen, als auch eine Darstellung wissenschaftlicher Datensätze, besonders im astronomischen Kontext, auditiv realisierbar zu machen. Eine solche Darstellung von Datensätzen wird auch als Sonifikation bezeichnet und stellt das übergeordnete Endziel dieses Audiosystem dar.

1.2 Aufbau der Arbeit

Die Arbeit ist in die Abschnitte Grundlagen, Entwicklung und Auswertung unterteilt. Im Abschnitt Grundlagen werden wesentliche Sachverhalte erklärt, die für ein Verständnis dieser Arbeit erforderlich sind. Der Abschnitt Entwicklung gliedert sich in zwei Teile: die Auswahl einer Audiobibliothek und die Vorstellung der Audio Engine. Der erste Teil präsentiert einen Vergleich potenzieller Bibliotheken, auf denen die Audio Engine aufbauen kann, und zeigt den Auswahlprozess auf. Der zweite Teil stellt die grundlegenden Funktionen des Audiosystems vor. Abschließend erfolgt eine Auswertung der Audio Engine über die Qualität sowie ein Ausblick darauf, wie diese für Sonifikationszwecke genutzt werden kann.

In dieser Arbeit sind alle Begriffe, die *kursiv* geschrieben sind im Glossar enthalten.

2 Related Work

Eine ähnliche und bereits existierende Arbeit ist die Arbeit von Elmquit et al. [7]. Auch in dieser wird durch ein Audiosystem eine Sonifikaton in einem virtuellen Sonnensystem durchgeführt. Jedoch handelt es sich bei der genutzten Software um OpenSpace, ebenfalls ein virtuelles Sonnensystem zur visualliserung von astronomischen Datensätzen [4]. Des Weiteren beschreibt die Arbeit kein allgemeines Audiosystem, sondern ist eine spezielle Anwendung eines Synthesizer zur verklanglichung von Planeteneigenschaften im Rahmen von Präsentationen in einem Planetarium.

3 Grundlagen

In diesem Kapitel werden die für diese Arbeit relevanten Oberbegriffe erläutert.

3.1 CosmoScout VR

CosmoScout VR ist ein virtuelles modulares Sonnensystem, welches eine interaktive Erkundung und Präsentation von astronomischen Datensätzen ermöglicht. Die Software ist *open-source* und plattformübergreifend nutzbar. Es unterstützt die Wiedergabe auf einer Vielzahl an Ausgabeformaten, wie z. B. auf Desktops, in Virtual Reality (im Verlaufe dieser

Arbeit oftmals unter der Verwendung eines *Head-Mounted-Display* bezeichnet) oder *stereoskopische* Mehrbildschirm-Systeme.

Ein wichtiges Konzept für das Verständnis von CosmoScout VR ist die pluginbasierte Natur der Software. Der eigentliche Kern der Software ist alleine nicht sinnvoll nutzbar. Alle Inhalte und Funktionalitäten, mit denen ein Nutzer hauptsächlich in Kontakt kommen wird, werden durch Plugins, also optionale Softwarekomponenten, die an- und abwählbar sind, implementiert. [20].

3.2 Sonifikation

Die Sonifikation ist ein Teilgebiet der auditory displays. Ein auditory display ist, sehr allgemein formuliert, eine beliebige Darstellung von Informationen, welche durch Klang ausgedrückt wird [26]. Die Sonifikation als spezielle Form eines auditory displays lässt sich als datenabhängige Klanggenerierung definieren. Wichtig ist in diesem Zusammenhang, dass die Umwandlung der Daten in einen Klang „systematisch, objektiv und reproduzierbar“ ist, da nur unter diesen Umständen eine Nutzung im wissenschaftlichen Kontext haltbar ist. [10]

4 Audio Bibliothek

Da die Audio Engine nicht von Grund auf alle Audiomöglichkeiten selbst implementieren soll oder kann, da dies den Umfang dieser Arbeit übersteigen würde, baut die Audio Engine auf eine bereits existierende Audiobibliothek auf. In diesem Kapitel wird aufgezeigt, wie die Wahl für eine Audiobibliothek zustande kam.

4.1 Welche Audiobibliotheken gibt es?

Um eine geeignete Audiobibliothek für die Audio Engine zu finden, wurde eine Recherche im Sinne einer systematischen Literaturrecherche durchgeführt. Für die Suche wurde die folgende Menge an Suchbegriffen definiert: 'audio library', 'c++', 'spatialization', 'sound library', 'spatialized sound' und 'audio engine'. Mittels dieser Sammlung wurde entweder mit einzelnen Begriffen oder einer Kombination aus den Begriffen in den Suchmaschinen Google, Ecosia und GitHub nach verfügbaren Audiobibliotheken gesucht. Eine Auflistung aller Bibliotheken, die in die nähre Betrachtung kamen, und eine kurze Beschreibung ihrer

Schlüsselfunktionalitäten ist wie folgt gegliedert:

OpenAL

OpenAL ist eine plattformübergreifende Softwareschnittstelle zur Audiohardware eines Computers, welches in Syntax, Konventionen und Stil *OpenGL* nachempfunden ist. Die Bibliothek bietet alle grundlegenden Funktionen, um Audioquellen dreidimensional zu verräumlichen, unter anderem mit Distanzdämpfung und Dopplereffekt, und stellt außerdem Funktionalitäten für Audioaufnahmen [11, S. 8-9], sowie Effekte und Filter bereit [18].

SDL2

SDL2 ist eine Bibliothek für einen hardwarenahen Zugriff auf medienrelevante Komponenten eines Computers, wie z. B. Audio, Grafik oder die Steuerung [21]. Um die Audio Funktionalitäten zu Nutzen wird die SDL2 Erweiterng SDL2 Mixer benötigt. Mittels dieser Bibliothek ist es möglich, Audioquellen zu verräumlichen, jedoch ist in der Distanz eine Differenzierung nur im Bereich von 0 bis 255 möglich [24].

Resonance Audio

Resonance Audio ist eine webbasierte ‚spatial audio SDK', mit welcher unter anderem eine Anpassung der Schallgeschwindigkeit, Nahfeldeffekte, *Occlusion* oder geometrie-basierter Nachhall möglich ist [19].

FMOD Core

FMOD Core ist eine Echtzeit-Audio-Engine, welche Funktionalitäten wie Distanzdämpfung, Dopplereffekt, polygonbasierte *Occlusion*-Effekte oder ein Virtual-Voice-System (System, um nicht hörbare Audioquellen automatisch abzuschalten) bietet. Die Audiobibliothek ist ein kommerzielles Produkt, welches aber für nichtkommerzielle Nutzung kostenlos ist [8].

irrKlang

IrrKlang ist eine einfache zwei- und dreidimensionale Audio Engine, welche zwar Funktionalitäten wie Effekte und Audioaufnahmen bietet, aber keine Verräumlichkeitseffekte enthält, wie z. B. den Dopplereffekt oder Nachhall [9]

soLoud

SoLoud ist eine portable Audio Engine, welche eine Distanzdämpfung,

den Dopplereffekt, eine Distanzverzögerung und eine Anpassung der Schallgeschwindigkeit bietet. Des Weiteren bietet die Bibliothek auch ein Virtual-Voice-System (siehe FMOD Core) und einen einfachen Synthesizer [15].

aeon wave

Aeon wave ist eine drei- und vierdimensionale Audiobibliothek, welche auch die typischen dreidimensionalen Audioeffekte bietet, wie z. B. Distanzverzögerung, Distanzdämpfung, Dopplereffekt und *Occlusion*. Des Weiteren werden auch Funktionalitäten wie ein Synthesizer, Konfigurationsdateien und eine Gruppierung von Audioquellen in einer Baumstruktur geboten [2].

4.2 Anforderungen und Tests

Um die richtige Audiobibliothek auszuwählen, wurden die folgenden Anforderungen und die daraus resultierenden Tests aufgestellt. Die Anforderungen ergeben sich aus dem Ziel der Audio Engine, Diskussionen mit möglichen Entwicklern und dem Überblick an Funktionen, die von einer Audio-Bibliothek zu erwarten sind.

Die Anforderungen lassen sich in zwei Kategorien einteilen: funktionelle Anforderungen und benotete Anforderungen. Funktionelle Anforderungen sind Anforderungen, die sich meist mit einem Erfüllt oder nicht Erfüllt beantworten lassen. Benotete Anforderungen hingegen werden durch einen Wert auf einem Spektrum ausgedrückt, welcher den Umfang oder die Benutzbarkeit einer Audiobibliothek anhand eines Kriteriums beschreibt.

4.2.1 Funktionelle Anforderungen

Funktionelle Anforderungen lassen sich entweder mit „erfüllt“, „teilweise erfüllt“ oder „nicht erfüllt“ beantworten und sind wie folgt absteigend gewichtet:

1. Mindestanforderungen, die von der Audiobibliothek erfüllt sein müssen:
   - A1: Eine möglichst große Anzahl an Audioquellen im dreidimensionalen Raum zu platzieren TODO: anpassen
   - A2: Den Klang von Audioquellen anhand ihrer Position zum Hörer verräumlichen

2. Ein Großteil dieser Anforderungen sollte von der Audiobibliothek erfüllt sein:
    - B1: Eine automatische Berechnung des Dopplereffekts
    - B2: Eine Dämpfung der Lautstärke anhand der Distanz
    - B3: Eine Verzögerung des Klangs anhand der Distanz
    - B4: Eine variable Schallgeschwindigkeit
    - B5: Methoden zur Realisierung von *Occlusion*
    - B6: Unterstützung von Surround-Sound-Systemen
    - B7: Möglichkeiten zur Modellierung atmosphärischer Eigenschaften
    - B8: Möglichkeiten, um Daten zu *streamen*
    - B9: Unterstützung von *HRTF*
3. Diese Anforderungen sind nützliche Extras, aber nicht notwendig:
    - C1: Möglichkeiten, den Klang eines Tons anzupassen, z. B. über Effekte oder Filter
    - C2: Möglichkeiten, um Klänge zu erzeugen
    - C3: Möglichkeiten, um die Lautsprecherpositionen anzupassen TODO was ist damit genau gemeint?

4.2.2 Benotete Anforderungen

Benotete Anforderungen werden auf einer absteigenden Skala von 1–6 benotet und lauten:

- D1: Performance der Audiobibliothek
- D2: Qualität der Dokumentation
- D3: Schwierigkeit der Entwicklung
- D4: Abhängigkeiten von anderen Bibliotheken

4.2.3 Tests

Um die formulierten Anforderungen zu überprüfen, wurden die nachfolgenden Tests definiert:

1. **Verräumlichung**
   Aufbau: Eine Quelle wird im Raum platziert und der Hörer wird um die Quelle herum bewegt.
   Ziel: Prüfung der Verräumlichug.
   Anforderung(en): A2

2. **Verräumlichung und Mixing**
   Aufbau: Mehrere Quellen werden kreisförmig um den Hörer platziert. Jede Quelle erhält einen anderen Klang zum ausspielen.
   Ziel: Prüfung der Verräumlichung mit mehreren Quellen.
   Anforderung(en): A2

3. **Stresstest**
   Aufbau: 1. Performancemessung: Die Anzahl an Audioquellen wird schrittweise auf 1000 erhöht. Dabei wird der CPU- und Speicherbedarf aufgenommen. 2. Quellenmaximum: Die Anzahl an Audioquellen schrittweise erhöhen, bis die Audioausgabe fehlerhaft wird.
   Ziel: 1. Performancemessung: CPU- und Speicherbedarf ermitteln. 2. Quellenmaximum: Die maximale Anzahl an gleichzeitig spielenden Quellen herausfinden.
   Anforderung(en): A1

4. **Dopplereffekt**
   Aufbau: Der Hörer wird im Raum platziert und eine Quelle wird mit einer hohen Geschwindigkeit am Hörer vorbeifliegen. Die Geschwindigkeit wird, falls möglich, an- und abschaltbar sein. (Anmerkung: Die Geschwindigkeit ist in den meisten Audiobibliotheken i.d.R. eine komplett unabhängige Variable. D.h. eine Positionsänderung ist möglich, ohne eine Geschwindigkeit zu besitzen.)
   Ziel: Den Einfluss des Dopplereffekts auf den Klang überprüfen.
   Anforderung(en): B1

5. **Distanzdämpfung**
   Aufbau: Eine Quelle wird im Raum platziert und der Hörer in Entfernung dazu aufstellt. Die Entfernung sollte schrittweise erhöht werden.
   Ziel: Überprüfung der Distanzdämpfung.
   Anforderung(en): B2

6. **Distanzverzögerung und Schallgeschwindigkeit**
   Aufbau: Eine Quelle wird im Raum platziert und der Hörer in einer variablen Entfernung dazu aufstellt. Die Schallgeschwindigkeit

und die Hörerentfernung sollten, falls möglich, variabel sein.
Ziel: Schallgeschwindigkeitsänderung und Distanzverzögerung überprüfen.
Anforderung(en): B3 und B4

7. ***Occlusion***
   Aufbau: Der Hörer und eine Quelle werden gegenüber aufstellt und eine Wand wird dazwischen platziert. Die Position des Hörers wird so anpassbar sein, dass die direkte Linie zwischen dem Hörer und der Quelle von der Wand sowohl geschnitten als auch nicht geschnitten werden kann.
   Ziel: Möglichkeiten einer Implementierung von *Occlusion* testen.
   Anforderung(en): B5

8. **Surround-Sound**
   Aufbau: Ein vorheriger Testaufbau wird wiederholt und auf einem Surround-Sound-System ausgegeben.
   Ziel: Umgang mit Surround-Sound testen.
   Anforderung(en): B6

9. **Atmosphärische Eigenschaften**
   Aufbau: Es wird überprüft, welche Einstellungsmöglichkeiten es gibt, um unterschiedliche atmosphärische Eigenschaften darzustellen.
   Ziel: Umfang der atmosphärischen Einstellungsmöglichkeiten herausfinden.
   Anforderung(en): B7

10. ***Streaming* langer Inhalte**
    Aufbau/Ziel: Es wird überprüft, welche Möglichkeiten es gibt, um lange Audioinhalte wiederzugeben.
    Anforderung(en): B8

11. ***HRTF* Unterstützung**
    Aufbau/Ziel: Es wird überprüft, ob *HRTF* unterstützt wird.
    Anforderung(en): B9

12. **Stresstest mit *HRTF***
    Aufbau: Test Nummer 3 und 11 werden mit *HRTF* wiederholt.
    Ziel: Performanceeinfluss von *HRTF* ermitteln.
    Anforderung(en): B9 und A1

13. **Effeke und Filter**
    Aufbau: Es wird überprüft, welche Effekte- und Filtermöglichkeiten es gibt.
    Ziel: Möglichkeiten der Effekt- und Filterwerkzeuge überprüfen.
    Anforderung(en): C1

14. **Stresstest mit Effekten und Filter**
    Aufbau: Test Nummer 3 wird mit einer Auswahl an Effekten und Filtern wiederholt.
    Ziel: Performanceeinfluss von Effekten und Filtern testen.
    Anforderung(en): C1 und A1

15. **Tonerzeugung**
    Aufbau: Es wird überprüft, welche Möglichkeiten es gibt, um Töne zu erzeugen.
    Ziel: Möglichkeiten der Tonerzeugung überprüfen.
    Anforderung(en): C2

16. **Lautsprecherpositionen konfigurieren**
    Aufbau/Ziel: Es wird überprüft, welche Möglichkeiten es gibt, um die Position von Lautsprechern anzupassen.
    Anforderung(en): C3

4.3 Testsoftware

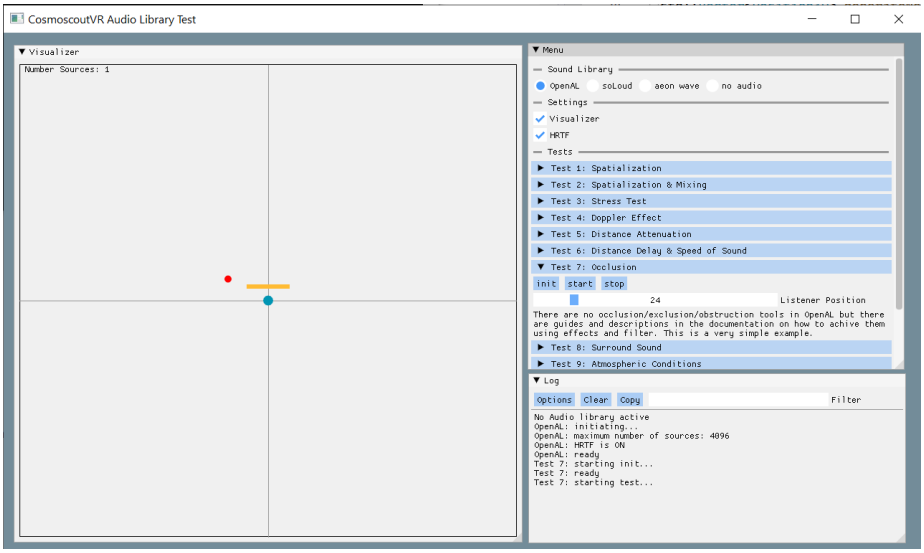

Abbildung 2: Screenshot der Testsoftware. Zu sehen ist der *Occlusion*-Test mit OpenAL. Die Visualisierung der Audioszene wird im „Visualizer“-Fenster dargestellt. Der rote Punkt zeigt den Hörer, der türkise Punkt eine Audioquelle und der gelbe Balken eine Wand. Im „Menu“-Fenster lässt sich in der Kategorie „Sound Library“ die Audiobibliothek ändern. In der Kategorie „Tests“ ist eine Auflistung aller durchzuführenden Tests zu sehen, welche gestartet oder beendet werden können. Falls notwendig, bietet die Software dort auch Anpassungsmöglichkeiten für ein Testszenario, wie z. B. in der Abbildung zu sehen, die Echtzeitanpassung der Hörerposition mittels eines Schiebereglers.

Zur Durchführung der Tests wurde eine eigene Software entwickelt. Das Ziel dieser Software ist es, einen möglichst einfachen Vergleich zwischen den Audiobibliotheken zu ermöglichen. Um dies zu erreichen, bietet die Software unter anderem eine 2-dimensionale Visualisierung der Audioszene, eine einfache Methode, um die Audiobibliothek zu wechseln, und erlaubt einen einfachen Übergang zwischen den verschiedenen Testvarianten. Ein Screenshot der Software und den gerade beschriebenen Eigenschaften lässt sich in Abbildung 2 sehen.

Zu beachten ist, dass nicht jeder Test ein eigenes Testszenario in der Software erhält, da einige Anforderungen sich auch ohne expliziten Fall überprüfen lassen, wie z.B. die Unterstüzung von HRTF oder Surround-Sound.

Das Testprogramm ist in C++ geschrieben und nutzt die „Dear Im-

Gui“ Bibliothek für die Benutzeroberfläche [5]. Zur Aufnahme des CPU- und Speicherbedarfs für die Stresstests wird die WIN32 API „Performance Counters“ genutzt [17]. Die Durchführung eines Quellen-Maximums-Tests (Test Nummer 3, 12 und 14) wird sowohl im *Release*- als auch im *Debug-Modus* durchgeführt, um auch das Limit, welches während der Entwicklung eintreten würde, zu testen. Die reine Performancemessung wird dagegen nur im Release-Modus durchgeführt.

Da eine Durchführung aller Tests in allen Audiobibliotheken aus Kapitel 4.1 den Rahmen dieser Arbeit übersteigen würde, wurde im Vorfeld bereits eine Vorabentscheidung über die vielversprechendsten Bibliotheken anhand der aufgestellten Anforderungen und der theoretischen Fähigkeiten einer Bibliothek getroffen. Die Ausschlusskriterien lassen wie folgt zusammenfassen: SDL2 besitzt in der Entfernung nur eine Differenzierung von 0 bis 255 Einheiten, was für die großen Ausdehnungen von Cosmoscout VR zu gering ist. Resonance-Audio ist web-basiert. Zwar wäre eine Umsetzung möglich, da die Benutzeroberfläche von CosmoScout VR eine Webseite ist, aber dies ist nicht gewünscht. FMOD Core ist ein kommerzielles Produkt, welches zwar für nicht kommerzielle Produkte kostenlos ist, aber in der open-source Natur von CosmoScout VR herrausstechen würde. IrrKlang bietet nicht gewünschten Funktionsumfang. Zur finalen Auswahl stehen schließlich: OpenAL, soLoud und aeon wave.

4.4 Testergebnisse und Wahl der Audiobibliothek

Hier folgt eine zusammengefasste Form der Ergebnisse, um einen möglichst einfachen Überblick über den Vergleich der Audiobibliotheken zu erhalten. Eine detaillierte Auflistung der Testergebnisse lässt sich im Anhang A.1 finden. Die erhobenen Daten aus den Performancemessungen lassen sich in der angehängten Datei „Performancemessungen_Audio_Bibliotheken“ einsehen (siehe Anhang A.2).

Für die Durchführung der Tests wurde ein System mit der folgenden Hardware verwendet:

Betriebssystem	Windows 10 Enterprise
CPU	Intel Xeon E5-2690 v4 2.6GHz
RAM	128 GB DDR4
GPU	Nvidia Quadro P6000

Tabelle 1: Hardware-Komponenten des Testsystems

Anforderung	OpenAL	soLoud	aeon wave
A1	erfüllt	erfüllt[a]	erfüllt
A2	erfüllt	erfüllt	erfüllt
B1	erfüllt	erfüllt	erfüllt
B2	erfüllt	erfüllt	erfüllt
B3	nicht erfüllt	erfüllt	erfüllt
B4	nicht erfüllt	erfüllt	nicht erfüllt
B5	teilw. erfüllt	teilw. erfüllt	erfüllt
B6	erfüllt	erfüllt	nicht erfüllt
B7	teilw. erfüllt	nicht erfüllt	teilw. erfüllt
B8	teilw. erfüllt	erfüllt	teilw. erfüllt
B9	erfüllt	nicht erfüllt	erfüllt
C1	erfüllt	erfüllt	erfüllt
C2	nicht erfüllt	erfüllt	erfüllt
C3	teilw. erfüllt	erfüllt	nicht erfüllt
D1	2	2	3
D2	1-	2	4
D3	3	1	2-
D4	1	1	2

[a] auf 255 gleichzeitig spielende Quellen begrenzt

Tabelle 2: Überblick über die Testresultate der Audiobibliotheken

aeon wave

Die Audiobibliothek aeon wave bietet, zumindest auf dem Papier, die beste Auswahl an Funktionalitäten. Dazu zählen unter anderem Konfigurationsdateien, um eine Audioszene zu beschreiben, die größte Auswahl an Effekten und Filtern, *Occlusion*-Werkzeuge, MIDI-Unterstützung [2] und ein Buffermanagement-System, mit dem *Buffer* automatisch wiederverwendet werden, wenn die selbe Audio-Datei ein weiteres Mal ausgespielt werden soll [1].

Allerdings überwiegen die negativen Aspekte von aeon wave die positiven. Zu den Hauptgründen zählen die im Vergleich schlechte Dokumentation und die nicht nahtlose automatische Wiederholung einer Wiedergabe. Des Weiteren konnte kein anderes Programm zur selben Zeit Audio ausgeben, solange aeon wave ausgeführt wurde, obwohl dies laut einer von aeon wave bereitgestellten Funktion möglich sein sollte. Außerdem war die Surround-Sound-Ausgabe aus unerkenntlichen Grün-

den nicht möglich.

soLoud

SoLoud stellt dagegen die beste Wahl dar, wenn man nach Anzahl der erfüllten funktionellen Anforderungen und dem Durchschnitt der benoteten Anforderungen geht (siehe Tabelle 2). Des Weiteren ist die Audiobibliothek sehr einfach zu benutzen und besonders die Anpassung der Schallgeschwindigkeit und die Echtzeitanpassung von Lautsprecherpositionen stechen im Vergleich positiv heraus [15].

Allerdings gibt es auch hier negative Aspekte, die überwiegen. Dazu gehört der fehlende Support von *HRTF*, was ein natürlicheres Hörerlebnis ermöglichen würde. Außerdem gab es ein Limit von 255 gleichzeitig spielenden Quellen, obwohl bis zu 4096 Quellen möglich sein sollen. Dies war aber aus unerkenntlichen Gründen nicht möglich. Das ausschlaggebendste Kriterium ist aber, dass es seit 3 Jahren kein Update mehr für soLoud gab, was für eine möglichst zukunftssichere Audio Engine von Nachteil wäre [16].

OpenAL

OpenAL bietet im Vergleich die geringste Anzahl an funktionellen Anforderungen (siehe Tabelle 2) und bietet auch außerhalb dieser den geringsten Funktionsumfang. Wie etwa das Nichtvorhandensein einer Funktion, um Dateien auszulesen oder Audioquellen zu gruppieren.

Aber dafür bietet OpenAL die beste Dokumentation und es gibt viele Informationen und Hilfen zur Bibliothek im Internet (Foren, Artikel, Guides, ...). Außerdem ist OpenAL an *OpenGL* angelehnt, was einen einfacheren Umstieg für andere Entwickler ermöglichen sollte, da die Grafik Engine von CosmoScout VR auf *OpenGL* basiert. Des Weiteren sind die Funktionalitäten, die in OpenAL fehlen, noch am einfachsten selbst umsetzbar, wie z. B. das Erzeugen eines Klangs oder die Verzögerung eines Klangs über die Distanz. Hinzu kommt, dass OpenAL aktiv weiterentwickelt wird und der OpenAL-Soft Entwickler auf Anfragen zeitnah antwortet.

Aus den gerade aufgestellten Gründen wird OpenAL für die Audio Engine von CosmoScout VR genutzt. Besonders die Robustheit der grundlegenden Funktionen, die vielen Anlaufstellen für Informationen, die aktive Weiterentwicklung und die zeitnahe Antwort auf Anfragen vom Entwickler sind ausschlaggebende Kriterien für die Wahl.

5 Audio Engine

Als Audio Engine wird in dieser Arbeit der Programmteil verstanden, der es Entwicklern ermöglicht, Audio in CosmoScout VR darzustellen, ohne dass sich der Entwickler um dessen Implementierung Gedanken machen muss. Die Audio Engine ist speziell auf CosmosScout VR zugeschnitten und wird auf die im vorherigen Kapitel ausgewählte Audiobibliothek, OpenAL, aufbauen.

5.1 Anforderungen

Aus dem Ziel für die Audio Engine ergeben sich die folgenden übergeordneten Anforderungen:

1. Möglichst viele Audioquellen im Raum gleichzeitig abspielen.
2. Eine einfache *Schnittstelle* für Entwickler.
3. Unterstützung von allen Plattformen, die auch CosmoScout VR unterstützt. Dazu zählen Desktops, *Head-Mounted-Displays* und *multi-pipe rendering clusters* unter Windows und Linux [20, S. 4]. Daraus resultiert die Unterstützung folgender Audioausgabeformate: Stereo über Kopfhörer und Lautsprecher sowie 5.1 und 7.1 Surround-Sound über Lautsprecher. Dabei sollte eine Unterscheidung zwischen stationären Ausgabegeräten und mitbewegenden Ausgabegeräten unterschieden werden. Als stationäre Ausgabegeräte wird der Normalfall verstanden, wie z. B. die Nutzung von Kopfhörern auf dem Desktop oder Lautsprechern mit einem *Head-Mounted-Display*. Ein mitbewegendes Ausgabegerät ist hingegen eins, welches immer der Bewegung des Nutzer folgt, wie z. B. bei der Nutzung von Kopfhörern mit einem *Head-Mounted-Display*.
4. Eine einfache Möglichkeit, um die Audio Engine mit neuen Funktionalitäten zu erweitern, da eine Audio Engine sehr komplex sein kann und der Umfang dieser Arbeit nur begrenzt ist.

5.2 Ideen, Konzepte und Funktionsweise

Um die Anforderungen zu erfüllen, ist die Audio Engine nach den in diesem Kapitel beschriebenen Konzepten aufgebaut. Die Erklärungen in diesem Kapitel gehen, soweit wie möglich, nur auf das Prinzip eines Konzepts ein und nicht auf die technische Implementierung dessen.

5.2.1 Semimodulares Design: Processing-Steps

Um eine einfache Erweiterung der Audio Engine durch neue Funktionalitäten zu ermöglichen, basiert ein Großteil der Funktionalität auf Processing-Steps. Ein Processing-step ist eine Klasse, welche eine Eigenschaft einer Audioquelle beeinflussen bzw. steuern kann. Eine Quell-Eigenschaft beschreibt den Zustand oder das Verhalten einer Audioquelle. Dazu zählen beispielhaft: Lautstärke, Position oder der Wiedergabezustand einer Audioquelle.

Für Processing-Step Klassen gibt es eine fest definierte Schnittstelle, welche alle Processing-Steps implementieren müssen. Dadurch ist es möglich, dass neue Funktionen, die das Ziel haben, das Verhalten oder den Zustand einer Audioquelle anzupassen, relativ einfach hinzugefügt werden können, ohne dass sich ein Entwickler mit den internen Strukturen der Audio Engine auseinandersetzen muss. Interne Grundfunktionen der Audio Engine, wie z. B. das Auslesen von Dateien oder die Initialisierung von OpenAL, lassen sich über Processing-Steps nicht anpassen.

Da CosmoScout VR durch sein Plugin-System sehr modular ist, können die genauen Anforderungen an die Audio Engine und somit auch die benötigten Processing-Steps für einen Anwendungsfall sehr unterschiedlich sein. Daher sind Processing-Steps nicht global und immer verfügbar, sondern müssen vom Entwickler manuell aktiviert werden. Eine Sammlung von aktiven Processing-Steps wird als Pipeline bezeichnet. Es kann eine beliebige Anzahl an Pipelines existieren, jedoch sollte es im Idealfall für jeden Anwendungsfall der Audio Engine eine Pipeline geben. Eine Audioquelle ist immer genau einer Pipeline zugewiesen.

Die Definition einer Pipeline ist zwingend notwendig für die Nutzung der Audio Engine. Zwar können Audioquellen erstellt werden, aber selbst eine einfache Wiedergabe ist ohne dazugehörigen Processing-Step nicht möglich.

Damit ein Processing-Step eine Eigenschaft einer Audioquelle umsetzen kann, muss die Audioquelle diese Eigenschaft und den Wert dieser Eigenschaft selbst definieren. Die Definition aller Eigenschaften einer Audioquelle erfolgt über eine Zuordnungstabelle, bei der der Bezeichner die zu setzende Eigenschaft beschreibt und das dazugehörige Element den Wert definiert. Das Element kann dabei einen beliebigen Datentypen annehmen. Somit können Processing-Steps selbst bestimmen, welche Art von Daten eine Eigenschaft beschreiben. Dieses Vorgehen hat den Vorteil, dass keine Anpassungen in der internen Verarbeitung von

Quell-Eigenschaften innerhalb der Audio Engine nötig sind, wenn neue Quell-Eigenschaften hinzugefügt werden. Des Weiteren können unabhängig voneinander entwickelte Processing-Steps in einer Pipeline zusammen genutzt werden, solange diese nicht den selben Bezeichner für unterschiedliche Eigenschaften verwenden.

Eine Quell-Eigenschaft kann aber nicht nur durch eine Audioquelle definiert werden, sondern auch durch eine Gruppe oder einen Audio-Controller.

5.2.2 Öffentliche Audioobjekte: Controller, Quellen und Gruppen

Als öffentliche Audioobjekte werden die Objekte verstanden, mit denen ein Entwickler direkt arbeitet, um Audioinhalte in CosmoScout VR zu implementieren. Insgesamt gibt es 3 öffentliche Audioobjekte, die sich wie folgt zusammenfassen lassen:

1. **Audioquelle**
   Ein Objekt, welches für die Ausgabe von Audio zuständig ist.
2. **Gruppen**
   Ein Objekt, mit welchem man die Quell-Eigenschaften von mehreren Audioquellen definieren kann.
3. **Audio-Controller**
   Ein Objekt, welches die Nutzung der Audio Engine ermöglicht und für die Erstellung von Audioquellen und Gruppen zuständig ist.

Alle 3 Objekte ermöglichen die Definition von Quell-Eigenschaften. Im Fall eines Audio-Controllers und einer Gruppe sind diese Quell-Eigenschaften aber nur Werkzeuge, um Quell-Eigenschaften vereinfacht für viele Audioquellen zu definieren. Da durch dieses Vorgehen eine mehrfache Definition einer Quell-Eigenschaft möglich ist, gibt es eine Hierarchie der Quell-Eigenschaften. Es gilt: Lokal überschreibt Global. In diesem Kontext bedeutet dies, dass eine Gruppe den Audio-Controller überschreiben kann und eine Audioquelle den Audio-Controller und die Gruppe.

Audio-Controller

Der Audio-Controller ist als Eingangspunkt in die Audio Engine zu verstehen. Dieser ist somit als Erstes zu initialisieren und ermöglicht die Initialisierung aller anderen Audio-Objekte. Der Audio-Controller hat folgende Aufgaben:

1. Erstellen von Audioquellen und Gruppen.

2. Definition einer Pipeline für alle Audioquellen, die vom selben Audio-Controller erstellt wurden.

3. Initialisierung von internen Vorgängen der Audio-Engine, um die Verarbeitung von Quell-Eigenschaften zu ermöglichen.

4. Starten des Prozesses, um Quell-Eigenschaften durch die Pipeline zu verarbeiten.

Im Idealfall sollte es immer einen Audio-Controller pro Audio-Anwendungsfall geben. Der Audio-Controller kann, wie alle anderen öffentlichen Audioobjekte, Quell-Eigenschaften definieren. Diese gelten immer allen vom Audio-Controller erstellten Audioquellen, stehen aber als Letztes in der Hierarchie und können von allen anderen überschrieben werden.

Gruppe

Eine Gruppe ist das einzig optionale öffentliche Audioobjekt und hat nur eine Aufgabe:

1. Definieren von Quell-Eigenschaften auf einer Gruppe von Audioquellen.

Wichtig für das Gruppieren von Audioquellen ist, dass die Audioquelle und die Gruppe vom selben Audio-Controller stammen. Die von der Gruppe definierten Quell-Eigenschaften stehen als Zweites in der Hierarchie und können somit den Audio-Controller überschreiben und können von einer Audioquelle überschrieben werden.

Audioquelle

Eine Audioquelle stellt die Kernfunktion der Audio Engine dar. Dieses Objekt hat nur eine Aufgabe:

1. Abspielen einer Audiodatei.

Die Audioquelle definiert die in der Hierarchie höchsten Quell-Eigenschaften und kann somit alle anderen überschreiben.

Es gibt 2 Arten von Audioquellen. Beide sind nahezu identisch und unterscheiden sich nur durch die Art, wie ein *Buffer* der Audioquelle gefüllt und verwaltet wird. Die 2 Arten sind:

1. Non-*streaming*-Audioquelle
   Für Non-streaming-Audioquellen werden die Daten einer Audiodatei komplett in einen einzelnen *Buffer* geschrieben. Der *Buffer* wird

aber nicht von der Audioquelle selbst gefüllt und verwaltet, sondern von einem Buffermanager-Objekt. Der Buffermanager speichert sich alle *Buffer* und die dazugehörigen Dateien. Dies hat den Vorteil, dass, wenn zwei oder mehr Audioquellen die selbe Audiodatei ausspielen sollen, der Buffermanager allen denselben *Buffer* zur Verfügung stellen kann. Dadurch muss die Audiodatei nicht mehrfach ausgelesen und in einen *Buffer* geschrieben werden. Der Buffermanager ist global tätig, d. h. alle Audioquellen, unabhängig vom Audio-Controller, nutzen dasselbe Buffermanager-Objekt.

2. *Streaming*-Audioquelle
   Für *Streaming*-Audioquellen werden die Daten einer Audiodatei nur stückweise in mehrere *Buffer* geschrieben. Die *Buffer* werden alle von der Audioquelle selbst gefüllt und verwaltet. Dies ist vorteilhaft, wenn man große Audiodateien ausspielen muss, aber nicht die kompletten Audiodaten für die Dauer der Audioquelle in den Speicher laden möchte oder kann. Dieses Verhalten erfordert einen zusätzlichen Aktualisierungsaufruf für jeden *Frame*, um die *Buffer* der *Streaming*-Audioquellen zu aktualisieren. Dies geschieht automatisch durch die Audio Engine. Die Anzahl und Größe der einzelnen *Buffer* für eine*Streaming*-Audioquellen ist anpassbar.

Lebenszeit von öffentlichen Audioobjekten
Die Lebenszeit eines öffentlichen Audioobjekts liegt vollständig beim Entwickler, der die Audio Engine nutzt. Das bedeute, dass, wenn ein öffentliches Audioobjekt im Programmteil des Entwicklers außerhalb des Gültigkeitsbereichs gelangt, verlässt dieses Objekt auch den Gültigkeitsbereich in der Audio Engine. In der Audio Engine werden keine dauerhaften Referenzen zu einem öffentlichen Audioobjekt gespeichert, welche einen Einfluss auf die Lebenszeit eines Objekts haben.

5.2.3 Aktualisierungszyklus und das Setzen von Quell-Eigenschaften

Da ComoScout VR ein interaktives Medium ist, sind in jedem *Frame* zahlreiche Aktualisierungen erforderlich. Die Audio Engine besitzt 2 Aktualisierungszyklen, um die Funktionalität zu gewährleisten:

1. Audio-Engine-Aktualisierung
   Die Audio-Engine-Aktualisierung erfolgt automatisch und wird zu jedem *Frame* von CosmosScout VR ausgeführt. In diesem Zyklus

finden zwei Prozesse statt. Es werden alle Processing-Steps aufgerufen, die einen optionalen Aktualisierungsaufruf benötigen, und es erfolgt ein Aktualisierungsaufruf an alle *Streaming*-Audioquellen, um die fertig ausgespielten *Buffer* zu erneuern. Dieser Zyklus ist global und wird unabhängig vom Audio-Controller ausgeführt.

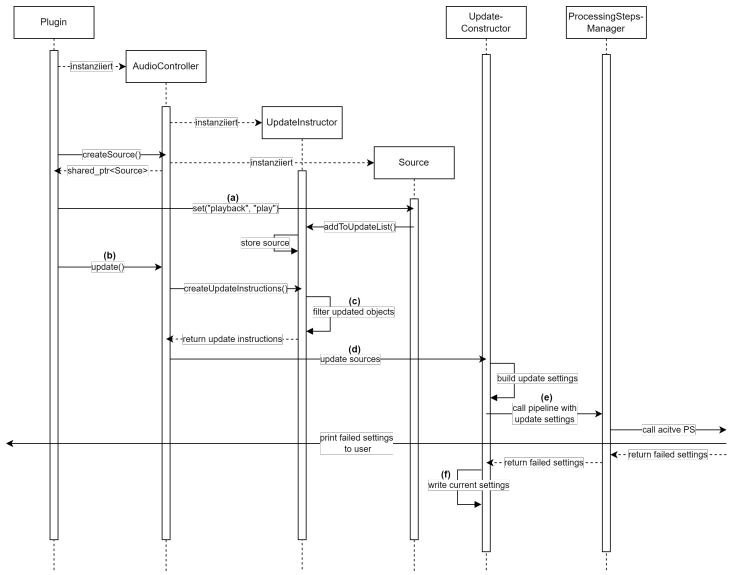

Abbildung 3: Prozess des Setzens und Aktualisierens einer Quell-Eigenschaft im Sequenz-Diagramm. Die dargestellten Buchstaben beziehen sich auf die im Kapitel 5.2.3 im Abschnitt "2. Audio-Controller-Aktualisierung"beschriebenen Prozesse.

2. Audio-Controller-Aktualisierung
   Die Audio-Controller-Aktualisierung erfolgt manuell vom Entwickler und ist nur auf den aufgerufenen Audio-Controller begrenzt. Dieser Zyklus ist für die Aktualisierung der Quell-Eigenschaften von allen öffentlichen Audioobjekten, die von diesem Audio-Controller stammen, zuständig.

   Damit eine Quell-Eigenschaft von der Definition an einem öffentlichen Audioobjekt mittels einer Audio-Controller-Aktualisierung an einen Processing-Step gelangt, sind diverse Prozesse innerhalb der Audio Engine nötig. Eine visuelle Darstellung dieses Prozesses kann in Abbildung 3 nachverfolgt werden. Der Porzess ist folgendermaßen aufgebaut:

   (a) Setzen einer Quell-Eigenschaft: Das öffentliche Audioobjekt

registriert sich an einem, für den Audio-Controller zentralen Objekt, dem UpdateInstructor.

(b) Audio-Controller-Aktualisierung: Die Aktualisierung wird gestartet und der Audio-Controller fragt bei dem UpdateInstructor alle öffentlichen Audioobjekte an, die eine Änderung der Quell-Eigenschaften seit dem letzten Aktualisierungsaufruf erhalten haben.

(c) Filterung der geänderten Objekte: Der UpdateInstructor sortiert alle Audioquellen in drei Kategorien. Diese Kategorien beschreiben, welches öffentliche Audioobjekt eine Quell-Eigenschaftenänderung für diese Audioquelle erhalten hat.

(d) Verarbeitung der Quell-Eigenschaften: Nachdem der UpdateInstructor die zu aktualisierenden Audioquellen an den Audio-Controller weitergegeben hat, werden diese an ein globales Verarbeitungsobjekt weitergegeben, dem UpdateContructor. Dieser baut für jede Audioquelle mittels der vom UpdateInstructor gegebenen Kategorie und unter Einbehaltung der Hierarchie von Quell-Eigenschaften einen Satz an Quell-Eigenschaften, welcher nur die neu zu setzenden Eigenschaften beinhaltet. Die Idee ist es in diesem Schritt, alle Quell-Eigenschaften herauszufiltern, welche entweder schon gesetzt sind oder nicht gesetzt werden dürfen, durch z. B. der Hierarchiestufe.

(e) Aufruf der Pipeline: Der UpdateConstructor nutzt den generierten Satz an Quell-Eigenschaften, um damit die Pipeline aufzurufen.

(f) Schreiben der gesetzten Werte: Nachdem die Quell-Eigenschaften von der Pipeline verarbeitet wurden, werden die gesetzten Werte für jedes öffentliche Audioobjekt in einem separaten Satz an Quell-Eigenschaften gespeichert, damit eine spätere Abfrage von den gesetzten Quell-Eigenschaften möglich ist. Die Quell-Eigenschaften, die einen Fehler in der Pipeline ausgelöst haben und somit nicht gesetzt wurden, werden in diesem Vorgang nicht mitgeschrieben.

5.2.4 Verräumlichung von Audioquellen

Ein wesentlicher Bestandteil im Funktionsumfang der Audio Engine ist die Verräumlichung von Audioquellen. Also eine dreidimensionale Darstellung von Klang, in welcher sich ein Nutzer interaktiv bewegen

kann. Da eine Positionierung einer Audioquelle eine Quell-Eigenschaft ist, muss die Verräumlichung durch ein Processing-Step implementiert werden. Bei der Implementierung müssen einige wichtige Prinzipien beachtet werden. Diese lassen sich in 2 Kategorien teilen:

1. Prinzipien, die vom Entwickler umgesetzt werden müssen:
   Der *Beobachter*, welcher immer die selbe Position besitzt wie der Hörer, befindet sich immer im Nullpunkt des Koordinatensystems, und alle Objekte, und somit auch alle Audioquellen, bewegen sich um diesen Punkt herum.

   Daraus folgt, dass eine Position immer relativ zum *Beobachter* angegeben werden muss. Die Berechnung dieser relativen Position muss vom Entwickler selbst, mittels bereits existierender Cosmoscout VR Funktionen, umgesetzt werden. Eine automatische Berechnung der realtiven Position innerhalb der Audio Engine ist nicht möglich, da es sonst zu einer *zirkulären Abhängigkeit* innerhalb der *Kernbibliotheken* von CosmoScout VR führen würde. Des Weiteren muss die relative Position manuell aktualisiert werden, wenn sich der Nutzer, und somit auch der Hörer, in der Welt von CosmoScout VR bewegt.

   Bei der Angabe einer Position ist zusätzlich noch die Skalierung des *Beobachters* zu beachten. Dieser Wert drückt die 'Größe' des Nutzers innerhalb von CosmoScout VR aus und skaliert mit der Distanz des *Beobachters* zum nächstgelegenstem Himmelsobjekt. Daraus folgt, dass eine Berechnete relative Position zum *Beobachter* um dessen Skalierung multipliziert werden muss, um die reale Distanz zu erhalten.

2. Prinzipien, die vom Processing-Step umgesetzt werden:
   Die Geschwindigkeit einer Audioquelle wird automatisch von der Audio Engine berechnet und aktualisiert.

   Eine zusätzliche Berechnung in der Position einer Audioquelle ist notwendig, wenn sich das Audioausgabegerät mit dem Nutzer mitbewegt. Dies ist z. B. der Fall, wenn CosmoScout VR über ein *Head-Mounted-Display* mit Kopfhörer ausgeführt wird. Dies führt dazu, dass sich durch eine Rotation des Head-Monted-Displays die Positionierung der Ausgabegeräte ändert. Dadurch ändert sich auch die relative Position der Audio-Ausgabegeräte zu einer Audioquelle, was wiederum zu einer verfälschten Verräumlichung führen

würde. Um dies zu kompensieren, wird die Position einer Audioquelle um die Inverse der Beobachterrotation rotiert. Um dieses Verhalten zu aktivieren, muss der Audio-Engine-Konfigurationswert 'stationaryOutputDevice' (siehe Tabelle 10) auf 'false' gesetzt werden.

Wichtig zu beachten ist, dass für eine räumliche Wiedergabe eines Buffers, dieser mit einer Mono (Einkanal) Datei befüllt wurde. Stereo (Zweikanal) Dateien lassen sich nicht verräumlichen.

5.2.5 Unterstützte Audio Formate

Die Audio Enine verwendet die Bibliothek LibSndFile, um Datein auszulesen (mehr dazu im Kaptiel 5.3.1). Zum Befüllen eines *Buffer* sind Ein- und Zweikanal .wav Dateien in den gängigen Formaten 8, 16, 24 und 32 Bit unterstützt.

5.2.6 Cluster-Modus

CosmoScout VR kann in einer Vielzahl an unterschiedlichen Umgebungen ausgeführt werden. Eine für die Audio Engine relevante ist die Ausführung im Cluster-Modus. Als Cluster-Modus wird verstanden, dass CosmoScout VR auf mehreren Instanzen gleichzeitig ausgeführt wird, wobei es eine Instanz gibt, die als Leader agiert und alle anderen als Member. Der Leader hat die Aufgabe, alle Member zu koordinieren, und die Member sind dafür verantwortlich, einen Ausschnitt des auszugebenden Bildes zu berechnen und anzuzeigen. Dies wird typischerweise in einem Mutli-Screen-Setup getan, wie z. B. in einer *CAVE*.

In diesem Fall wird die Audio Engine zwar auf jeder Instanz ausgeführt, aber die Member führen nur eine „Dummy“-Version aus. D. h. alle öffentlichen Audioobjekte, die auf den Member-Instanzen ausgeführt werden, können zwar aufgerufen werden, besitzen aber keine Funktionalität. Des Weiteren wird zu keinem Zeitpunkt ein nicht öffentliches Audioobjekt auf einen Member instanziiert. Dies passiert nur auf der Leader-Instanz, welche für die Audioausgabe zuständig ist.

5.3 Architektur und Aufbau

In diesem Kaptiel wird auf die technische Umsetzung der im vorherigen Kaptiel vorgestellten Konzepte eingegangen. Dabei handelt es sich jedoch nur um einen Überblick über das Gesamtsystem und keine detaillierte Aufschlüsselung.

5.3.1 High-Level-Übersicht

Die High-Level-Übersicht zeigt auf, wie sich die Audio Engine in das gesamte System einfügt. Eine bildliche Übersicht lässt sich in Abbildung 4 erkennen.

Die Audio Engine baut auf zwei Bibliotheken auf: OpenAL und LibSndFile. OpenAL ist für die eigentliche Audioausgabe und Verräumlichung der Audioquelle zuständig. Für die Audio Engine wird die OpenAL-Implementierung OpenAL-Soft verwendet [14]. LibSndFile ist verantwortlich, um Audiodateien auszulesen, damit deren Daten dann im richtigen Format an OpenAL für das Füllen von *Buffer*-Objekten gegeben werden können [25]. Die meisten Funktionalitäten der Audio Engine basieren auf einer der beiden Bibliotheken, und die Audio Engine abstrahiert lediglich die Nutzung dieser, um einem Entwickler eine möglichst einfache Verwendung innerhalb von ComsoScout VR zu verschaffen.

Die Audio Engine selbst ist aber nur der Kern der Funktionalität, der ohne zusätzliche Erweiterungen nicht sinnvoll nutzbar ist. Dies geschieht erst mit der Nutzung von Processing-Steps, die für die Implementierung jener Funktionalitäten verantwortlich sind, mit denen ein Entwickler die meiste Zeit verbringen wird. Der Grund für diese Teilung ist die Anpassungsfähigkeit der Audio Engine an einen Anwendungsfall und die einfache Entwicklung neuer Funktionalitäten.

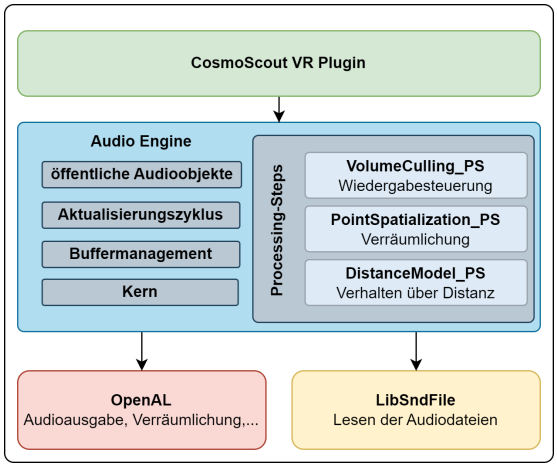

Abbildung 4: High-Level-Übersicht über den Aufbau der Audio Engine

5.3.2 Übersicht der Komponenten

Um einen tieferen Einblick in die Audio Engine zu erhalten, ohne zu sehr ins Detail gehen zu müssen, lassen sich die Klassen der Audio Engine in eine oder mehrere von 5 Komponenten einteilen: Kern, öffentliche Audioobjekte, Buffermanagement, Aktualisierungszyklus und Processing-Steps. Diese Komponenten zeigen die internen Strukturen auf, aus denen sich die Funktionsweise ableiten lässt. Um einen Überblick über die detaillierte Funktionsweise zu bekommen, befindet sich im Anhang das komplette UML-Diagramm der Audio Engine (siehe Anhang A.2). Im Text wird lediglich eine vereinfachte Version der einzelnen Komponenten aufgezeigt, die nur die Namen der einzelnen Klassen und deren Interaktion aufzeigt.

1. **Kern**
   Die Kernkomponente ist verantwortlich, um alle Prozesse zu starten, die für die Nutzung der Audio Engine notwendig sind. Dieser Teil der Audio Engine wird direkt zum Start von CosmoScout VR initialisiert und initialisiert selbst alle anderen Komponenten der Audio Engine. Der wichtigste Prozess in dieser Komponente ist das Starten aller globalen Manager-Objekte. Dadurch wird OpenAL initialisiert, eine gemeinsame Nutzung der *Buffer* durch alle Audioquellen ermöglicht und Pipelines definier- und aufrufbar gemacht.

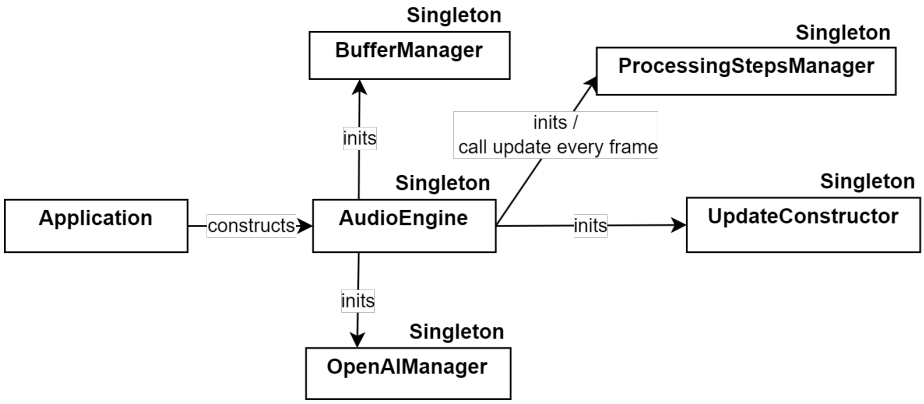

Abbildung 5: Vereinfachte Übersicht über die Core-Komponente

2. **Öffentliche Audioobjekte**
   Die öffentliche-Audioobjekte-Komponente ermöglicht die Nutzbar-

machung der Audio Engine für einen Entwickler. Dadurch wird unter anderem das Erstellen von Audioquellen, die Definition von Quell-Eigenschaften, sowie von Pipelines ermöglicht. Für eine detaillierte Beschreibung siehe Kapitel 5.2.2.

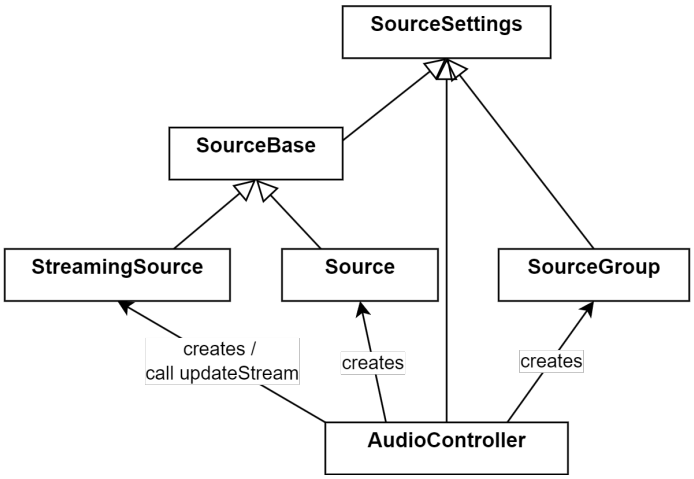

Abbildung 6: Vereinfachte Übersicht über die öffentliche-Audioobjekte-Komponente

3. **Buffermanagement**
   Die Buffermanagement-Komponente implementiert das Auslesen von Daten aus Audiodateien und das Schreiben in OpenAL *Buffer*. Des Weiteren wird die Mehrfachnutzung von *Buffer* ermöglicht und das *Streamen* von Audiodaten realisiert.

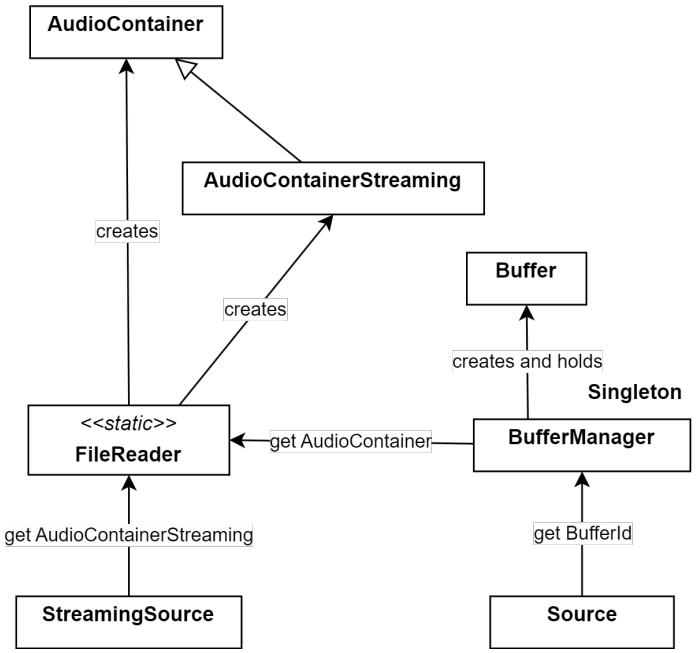

Abbildung 7: Vereinfachte Übersicht über die Buffermanagement-Komponente

4. **Aktualisierungszyklus**
   Diese Komponente implementiert den Aktualisierungszyklus der Audio Engine, um Quell-Eigenschaften zu verarbeiten. Dadurch können Aktualiserungsaufrufe registriert werden, Quell-Eigenschaften unterschiedlicher Objekte unter Einbehaltung einer festen Hierarchie zusammengeführt werden und das Weiterleiten von Quell-Eigenschaften an eine Pipeline wird ermöglicht. Für eine detaillierte Übersicht siehe Kaptiel 5.2.3.

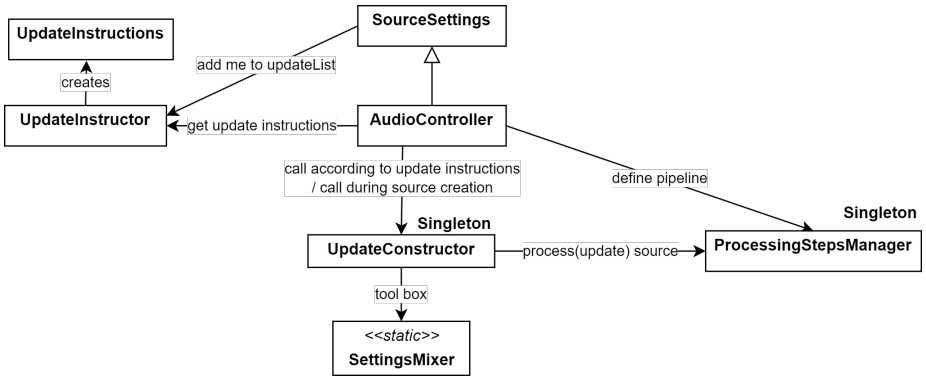

Abbildung 8: Vereinfachte Übersicht über die Aktualisierungszyklus-Komponente

5. **Processing-Steps**
   Diese Komponente stellt den modularen Teil der Audio Engine dar. Dadurch können Processing-Steps aufgerufen und Pipelines registriert werden. In der bildlichen Übersicht 9 sind die bereits vorhandenen Processing-Steps zu erkennen, welche in Kaptiel 5.4 erläutert werden.

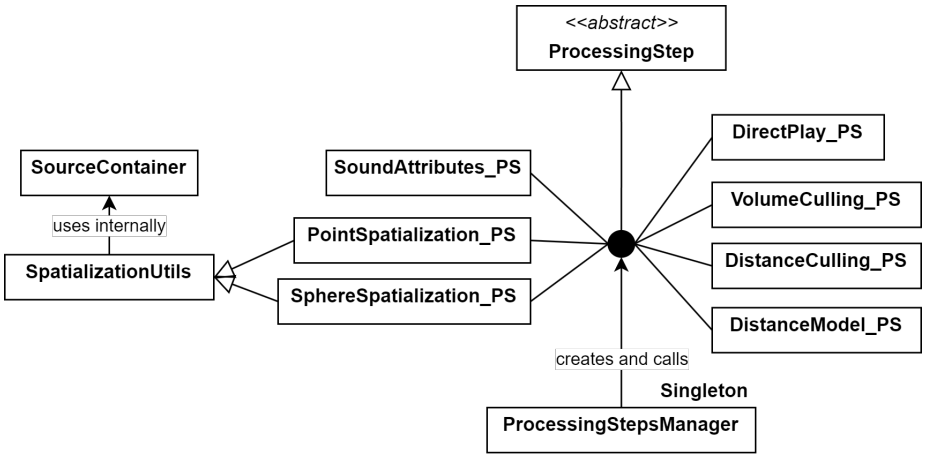

Abbildung 9: Vereinfachte Übersicht über die Processing-Steps-Komponente

5.4 Vorhandene Processing-Steps

In diesem Kapitel werden die bereits vorhandenen Processing-Steps und die damit vorhandenen Quell-Eigenschaften vorgestellt. Insgesamt gibt es 7 Processing-Steps:

1. **SoundAttributes_PS**
   Dieser Processing-Step ermöglicht es, einfache Klangmerkmale einer Audioquelle zu setzen.

Name	Typ	Range	Beschreibung
gain	float	R^+	Multiplikator für die Lautstärke.
pitch	float	0.5 - 2.0	Multiplikator für die Sample Rate des *Buffers*.
looping	bool		Ob sich die Quelle nach abspielen des *Buffers* wiederholt oder die Wiedergabe stoppt

Tabelle 3: Übersicht über die Eigenschaften von SoundAttributes_PS

2. **DirectPlay_PS**
   DirectPlay_PS lässt sich in die Kategorie der Wiedergabesteuerungs-Processing-Steps einteilen, also Processing-Steps, die eine Audioquelle abspielen, pausieren oder stoppen können. DirectPlay_PS ist der einfachste in dieser Kategorie und setzt einen definierten Zustand direkt um.

Name	Typ	Range	Beschreibung
playback	std::string	'play' 'stop' 'pause'	Wiedergabezustand

Tabelle 4: Übersicht über die Eigenschaften von DirectPlay_PS

3. **DistanceCulling_PS**
   DistanceCulling_PS ist ein Wiedergabesteuerungs-Processing-Step. Dieser setzt den Zustand jedoch nicht direkt um, sondern macht dies abhängig von der Distanz zwischen dem *Beobachter* und einer Audioquelle. Wenn der Zustand auf „play“ gesetzt wird, wird bei jeder Änderung der Position einer Audioquelle die Distanz zum *Beobachter* berechnet. Wenn diese unter dem definierten Grenzwert liegt, wird die Audioquelle abgespielt. Andernfalls wird diese pausiert. Der Grenzwert wird in der Konfigurationsdatei unter der Bezeichnung „distanceCullingThreshold“ global für alle Audioquellen definiert.

Name	Typ	Range	Beschreibung
playback	std::string	'play' 'stop' 'pause'	Wiedergabezustand

Tabelle 5: Übersicht über die Eigenschaften von DistanceCulling_PS

4. **VolumeCulling_PS**
   VolumeCulling_PS ist ein Wiedergabesteuerungs-Processing-Step. Der Wiedergabezustand wird nicht direkt umgesetzt, sondern ist abhängig von der Lautstärke einer Audioquelle an der Position des *Beobachters*. Wenn der Zustand auf „play“ gesetzt wird, wird bei jeder Änderung der Position einer Audioquelle die theoretische Lautstärke für den Hörer berechnet. Diese Information ist nicht von OpenAL erhältlich, sondern wird vom Processing-Step berechnet. Wenn die Lautstärke über dem definierten Grenzwert liegt, wird die Audioquelle abgespielt. Andernfalls wird diese gestoppt. Der Grenzwert wird in der Konfigurationsdatei unter der Bezeichnung „volumeCullingThreshold“ global für alle Audioquellen definiert.

Name	Typ	Range	Beschreibung
playback	std::string	'play' 'stop' 'pause'	Wiedergabezustand

Tabelle 6: Übersicht über die Eigenschaften von VolumeCulling_PS

5. **PointSpatialization_PS**
   PointSpatialization_PS lässt sich in die Kategorie der Verräumlichungs-Processing-Steps einteilen. Dies sind Processing-Steps, die eine Audioquelle einer Position im dreidimensionalen Raum zuweisen können. Die Audioquelle wird in diesem Fall als einzelner Punkt verräumlicht, ohne Volumen. Die Geschwindigkeit der Audioquelle wird automatisch berechnet.

Name	Typ	Range	Beschreibung
position	glm::dvec3		Position der Quelle relativ zum *Beobachter*

Tabelle 7: Übersicht über die Eigenschaften von PointSpatialization_PS

6. **SphereSpatialization_PS**

SphereSpatialization_PS ist ebenfalls ein Verräumlichungs-Processing-Step, welcher eine Audioquelle jedoch nicht als Punkt definiert, sondern als Kugel. Wenn sich der Hörer innerhalb der Kugel befindet, ist keine Verräumlichung wahrzunehmen, denn mittels dieses Processing-Steps wird eine Audioquelle als Volumen dargestellt. Nur wenn sich der Hörer außerhalb der Kugel befindet, ist eine Richtung des Klangs hörbar. Falls die Kugel einen großen Radius besitzt, ist es möglich, dass nach Verlassen der Kugel ein sehr plötzlicher Abfall der Lautstärke zu vernehmen ist. Um dies zu vermeiden, kann mittels DistanceModel_PS der Start des Lautstärkeabfalls auf den selben Wert wie der Radius der Kugel gesetzt werden.

Name	**Typ**	**Range**	**Beschreibung**
position	glm::dvec3		Position der Quelle relativ zum *Beobachter*
radius	float	R^+	Radius der Kugel

Tabelle 8: Übersicht über die Eigenschaften von SphereSpatialization_PS

7. **DistanceModel_PS**
   Mittels des DistanceModel_PS lässt dich das Verhalten einer Audioquelle über die Distanz anpassen. Um diese Funktion zu nutzen, muss ein 'Verräumlichungs Processing-Step' ebenfalls aktiv sein.

Name	Typ	Range	Beschreibung
distanceModel	std::string	'inverse' 'linear' 'exponent'	Definiert die Form des Lautstärkeabfalls über die Distanz
fallOffStart	float	R^+	Distanz, bei welcher der Lautstärkeabfall beginnt. Darunter wird die Quelle mit voller Lautstärke verräumlicht ausgespielt.
fallOffEnd	float	R^+	Distanz, bei welcher der Lautstärkeabfall begrenzt wird. Dies bedeutet nicht, dass die Audioquelle aufhört zu spielen, sondern, dass es keinen weiteren Abfall gibt und die Lautstärke hinter diesem Grenzwert konstant bleibt.
fallOffFactor	float	R^+	Multiplikator der Distanzdämpfung. Keine Dämpfung, wenn der Wert auf 0.0 gesetzt wird.

Tabelle 9: Übersicht über die Eigenschaften von DistanceModel_PS

Wie zu einigen Processing-Steps zu lesen ist, haben sich bereits unterschiedliche Implementierungen gleicher Eigenschaften gebildet (Verräumlichungs-Processing-Steps und Wiedergabesteuerungs-Processing-Steps). Dies stellt einen weiteren Vorteil des modularen Designs dar, dass es sehr einfach sein kann, unterschiedliche Arten der selben Funktion zu entwickeln, ohne dass eine Implementierung Rücksicht auf die anderen nehmen muss. Des Weiteren kann eine Implementierung sehr einfach gegen eine andere ausgetauscht werden, wie im nächsten Kaptiel (Kaptiel 5.6) zu sehen ist. Dabei ist jedoch zu beachten, dass es pro Kategorie nur einen Processing-Step geben sollte, da ansonsten beide Processing-Steps aktiv werden und ein Processing-Step den anderen jederzeit überschreiben würde.

5.5 Konfiguration der Audio Engine

CosmoScout VR bietet viele Stellschrauben, um die Software an einen Anwendungsfall anzupassen. Diese Anpassung geschieht über eine Konfigurationsdatei im *JSON-Format*. Wie anderen Komponenten von CosmoScout VR bietet auch die Audio Engine diverse Einstellungen, um das Programm an ein System oder einen Anwendungsfall anzupassen. In der Tabelle 10 ist eine Übersicht über die möglichen Einstellungen zu sehen.

Name der Einstellung	Typ	Standard	Beschreibung
enableHRTF	bool	true	Ob *HRTF* verwendet werden soll.
numberMonoSources	int	512	Die Maximale Anzahl an Mono-Quellen.
numberStereoSources	int	5	Die Maximale Anzahl an Stereo-Quellen.
refreshRate	int	30	Die Aktualisierungsrate des OpenAL-Kontexts pro Sekunde
contextSync	bool	false	Ob der OpenAL-Kontext *synchron oder asynchron* zu CosmoScout VR arbeiten soll.
mixerFrequency	int	48.000	Die Frequenz zum mixen des OpenAL Ausgabe *Buffers*. Angegeben in Hz.
volumeCullingThreshold	float	0,01	Die Lautstärke, bei welcher eine Audioquelle aufhört zu spielen, wenn VolumeCulling_PS aktiv ist.
distanceCullingThreshold	double	100,0	Die Reale Distanz, bei welcher eine Audioquelle aufhört zu spielen, wenn DistanceCulling_PS aktiv ist.
stationaryOutputDevice	bool	true	Ob das Ausgabegerät sich bei Bewegung des Nutzers mitbewegt oder stationär bleibt.

Tabelle 10: Übersicht über die Konfigurationsmöglichkeiten der Audio Engine

5.6 Beispiele

In diesem Kapitel folgen zwei Beispiele, wie die Audio Engine genutzt werden kann. Aus diesen sollte hervorgehen, wie man mittels der Audio Engine Audio Inhalte in CosmoScout VR implementiert.

5.6.1 Einfaches Beispiel

Dieses Beispiel zeigt die einfachste mögliche Nutzung der Audio Engine. Es wird eine Audioquelle erzeugt und einmal abgespielt:

```
mController = mAudioEngine->createAudioController();
 ↪  // Instanziert einen Audio-Controller, um
 ↪  Zugriff auf die Audio Engine zu bekommen.

mController->setPipeline(std::vector<std::string>{"D
 ↪  irectPlay"}); // Definiert die Pipeline. Es wird
 ↪  nur ein Wiedergabesteuerungs-Processing-Step
 ↪  definiert, um die Quelle abzuspielen.

mSource = mController->createSource("testFile.wav");
 ↪  // Instanziiert eine Audioquelle mit der Datei
 ↪  testFile.wav.

mSource->play(); // Setzt eine Quell-Eigenschaft,
 ↪  damit die Audioquelle bei der nchsten
 ↪  Audio-Controller-Aktualisierung gestartet wird.

mController->update(); //
 ↪  Audio-Controller-Aktualisierung. In diesem Fall
 ↪  wird DirectPlay_PS aktiv und startet die
 ↪  Wiedergabe von mSource.
```

5.6.2 Komplexeres Beispiel mit Verräumlichung

Dieses Beispiel zeigt einen realistischeren Einsatz der Audio Engine. In diesem Fall werden eine Reihe an Audioquellen erzeugt, welche mit ihren Eigenschaften in der CosmoScout VR Konfigurationsdatei definiert sind. Für jede Quelle ist eine Position und eine Distanz zum Start des Lautstärkeabfalls definiert. Optional definieren einige Quellen auch eine Pitch-Eigenschaft. Falls eine Audioquelle keinen Pitch definiert, wird dieser Wert durch eine Gruppe gesetzt. Damit die Audioquellen korrekt verräumlicht werden, wird die Position der Audioquellen im Aktualisierungsaufruf, welcher jeden *Frame* eintritt, berechnet.

Beispielhafte Struktur, wie Audioquellen in der Konfigurationsdatei definiert sein können:

```
"sources": [
    {
```

```
        "location": {
            "center": "Earth",
            "position": [
                -588086.8558471624,
                3727313.5198930562,
                10001091.473068066
            ]
        },
        "file": "C:/testFile1.wav",
        "fallOffStart": 500000
    },
    {
        "location": {
            "center": "Earth",
            "position": [
                188086.8558471624,
                8727313.5198930562,
                -4021091.473068066
            ]
        },
        "file": "C:/testFile2.wav",
        "fallOffStart": 1000000,
        "pitch": 1.5
    },
    ...
]
```

In diesem Codeabschnitt erfolgt die Initialisierung der Audioszene. Es werden alle Audioquellen erzeugt und die Quell-Eigenschaften werden gesetzt:

```
mController = mAudioEngine->createAudioController();

mController->set("looping", true); // Laesst die
 ↪  Wiedergabe aller Audioquellen, die von diesem
 ↪  Audio-Controller stammen, automatisch
 ↪  wiederholen.

mController->setPipeline(std::vector<std::string>{"D
 ↪  irectPlay", "SoundAttributes", "DistanceModel",
 ↪  "PointSpatialization"});  // Definiert die
 ↪  Pipeline. SoundAttributes wird fuer den Pitch
 ↪  benoetigt, DistanceModel fuer die Startdistanz
 ↪  des Lautstaerkeabfalls und PointSpatialization
 ↪  um die Audioquelle im Raum zu definieren.
```

```
6
7   mGroup = mController->createSourceGroup(); //
     ↪  Erstellt eine Gruppe.
8
9   mGroup->set("pitch", 0.8f); // Setzt den Pitch fuer
     ↪  alle Mitglieder auf 0.8.
10
11  // Erstellt eine Audioquellen pro Eintrag in der
     ↪  Konfigurationsdatei und setzt deren
     ↪  Eigenschaften:
12  for (auto& source : mPluginSettings->sources) {
13    source.instance =
       ↪  mController->createSource(source.file);
14    source.instance->set("fallOffStart",
       ↪  (float)source.fallOffStart); // Setzt die
       ↪  Startdistanz des Lautstrkeabfalls
15    if (source.pitch) {
16      source.instance->set("pitch",
         ↪  (float)source.pitch); // Ueberschreibt den
         ↪  Gruppen-Pitch, falls dieser in der
         ↪  Konfigurationsdatei vorhanden ist
17    }
18    source.instance->play();
19    mGroup->join(source.instance); // Fuegt eine
       ↪  Audioquelle der Gruppe hinzu.
20  }
21
22  mController->update(); // Aufruf der Pipeline. In
     ↪  diesem Fall werden SoundAttributes,
     ↪  DistanceModel und DirectPlay aktiv.
```

Dieser Abschnitt zeigt den Aktualisierungsaufruf, um die Position einer Audioquelle zu berechnen:

```
1   float observerScale = static_cast<float>(mSolarSyste
     ↪  m->getObserver().getScale()); // Die
     ↪  Beobachterskalierung wird benoetig, um die
     ↪  Distanz von einer Audioquelle auf die reale
     ↪  Distanz zu skalieren.
```

```
2
3  for (auto& source : mPluginSettings->sources) {
4
5   // Berechnung der Position einer Audioquelle
     ↪  relativ zum Beobachter:
6   auto celesObj =
     ↪  mSolarSystem->getObject(sphere.location.center);
7   if (celesObj == nullptr) { continue; }
8
9   glm::dvec3 sourcePosToCelesObj(sphere.location.posi
     ↪  tion[0],sphere.location.position[1],
     ↪  sphere.location.position[2]);
10  glm::dvec3 sourcePosToObserver = celesObj->getBeoba
     ↪  chterRelativePosition(sourcePosToCelesObj);
11  sourcePosToOberver *= observerScale;
12
13  source.instance->set("position",
     ↪  sourcePosToObserver); // Berechnete Position
     ↪  wird fuer die Audioquelle gesetzt.
14 }
15
16 mController->update(); // Aufruf der Pipeline, damit
   ↪  PointSpatialization aktiv wird und alle
   ↪  Positionen aktualisiert werden.
```

6 Evaluation

6.1 Performancemessung

Da die Performance der Audio Engine ein wesentlicher Bestandteil der Beurteilung ist, wurde für die Testung das CosmoScout VR Plugin csp-audio-test entwickelt. Mit diesem ist eine schrittweise Erhöhung der Anzahl an Audioquellen möglich. Der daraus resultierende Einfluss auf die Performance wird mit dem bereits vorhandenen CosmoScout VR Plugin csp-timings gemessen.

Der Test ist folgendermaßen aufgebaut: Audioquellen werden schrittweise zufällig auf der Erde verteilt und wiedergegeben. Alle Audioquellen sind Non-Streaming, spielen die selbe Audiodatei aus, erhalten ei-

ne Position mittels PointSpatialization_PS und definieren einen einen FallOffStart und FallOffFactor Wert mittels DistanceModel_PS (siehe Kapitel 5.4). Der *Beobachter* bleibt während des Tests stationär.

Für die Evaluation der Performance ergeben sich vier relevante Grenzwerte:

1. **Grenzwert für fehlerfreie Audioausgabe mit *HRTF***
   Dieser Wert stellt das Limit an gleichzeitig spielenden Audioquellen dar, welche durch OpenAL fehlerfrei wiedergegeben werden können, wenn *HRTF* aktiviert ist.

2. **Grenzwert für fehlerfreie Audioausgabe ohne *HRTF***
   Dieser Wert stellt das Limit an gleichzeitig spielenden Audioquellen dar, welche durch OpenAL fehlerfrei wiedergegeben werden können, wenn *HRTF* deaktiviert ist.

3. **Grenzwert 60 *Frames* pro Sekunde (FPS)**
   Dieser Wert stellt das Limit an gleichzeitig spielenden Audioquellen dar, mit denen die Wiedergabe von CosmoScout VR mit 60 *Frames* pro Sekunde noch möglich ist.

4. **Grenzwert 30 *Frames* pro Sekunde (FPS)**
   Dieser Wert stellt das Limit an gleichzeitig spielenden Audioquellen dar, mit denen die Wiedergabe von CosmoScout VR mit 30 *Frames* pro Sekunde noch möglich ist.

Die ersten beiden Grenzwerte werden durch den Leistungsbedarf von OpenAL limitiert, während die anderen beiden Grenzwerte durch den Leistungsbedarf der Audio Engine begrenzt sind.

Die Ermittlung der Grenzwerte eins und zwei erfolgt über ein Hörprobe während der Durchführung des beschriebenen Tests. Die Grenzwerte drei und vier werden durch die ermittelten Werte von csp-timings erschlossen. Die Durchführung der Tests der unterschiedlichen Grenzwerte erfolgt nicht gleichzeitig, sondern nacheinander.

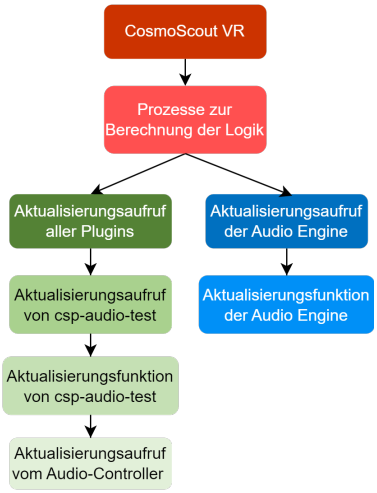

Abbildung 10: Darstellung der Baumstruktur aller audiorelevanten Prozesse innerhalb von CosmoScout VR.

Der Testaufbau wird in drei unterschiedlichen Konfigurationen umgesetzt. Diese sind identisch, aber unterscheiden sich in der Wahl des Wiedergabesteuerungs-Processing-Steps, da diese einen Einfluss auf die maximale Anzahl an Audioquellen haben können. Es werden alle zur Verfügung stehenden Wiedergabesteuerungs-Processing-Steps getestet, also DirectPlay_PS, DistanceCulling_PS und VolumeCulling_PS.

Um die ermittelten Resultate, wie z. B. in Abbildung 11, interpretieren zu können, ist ein Verständnis für die im Diagramm dargestellten Prozesse notwendig. In den Diagrammen wird sowohl aufgezeigt, wie hoch die benötigte Gesamtzeit für die Berechnung eines *Frames* ist, als auch die Aufteilung der Zeit in die audiorelevanten Prozesse. Die dargestellten Prozesse können als eine Baumstruktur interpretiert werden. Eine visuelle Darstellung dieser Struktur mit allen audiorelevanten Prozessen ist in Abbildung 10 erkennbar. Ein Kind-Prozess wird immer vom Eltern-Prozess aufgerufen und die Zeit, die von einem Kind-Prozess benötigt wird, ist Teil der Gesamtzeit eines Eltern-Prozesses.

Zur Durchführung der Tests wurde das selbe System wie aus Tabelle 1 verwendet.

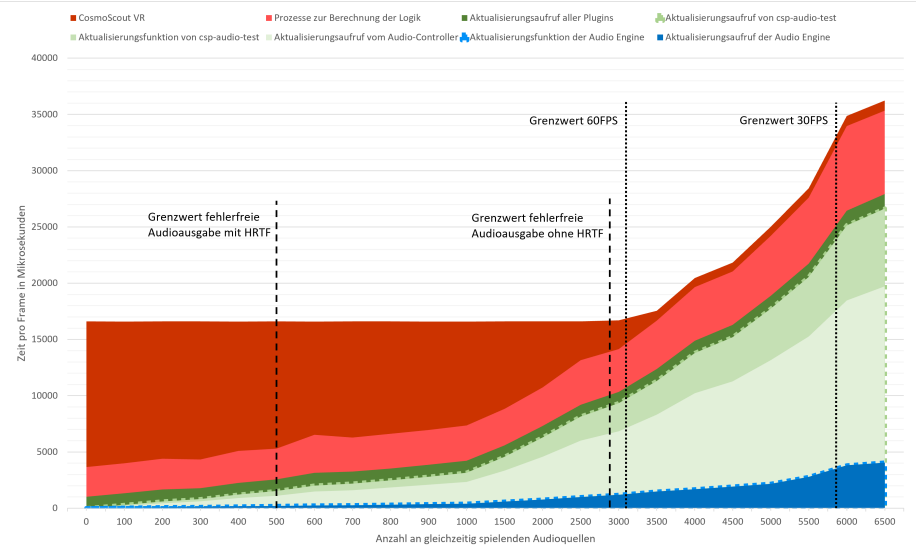

Abbildung 11: Performance Messung mit DirectPlay_PS. Zu beachten ist, dass die horizontale Achse nicht linear ist. Ab dem Wert 1000 wird die Schrittweite von 100 auf 500 erhöht.

In Abbildung 11 ist die Performance Messung unter der Verwendung von DirectPlay_PS zu sehen. Die Grenzwerte für diese Konfiguration lauten:

Fehlerfreie Ausgabe mit *HRTF*	ca. 500 Audioquellen
Fehlerfreie Ausgabe ohne *HRTF*	ca. 2900 Audioquellen
60 FPS	ca. 3200 Audioquellen
30 FPS	ca. 5600 Audioquellen

Tabelle 11: Performance Grenzwerte für DirectPlay_PS

Im Diagramm der Abbildung 11 lässt sich der Gesamtanstieg der benötigten Zeit pro *Frame* hauptsächlich durch 3 Prozesse erklären: der Aktualisierungsfunktion der Audio Engine (blau), der Aktualisierungsfunktion von csp-audio-test (mittleres grün) und des Aktualisierungsaufrufes des Audio-Controllers (helles grün).

Der Anstieg durch die Aktualisierungsfunktion der Audio Engine erfolgt, da durch diesen Prozess die Geschwindigkeiten aller Audioquellen berechnet werden.

Der Anstieg der Aktualisierungsfunktion von csp-audio-test wird hervorgerufen, da während dessen Ausführung alle Positionen der Audio-

quellen berechnet werden. Diese Berechnung muss jeden *Frame* durchgeführt werden und stellt bei einer großen Anzahl an Audioquellen keinen trivialen Anteil dar.

Der Anstieg durch den Aktualisierungsaufruf des Audio-Controllers erfolgt, da in diesem Prozess die Pipeline zur Aktualisierung aller Quell-Eigenschaften aufgerufen wird. In diesem Fall wird nur die Position einer Audioquelle durch die Pipeline gesetzt.

Da der Aktualisierungsaufruf des Audio-Controllers die längste Zeit in der Berechnung eines *Frames* einnimmt, lässt sich daraus schließen, dass sich dort das größte Optimierungspotenzial verbirgt. Allerdings würde eine Optimierung in diesem Bereich nicht zwingend eine bessere Leistung der Audio Engine hervorrufen, da der durch diesen Prozess ausgelöste Grenzwert (Grenzwert für 60 FPS) erst nach den Grenzwerten für die fehlerfreie Audioausgabe einsetzt. Somit sollte sich eine erste Optimierung auf die Grenzwerte der fehlerfreien Audioausgabe konzentrieren. Ein Ansatz, wie eine solche Optimierung aussehen kann, ist die Verwendung der Processing-Steps DistanceCulling_PS oder VolumeCulling_PS.

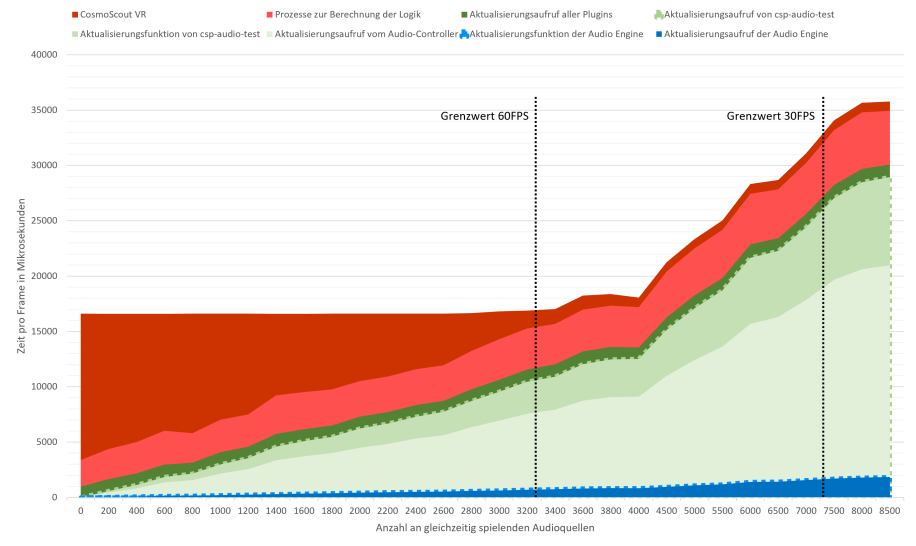

Abbildung 12: Performance Messung mit DistanceCulling_PS. Zu beachten ist, dass die horizontale Achse nicht linear ist. Ab dem Wert 4000 wird die Schrittweite von 200 auf 500 erhöht. Die Grenzdistanz für DistanceCulling_PS beträgt 300000.0.

In Abbildung 12 und Tabelle 12 lässt sich das Testresultat bei der

Nutzung des Processing-Steps DistanceCulling_PS erkennen.

60 FPS	ca. 3300 Audioquellen
30 FPS	ca. 5800 Audioquellen

Tabelle 12: Performance Grenzwerte für DistanceCulling_PS

Die Grenzwerte für eine fehlerfreie Audioausgabe sind bei der Verwendung von DistanceCulling_PS nicht allgemein ermittelbar. Dies liegt daran, dass durch den Processing-Step nur die Audioquellen ausgespielt werden, die innerhalb eines definierten Radius liegen und vom Entwickler zur Wiedergabe aktiviert wurden. Dadurch hängt der Grenzwert für die fehlerfreie Audioausgabe mit der Konfiguration des Radius und der Dichte an Audioquellen zusammen. Der Grenzwert wird somit erst erreicht, wenn innerhalb des Radius mehr Audioquellen spielen, als die Grenzwerte für die fehlerfreie Ausgabe aus Tabelle 11 angeben.

Im durchgeführten Test konnte dieses Limit bei einer Erhöhung von bis zu 20.000 Audioquellen und einem Distanzgrenzwert von 300000.0, nicht erreicht werden. Dabei ist jedoch zu beachten, dass die Audioquellen zufällig verteilt wurden, und nicht sichergestellt wurde, wie viele Audioquellen zu einem Zeitpunkt tatsächlich durch DistanceCulling_PS abgespielt werden.

Dennoch lässt sich eine signifikante Optimierung feststellen, da durch die Nutzung von DistanceCulling_PS die Grenzwerte für die fehlerfreie Audioausgabe unter den richtigen Umständen irrelevant werden.

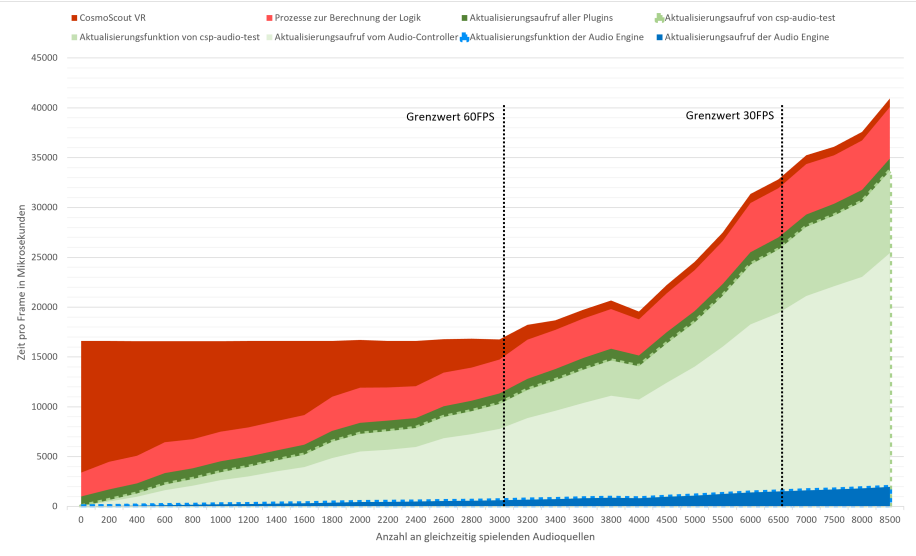

Abbildung 13: Performance Messung mit VolumeCulling_PS. Zu beachten ist, dass die horizontale Achse nicht linear ist. Ab dem Wert 4000 wird die Schrittweite von 200 auf 500 erhöht. Die Grenzlautstärke für VolumeCulling_PS beträgt: 0.01

In Abbildung 13 und Tabelle 13 lässt sich das Testresultat bei der Nutzung des Processing-Steps VolumeCulling_PS erkennen.

60 FPS	ca. 3000 Audioquellen
30 FPS	ca. 5800 Audioquellen

Tabelle 13: Performance Grenzwerte für DistanceCulling_PS

Auf VolumeCulling_PS treffen ebenso alle genannten Punkte wie von DistanceCulling_PS zu. Dabei ist jedoch erkennbar, dass dieser Processing-Step einen höheren Leistungsbedarf hat, da zusätzlich zur Berechnung der Distanz auch eine Berechnung der Lautstärke notwendig ist.

Daraus lässt sich schließen, dass, wenn die Distanz bekannt ist, wie weit eine Audioquelle spielen soll, DistanceCulling_PS stets bevorzugt werden sollte. Falls eine allgemeinere Lösung notwendig ist, ist VolumeCulling_PS zu empfehlen.

6.2 Erfüllung der Anforderungen

In den vorangegangenen Kapiteln wurden die grundlegenden Konzepte, deren Implementierung, Nutzung und die Messung der Nutzung dar-

geboten, mit dem Ziel, die im Kapitel 5.1 definierten Anforderungen zu erfüllen. Die Einhaltung dieser Anforderungen lässt sich wie folgt zusammenfassen:

1. **Möglichst viele Audioquellen im Raum gleichzeitig abspielen:**
   Wie im Kapitel 6.1 aufgezeigt wird, ist eine Darstellung von mehr als 3000 gleichzeitig spielenden Audioquelle möglich. Dieses Limit ist somit höher, als OpenAL ohne zusätzliche Hilfe auf dem Testsystem fehlerfrei ausspielen kann.

2. **Eine einfache Schnittstelle für Entwickler:**
   Eine leicht zugängliche Schnittstelle wird ermöglicht, indem sich ein Entwickler lediglich mit zwei Klassen befassen muss, um einen Klang zu erzeugen. Eine dieser Klassen, die Audioquelle, repräsentiert ein sehr konkretes Konzept. Daher ist es erforderlich, lediglich das Prinzip der Processing-Steps und Pipelines zu verstehen, um ein Verständnis für den Großteil der Funktionalität der Audio Engine zu erhalten.

3. **Unterstützung von alle Plattformen, die auch CSVR unterstützt:**
   Im Laufe der Entwicklung und Testung wurde die Audio Engine auf diversen Systemen ausgeführt. Darunter fallen Desktop (Windows, Linux), *Head-Mounted-Display* (Windows) und eine *CAVE* (Linux). Dabei wurde sowohl mit stationären Kopfhörern, mitbewegenden Kopfhörern und Lautsprecherkonfigurationen gearbeitet. Dies hat gezeigt, dass die Audio Engine auf allen notwendigen Systemen eingesetzt werden kann. Ein Problem, welches aber aufgetreten ist und noch nicht behoben werden konnte, ist, dass eine Änderung des Ausgabegerätes auf Linux in der jetzigen Version nicht möglich ist und somit nur das vom System ausgewählte Standardgerät verwendet werden kann. Anderweitig ist die Ausführung unter Windows und Linux aber identisch.

4. **Einfache Möglichkeit, um die Audio Engine mit neuen Funktionalitäten zu erweitern:**
   Eine einfache Erweiterung der Audio Engine ist durch die Verwendung von Processing-Steps gegeben. Durch diese können neue Quell-Eigenschaften hinzugefügt werden, ohne dass sich ein Entwickler mit den internen Strukturen der Audio Engine auseinandersetzen muss.

7 Sonifikations-Plugin

Um die Audio Engine praktisch zu testen und um einen konzeptionellen Beweis zu geben, dass eine Sonifikation mit der Audio Engine möglich ist, wurde ein Sonifikations-Plugin für CosmoScout VR entwickelt. Dieser dient zusätzlich als praktisches Beispiel für andere Entwickler, wie man das Audiosystem verwenden kann. Das Plugin ist nicht Teil des Kernumfangs dieser Arbeit. Aus Zeitgründen konnte das Sonifikations-Plugins nicht fertiggestellt werden, aber das verfolgte Prinzip ist dennoch erkennbar, oder besser gesagt: hörbar.

Die Arbeit in diesem Abschnitt erfolgte nicht alleine, sondern ist in Zusammenarbeit mit Peter Michael von der Nahmer vom Institut für Flugführung am Deutschen Zentrum für Luft- und Raumfahrt e. V. entstanden. Herr von der Nahmer hat insbesondere bei der Gestaltung des Gesamthörbildes, also wie das Endergebnis akustisch ausgedrückt wird, geholfen und ist verantwortlich für die Erstellung der ausgespielten Klänge.

7.1 Der verwendete Datensatz

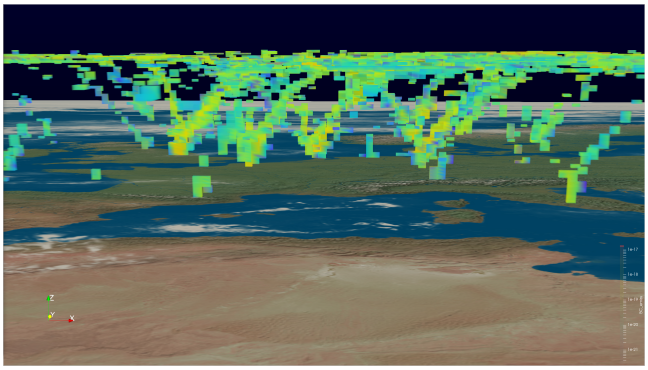

Abbildung 14: Visuelle Darstellung des verwendeten Datensatzes. Die Skalierung der Daten ist in der Höhe nicht maßstabsgetreu. Quelle: Deutsches Zentrum für Luft- und Raumfahrt e.V.

Das Sonifikations-Plugin bietet keine allgemeine Methode, um Daten zu verklanglichen, sondern ist speziell auf einen Datensatz angepasst. Der verwendete Datensatz ist eine dreidimensionale Datensammlung über die Emissionswerte des Luftverkehrs auf der Erde. Der Datensatz stammt vom Institut für Physik der Atmosphäre am Deutschen Zentrum

für Luft- und Raumfahrt e.V. und wurde wegen dessen Dreidimensionalität und Verfügbarkeit ausgewählt. Eine Visualisierung des Datensatzes ist in Abbildung 14 zu sehen.

Insgesamt umfasst die Datensammlung circa 50 Millionen Werte, die gleichmäßig über die Erde verteilt sind, bis zu einer Höhe von circa 14,6 km. Von den 50 Millionen Werten sind rund 600.000 Werte nicht Null und somit für die Sonifikation relevant.

7.2 Funktionsweise

Das Ziel des Sonifikations-Plugins ist eine interaktive akustische Erkundung der Luftverkehrsemissionswerte auf der Erde. Um dies zu erreichen, soll sich ein Nutzer frei über die Erde bewegen können und zu jedem Zeitpunkt die Emissionswerte um sich herum hören können.

7.2.1 Verarbeitung der Daten und Erstellung einer Audioszene

Um das definierte Ziel zu erreichen, müssen zwei grundlegene Problemstellungen gelöst werden: Die Verarbeitung der Daten und wie diese Daten durch eine Audioszene dargestellt werden.

Unter der Verarbeitung der Daten wird verstanden, wie der Datensatz bei einer interaktiven Nutzung ausgelesen wird. Dies ist bei einer Datengröße von 50 Millionen Werte keine triviale Fragestellung, da die Performance einen wichtigen Faktor bei einer interaktiven Nutzung darstellt, besonders wenn die Sonifikation auf einem *stereoskopischen Display* ausgeführt wird, da dieser Fall mindestens 60 Frames pro Sekunde benötigt.

Die Audioszene hingegen beschreibt, wie die Verklanglichung durch Platzierung von Audioquellen in der Welt realisiert wird.

Beide Problemstellungen sind voneinander abhängig. Eine Änderung der einen Komponente hat oftmals auch einen Einfluss auf die andere. Im Verlaufe der Entwicklung wurden 3 Ansätze verfolgt, wie die Sonifikation umgesetzt werden kann:

1. **Erdstationäre Audioquellen**
   Der erste und einfachste verfolgte Ansatz sind die erdstationären Audioquellen. Die Idee war es, über den Planeten Audioquellen zu platzieren und als Eingangswert für den ausgespielten Klang den Emissionswert an der Position der Audioquelle zu wählen. Dieser Ansatz ist aber aufgrund der Größe des Datensatzes nicht umsetzbar. Selbst wenn alle Nullwerte der Datensammlung entfernt

werden, sind es 600.000 Werte und somit Audioquelle, die dargestellt werden müssten. Eine solche Anzahl an Audioquellen liegt nicht im Funktionsumfang der Audio Engine.

Eine Reduktion der Anzahl der Emissionswerte, durch z. B. das Zusammenfassen von nahe zueinander liegenden Werten, ist möglich, aber es ist fraglich, ob nach einer Reduktion, die im Bereich der Audio Engine liegt, eine sinnvolle Sonifikation wahrnehmbar ist.

2. **Beobachterstationäre Klangkugel**
   Ein Ansatz, der die Limitierung der Erdstationäre Audioquellen zu umgehen versucht, ist die Beobachterstationäre Klangkugel. Das Prinzip sieht eine Kugel an Audioquellen vor, die sich zu jedem Zeitpunkt um den *Beobachter* befindet. Jede Audioquelle ist verantwortlich für die Verklanglichung der Emissionswerte, die, aus Perspektive des *Beobachters*, in Richtung der Audioquelle und darüber hinaus liegen. Eine Audioquelle stellt somit nicht mehr den Wert an einer Position dar, sondern eine Summe aus Werten aus einer bestimmten Richtung. Die Anzahl der Audioquellen ist variabel und kann an ein System angepasst werden. Eine hohe Anzahl an Audioquellen würde somit theoretisch eine höhere räumliche Auflösung des Klangs bewirken.

   Das Auslesen der Daten und die Zuweisung zu einer Audioquelle wird durch einen kugelförmigen *Raycaster* umgesetzt. Der Ursprung des *Raycaster* liegt in der Position des *Beobachters* und es wird ein Strahl in Richtung jeder Audioquelle geworfen. Aus allen von einem Strahl getroffenen Werten wird eine nach der Entfernung gewichtete Summe gebildet, welche als Eingangswert für die akustische Darstellung an einer Audioquelle genutzt wird.

   Durch einen praktischen Test hat sich jedoch gezeigt, dass durch eine große Anzahl an gleichzeitig spielenden Audioquelle eine genaue Verräumlichung nur schwer wahrnehmbar ist. Besonders wenn der ausgespielte Klang auf allen Audioquellen ähnlich ist, ist eine sinnvolle Verräumlichung fast unmöglich.

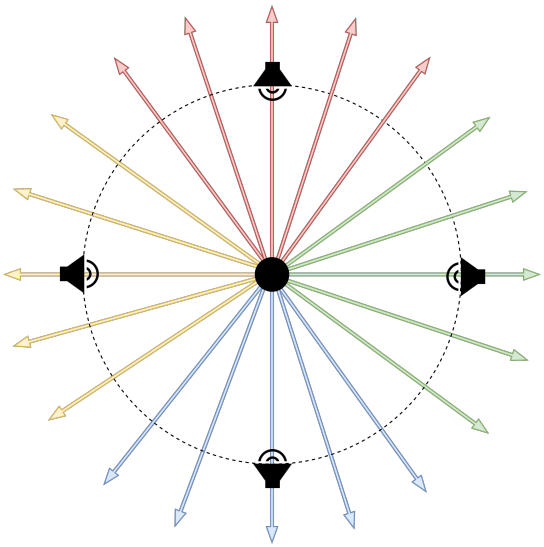

Abbildung 15: Zeidimensionale Darstellung der Audioszene und des *Raycastings* der Beobachterstationären Klangrichtungen. Der schwarze Punkt in der Mitte stellt den *Beobachter* und somit auch den Ursprung des *Raycastings* dar. Die Pfeile stellen die Strahlen dar, durch deren farbige Zuweisung die Zuordnung zu einer Audioquelle erkennbar wird.

3. **Beobachterstationäre Klangrichtungen**
   Der finale Ansatz versucht die nicht wahrnehmbare Verräumlichung der Beobachterstationären Klangkugel durch zwei Methoden zu korrigieren: Eine Reduktion der Audioquellen und eine stärkere Differenzierung des Klangs.

   Die Anzahl an Audioquellen wurde auf sechs Stück festgelegt und jede Audioquelle wird in einer der sechs Richtungen (Links, Rechts, Vorne, Hinten, Oben und Unten) zum *Beobachter* platziert. Eine Audioquelle stellt somit wieder eine Richtung dar, jedoch handelt es sich jetzt um eine Summe aus allen Werten, einer Richtung.

   Die Audioquellen befinden sich zwar immer im selben Abstand zum *Beobachter*, folgen aber nicht dessen Rotation. D. h. der *Beobachter* kann sich frei innerhalb der Audioquellen drehen, um somit dasselbe Klangbild aus unterschiedlichen Perspektiven wahrzunehmen.

   Zusätzlich wird es für jede Audioquelle ein eigenes Klangbild geben. Dies soll dabei helfen, die räumliche Wahrnehmung der Klänge zu verbessern, damit die Richtung, von der ein Klang kommt, einfa-

cher wahrzunehmen ist. Erweitert wird diese Differenzierung über eine Klangfokussierung. Das bedeutet, dass die Lautstärke einer Audioquelle abhängig ist von der Blickrichtung des *Beobachters*. Wenn dieser direkt auf eine Audioquelle schaut, sollte die maximale Lautstärke wiedergegeben werden. Wenn sich die Audioquelle hinter dem *Beobachter* befindet, wird diese mit der minimalen Lautstärke ausgespielt. Die minimale und maximale Lautstärke sind variabel und die Klangfokussierung ist an- und abschaltbar.

Das Auslesen der Daten erfolgt wieder über einen kugelförmigen *Raycaster*. Dieser besitzt die selbe Funktionsweise wie der *Raycaster* der Beobachterstationäre Klangkugel, jedoch ist die Anzahl an Strahlen in diesem Ansatz unabhängig von den Audioquellen. Es wird eine möglichst große Anzahl an Strahlen verwendet, welche einen Einfluss auf den Klang der am nächsten liegenden Audioquelle haben. Eine visuelle Darstellung dieses Prinzips ist in Abbildung 15 erkennbar.

7.2.2 Umwandeln der Daten in einen Klang

Nach den im letzten Kapitel erläuterten Funktionsweisen ist es möglich, den Datensatz durch eine Audioszene darzustellen. Damit eine Sonifikation möglich ist, müssen die Emissionwerte, die für die Audioquellen berechnet werden, in einen Klang umgesetzt werden. Wie bereits erwähnt, wird es für jede Richtung ein eigenes Klangbild geben. Damit eine Differenzierung über den Emissionsgrad erfolgen kann, muss es für den Nutzer möglich sein, den Klang zu interpretieren. In dieser Arbeit geschieht dies über eine Diskretisierung des Emissionswertes in drei Abstufungen. Es gibt für jedes Klangbild drei Ebenen, welche aufeinander aufbauen, sich jedoch in ihrer Komposition sowie Informationsgehalt verändern. Die Ebenen sind so aufgebaut, dass eine höhere Ebene immer mit den darunter liegenden Ebenen ausgespielt wird. Beispielsweise wird bei einem niedrigen Emissionswert die erste Ebene ausgespielt. Bei einem sehr hohen werden dagegen alle drei Ebenen gleichzeitig wiedergegeben.

Bei den Klängen handelt es sich um Instrumente bzw. Instrumentengruppen. Darin enthalten sind Klavier, Gitarre, Schlagwerke, Streicher, Synthesizer und Gesang.

Alle Klänge wurden von Peter Michael von der Nahmer erstellt.

7.3 Stand des Sonifikations-Plugins

Aufgrund der Zeit war es nicht möglich, das Sonifikations-Plugin im Umfang dieser Arbeit fertig zu stellen. Die Grundfunktionsweise ist jedoch erkennbar. Wenn sich ein Nutzer über die Erde bewegt, ist ein komplexeres Klangbild in Gebieten mit hohem Flugverkehr wahrnehmbar.

Allerdings fehlen noch Funktionalitäten, wie z. B. die Diskretisierung der Emissionswerte. Die Festlegung der Abstufungen benötigt eine dynamische Einstellung über die Benutzeroberfläche, da eine feste Normalisierung der Daten nicht sinnvoll ist. Die Ursache dafür liegt darin, dass es sich bei den auszuspielenden Emissonswerten immer um Summen von vielen Datenpunkten handelt, und eine Voraussage über den Bereich dieser Zahlen ist nur schwer haltbar.

Des Weiteren gibt es noch einen Fehler in der Berechnung der Strahlen des *Raycasters*. Dieser führt dazu, dass sich bei einer stationären Rotation des *Beobachters* das Klangbild ändert. Dies sollte aber nicht der Fall sein.

Außerdem fehlt eine Funktion, um die Klangfokussierung an- und abzuschalten. In der derzeitigen Version ist dieser immer angeschaltet, kann aber über die Gleichsetzung der minimalen und maximalen Lautstärke klanglich deaktiviert werden.

8 Fazit und Ausblick

Die vorliegende Arbeit hat einen umfassenden Einblick in den Entwicklungsprozess der Audio Engine für CosmoScout VR aufgezeigt. Die Auswahl einer Audiobibliothek bildet dabei das Fundament dieser Audio Engine. Des Weiteren wurden Schlüsselfunktionalitäten, wie das semimodulare Design, vorgestellt, welches die Anpassung des Audiosystems an eine Vielzahl an Anwendungsfällen ermöglicht. Zu guter Letzt gibt diese Arbeit einen Ausblick, wie das entwickelte System genutzt werden kann, um einen Sonifikation durchzuführen. Eine solche Anwendung ermöglicht es Datensätze auf eine neue Art und Weise wiederzugeben, um, besonders in komplexen und vielschichtigen Darstellungen, Information in einer weiteren Dimension wahrnehmbar zu machen.

Zusammenfassend markiert diese Arbeit und die Entwickelte Audio Engine den ersten Schritt von CosmoScout VR in Richtung der auditiven Darstellung und birgt somit einen potenziellen Aufbruch, um Informationen in unserem Sonnensystem anders und neu wahrzunehmen. Auch wenn noch weiterführende Arbeiten erforderlich sind, um eine ebenso umfassende Klangwiedergabe zu erreichen wie die visuelle Darstellung in CosmoScout VR, bietet das System dennoch vielversprechende neue Elemente.

9 Glossar

Begriff	Definition
Beobachter	Als Beobachter wird der Nutzer bzw. die vom Nutzer gesteuerte Kamera in CosmoScout VR bezeichnet.
Buffer	Für diese Arbeit wird als Buffer dasselbe Konzept verstanden wie von OpenAL. Ein Buffer ist ein Speicher, welcher Daten (i. d. R. Audiodaten) kapselt. Diese Daten können von Audioquellen verwendet werden, um Klänge auszuspielen [11, S. 45].
CAVE	Als CAVE wird eine Darstellung der virtuellen Realität bezeichnet, welche dadurch charakterisiert wird, dass ein ganzer Raum als Projektionsfläche des Bildes dient. Dies umfasst i. d. R. die Wände, kann aber auch die Decke und den Boden miteinbeziehen. Die Bilder werden mit Hilfe von Projektoten dargestellt. [6].
Debug und Release Modus	Diese Modi stellen Konfigurationen im Kompilierungsprozess dar. Der Debug-Modus enthält keine Optimierung und enthält Informationen, die das Lösen von Fehlern im Programm erleichtern. Der Release enthält dagegen keine Information, ist dafür aber optimiert [13].
Frame	Als Frame wird ein einzelner Aktualisierungszyklus von CosmosScout VR verstanden. Das behinhaltet alle Berechnung der Logik und der Darstellung. Falls kein Leistungsdefizit vorliegt, arbeitet Cosmoscout VR mit 60 Aktualisierungen, also Frames, pro Sekunde. I. d. R. wird eine hohe Frame Rate angezielt, da dies zu einer flüssigeren und damit angenehmeren Erfahrung führt.

HRTF	Die head-related transfer function ist ein Effekt, der als frequenzabhängige Amplitude und Zeitverzögerungsunterschiede betrachtet werden kann, der hauptsächlich durch die Form der Ohrmuscheln entsteht [3, S. 41]. Im Kontext dieser Arbeit wird HRTF allgemein als Synonym dafür verwendet, dieses Phänomen durch Software zu simulieren.
Head-Mounted-Display	Head-Mounted-Displays sind Bildschirme, die am Kopf eines Nutzers befestigt sind. [23]. Im Kontext dieser Arbeit wird immer davon ausgegangen, dass ein Head-Mounted-Display in der Lage ist, die Bewegung des Trägers zu erkennen, wodurch die Übertragung in die Bewegung in CosmoScout VR möglich ist.
JSON-Format	Das JSON-Format ist ein textbasiertes Datenformat, das in der Regel zum Austausch von Informationen definiert wird [12].
OpenGL	OpenGL ist eine Softwareschnittstelle zur Grafikeinheit eines Systems, welches es einem Entwickler erlaubt, geometrische Objekte im Raum zu definieren und diese darzustellen [22, S. 1].
Occlusion	Als Occlusion wird der Effekt bezeichnet, wenn Schallwellen durch ein Objekt, wie z. B. eine Wand, hindurchgehen, was eine Dämpfung des Klangs hervorbringt [18, S. 13].
synchron/asynchron	Ein Prozess wird als synchron verstanden, wenn dieser auf der selben Prozessorteinheit arbeitet wie der zu vergleichende Prozess. Zwei Prozesse werden asynchron verstanden, wenn diese auf unterschiedlichen Prozessorteinheiten arbeiten und somit keine gleichmäßige Abarbeitung von Instruktionen ohne zusätzliche Hilfe gewährleistet ist. Der Vorteil einer asynchronen Arbeitsweise ist, dass die Prozesse zeitgleich laufen können und nicht nacheinander erfolgen müssen.
Streaming	In dieser Arbeit wird Streaming als kontinuirlicher Fluss an (Audio-) Daten verstanden, welche nacheinander verarbeitet werden. Im Gegensatz zu non-streaming (Audio-) Daten, die in ihrem Ganzen auf einmal verarbeitet werden.

Tabelle 14: Glossar Übersicht

Abbildungsverzeichnis

Tabellenverzeichnis

Quellen

[1] Adalin B.V. *AeonWave Dokumentation*. Letzter Zugriff: 22.01.2023. o.J. URL: https://adalin.com/documentation.html.

[2] Adalin B.V., Hrsg. *AeonWave-HD 64 Technical Specifications*. o.J. URL: https://adalin.com/downloads/TechSpecs.pdf.

[3] Durand R. Begault. *3-D Sound for Virtual Reality and Multimedia*. 2000. URL: https://human-factors.arc.nasa.gov/publications/Begault_2000_3d_Sound_Multimedia.pdf.

[4] Alexander Bock, Emil Axelsson, Jonathas Costa, Gene Payne, Micah Acinapura, Vivian Trakinski, Carter Emmart, Cláudio Silva, Charles Hansen und Anders Ynnerman. "OpenSpace: A System for Astrographics". In: *IEEE transactions on visualization and computer graphics* PP (Aug. 2019). DOI: 10.1109/TVCG.2019.2934259.

[5] Omar Cornut. *Dear ImGui*. Hrsg. von GitHub. Letzter Zugriff: 22.01.2023. URL: https://github.com/ocornut/imgui.

[6] Carolina Cruz-Neira, Daniel J. Sandin, Thomas A. DeFanti, Robert V. Kenyon und John C. Hart. "The CAVE: audio visual experience automatic virtual environment". In: *Commun. ACM* 35.6 (Juni 1992), S. 64–72. ISSN: 0001-0782. DOI: 10.1145/129888.129892. URL: https://doi.org/10.1145/129888.129892.

[7] Elias Elmquist, Malin Ejdbo, Alexander Bock und Niklas Rönnberg. "OpenSpace Sonification: Complementing Visualization of the Solar System with Sound". In: *Proceedings of the 26th International Conference on Auditory Display (ICAD 2021) :* The International Community for Auditory Display, 2021, S. 135–142. ISBN: 9780967090474. DOI: 10.21785/icad2021.018.

[8] *FMOD Engine User Manual*. Letzter Zugriff: 22.01.2023. 2023. URL: https://www.fmod.com/docs/2.02/api/core-guide.html.

[9] Nikolaus Gebhardt. *irrKlang Dokumentation*. Letzter Zugriff: 22.01.2023. 2018. URL: https://www.ambiera.com/irrklang/docu/index.html.

[10] T. Hermann. *Sonification – A Definition*. Letzter Zugriff: 28.01.2024. 2011. URL: http://sonification.de/son/definition.

[11] Garin Hiebert. "OpenAL 1.1 Specification and Reference". In: (2005). Letzter Zugriff: 22.01.2023. URL: https://www.openal.org/documentation/openal-1.1-specification.pdf.

[12] ECMA International. *The JSON Data Interchange Syntax*. Letzter Zugriff: 29.01.2023. 2017. URL: https://ecma-international.org/wp-content/uploads/ECMA-404_2nd_edition_december_2017.pdf.

[13] Mike Jones, Gordon Hogenson, Mohit Patel, Saisang Cai, Dennis Lee, David Coulter, Genevieve Warren und Caro Caserio. *Set debug and release configurations in Visual Studio*. Letzter Zugriff: 29.01.2023. 2024. URL: https://learn.microsoft.com/en-us/visualstudio/debugger/how-to-set-debug-and-release-configurations?view=vs-2022.

[14] kcat. *OpenAL-Soft*. Hrsg. von GitHub. Letzter Zugriff: 22.01.2023. URL: https://github.com/kcat/openal-soft.

[15] Jari Komppa. *soLoud Dokumentation*. Letzter Zugriff: 22.01.2023. 2020. URL: https://solhsa.com/soloud/index.html.

[16] Jari Komppa. *soLoud Github Repository*. Letzter Zugriff: 24.01.2023. 2020. URL: https://github.com/jarikomppa/soloud.

[17] Microsoft, Hrsg. *Performance Counters*. Letzter Zugriff: 22.01.2023. 2023. URL: https://learn.microsoft.com/en-us/windows/win32/api/_perf/.

[18] Daniel Peacock, Peter Harrison, Andrea D'Orta, Valery Carpentier und Edward Cooper. "OpenAL Effects Extension Guide". In: (2006). Letzter Zugriff: 24.01.2023. URL: https://openal-soft.org/misc-downloads/Effects%20Extension%20Guide.pdf.

[19] *Resonance Audio Dokumentation*. Letzter Zugriff: 22.01.2023. o.J. URL: https://resonance-audio.github.io/resonance-audio/develop/overview.html.

[20] Simon Schneegans, Moritz Zeumer, Jonas Gilg und Andreas Gerndt. "CosmoScout VR: A Modular 3D Solar System Based on SPICE". In: *2022 IEEE Aerospace Conference (AERO)*. IEEE, 2022. ISBN: 978-1-6654-3760-8. DOI: 10.1109/AERO53065.2022.9843488.

[21] *SDL2 Documentation*. Letzter Zugriff: 22.01.2023. 2023. URL: https://wiki.libsdl.org/SDL2/FrontPage.

[22] Mark Segal und Kurt Akeley. "The OpenGL Graphics System: A Specification (Version 4.0(Core Profile) - March 11, 2010)". In: (20010). Letzter Zugriff: 29.01.2023. URL: https://registry.khronos.org/OpenGL/specs/gl/glspec40.core.pdf.

[23] Takashi Shibata. "Head mounted display". In: *Displays* 23.1 (2002), S. 57–64. ISSN: 0141-9382. DOI: https://doi.org/10.1016/S0141-9382(02)00010-0. URL: https://www.sciencedirect.com/science/article/pii/S0141938202000100.

[24] *SLD2 Mixer Mix_SetPosition*. Letzter Zugriff: 22.01.2023. o.J. URL: https://wiki.libsdl.org/SDL2_mixer/Mix_SetPosition.

[25] Arthur Taylor, Erik de Castro Lopo, evpobr und David Seifert. *LibSndFile*. Letzter Zugriff: 22.01.2023. URL: https://github.com/libsndfile/libsndfile.

[26] Bruce N. Walker und Michael A. Nees. *Theory of Sonification*. Logos Verlag Berlin GmbH, S. 29. ISBN: 978-3-8325-2819-5.

A Anhang

A.1 Testergebnisse für die Wahl der Audiobibliothek

In vielen Tests lässt sich das Resultat „Ergebnis wie erwartet“ erkennen. Dies bedeutet, dass es keine Komplikationen in der Umsetzung des Tests gab und dass das resultierende Klangergebnis einen realistischen Eindruck macht.

A.1.1 OpenAL

1. **Verräumlichung**
   Ergebnis wie erwartet.

2. **Verräumlichung und Mixing**
   Ergebnis wie erwartet.

3. **Stresstess**

Konfiguration	CPU Mittelwert	CPU Max	Speicher Min[a][b]	Speicher Max[b]	Maximale Anzahl[c]
1 Buffer	1,1%	13%	5 MB (74)	16 MB (85)	ca. 2880
1 Buffer pro Quelle	1,5%	13,1%	4 MB (73)	657 MB (726)	ca. 2880

[a] Geringster Speicherbedarf, sobald mindestens 1 Quelle spielt
[b] Bereinigter Speicherbedarf, in den Klammern der unbereinigter Bedarf
[c] Maximale Anzahl an spielenden Quellen bevor die Audio fehlerhaft wird (separater Test von Performancemessung)

   Für die Buffer Daten wurde eine Datei mit der Größe 626 KB verwendet.

4. **Dopplereffekt**
   Ergebnis wie erwartet.

5. Distanzdämpfung
   Ergebnis meist wie erwartet.
   Es gibt 3 verschiedene Distanz Modelle in jeweils 2 Version:

   - Inverse Distance (Clamped)
   - Linear Distance (Clamped)
   - Exponent Distance (Clamped)

   Das lineare Distanzmodell hat aber aus bisher unerkenntlichen Gründen keine Dämpfung.

6. **Distanzverzögerung und Schallgeschwindigkeit**
   Es gibt keine Distanzverzögerung in OpenAL. Die Schallgeschwindigkeit lässt sich nur für die Berechnung des Dopplereffekts anpassen.

7. **Occlusion**
   Es gibt keine fertigen Werkzeuge für Occlusion/Exclusion/Obstruction in OpenAL. Die Dokumentation für die Effects Extension enthält aber eine relativ ausgiebige Erklärung, wie man diese mit den enthaltenen Effekten und Filtern selbst umsetzen kann. Im Testprogramm wird dies mittels eines Tiefpassfilters ermöglicht, welcher angeschaltet wird, sobald sich der Hörer hinter der Wand befindet.

8. **Surround-Sound**
   Ergebnis wie erwartet.

9. **Atmosphärische Eigenschaften**
   OpenAL bietet durch den Air Absorption Faktor (enthalten in der Effetcs Extension) eine Möglichkeit, um andere atmosphärische Konditionen zu modellieren. Dieser Wert bietet aber keine direkte Beschreibung der Atmosphäre, sondern ist nur eine distanzabhängige Dämpfung von hohen Frequenzen, welcher durch einen einzelnen Wert von 0 bis 10 ausgedrückt wird. Dies kann nützlich sein, um eine Audioquelle in einer anderen Umgebung (z. B. in einer Wolke) darzustellen. Aber eine physikalisch realistische Modellierung ist dadurch nur wenig möglich.

10. **Streaming langer Inhalte**
    Streaming ist möglich, muss aber selbst implementiert werden. Dies kann man mit der Buffer-Queue-Funktion realisieren. Alternativ kann man zur Hilfe auch die Extension SOFT_callback_buffer nutzen, mit welcher man eine Rückruf-Funktion definieren kann, welche aufgerufen wird, sobald einer Quelle die Daten zum Wiedergeben ausgehen.

11. **HRTF Unterstützung**
    Ergebnis wie erwartet.

12. **Stresstest mit HRTF**

Konfiguration	CPU Mittelwert	CPU Max	Speicher Min[a,b]	Speicher Max[b]	Maximale Anzahl[c]
1 Buffer	4,2%	13,3%	5 MB (74)	17 MB (86)	ca. 510
1 Buffer pro Quelle	4,6%	13,9%	5 MB (74)	658 MB (727)	ca. 510

[a] Geringster Speicherbedarf, sobald mindestens 1 Quelle spielt
[b] Bereinigter Speicherbedarf, in den Klammern der unbereinigter Bedarf
[c] Maximale Anzahl an spielenden Quellen bevor die Audio stottert (separater Test von Performance Messung)

Für die Buffer Daten wurde eine Datei mit der Größe 626 KB verwendet.

13. **Effekte und Filter**
    Durch die Effects Extension bietet OpenAL 12 Effekte und 3 Filter. Einige Effekte lassen sich jedoch nicht verräumlichen, da diese aus einem Mono ein Stereo-Signal erzeugen. Dies ist bei allen Effekten im Testprogramm (Reverb, Echo, Flanger) der Fall. Außerdem lassen sich Effekte und Filter nicht global setzen und man muss jede Quelle mit den Effekten und Filtern einzeln zum Mixer routen.

14. **Stresstest mit Effekten und Filter**

Konfiguration	CPU Mittelwert	CPU Max	Speicher Min[a,b]	Speicher Max[b]	Maximale Anzahl[c]
ohne HRTF					
1 Buffer	1,1%	13,3%	6 MB (75)	17 MB (86)	ca. 2880
1 Buffer pro Quelle	1,6%	13%	6 MB (75)	658 MB (727)	ca. 2880
mit HRTF					
1 Buffer	4,2%	13,1%	8 MB (77)	19 MB (88)	ca. 510
1 Buffer pro Quelle	4,5%	13,1%	6 MB (75)	659 MB (728)	ca. 510

[a] Geringster Speicherbedarf, sobald mindestens 1 Quelle spielt
[b] Bereinigter Speicherbedarf, in den Klammern der unbereinigter Bedarf
[c] Maximale Anzahl an spielenden Quellen bevor die Audio stottert (separater Test von Performance Messung)

Für die Buffer Daten wurde eine Datei mit der Größe 626 KB verwendet.
Benutzte Effekte und Filter: Flanger, Echo, Reverb und Tiefpass.

15. **Tonerzeugung**
    Es gibt keine Möglichkeit, direkt mit OpenAL einen Ton zu erzeugen. Da man die Buffer aber mit beliebigen Daten füllen kann, kann man sich jedoch ein eigenes Signal erzeugen und dieses in den Buffer laden.

16. **Lautsprecherpositionen konfigurieren**
    Eine Live-Konfiguration von Lautsprecherpositionen ist nicht möglich. Man kann aber Presets für Lautsprecherpositionen einrichten, welche jedoch nicht während der Laufzeit gewechselt werden können. Die Presets sind AmbDec-Configuration-Dateien.

17. Anmerkungen

    - Es gibt keine eingebaute Funktion, um Dateien auszulesen. Dies muss man entweder selbst implementieren oder eine Hilfslibrary nutzen.
    - Die Effects Extension enthält eine ausgiebige Erklärung und Tipps, wie man ein interaktives Soundsystem aufbaut. Diese lassen sich auch auf andere Libraries anwenden.
    - Es gibt eine Reihe an Extensions für OpenAL, die die Möglichkeiten noch erweitern. Eine Liste für die OpenAL-Soft-Implementierung findet man unter: https://openal-soft.org/openal-extensions/
    - Die maximale Anzahl an gleichzeitig spielenden Quellen ist sehr wahrscheinlich durch das Limit von einem Thread, auf dem OpenAL läuft, limitiert. Durch die EXT_thread_local_context Extension kann man jedoch relativ einfach mehrere Contexte auf mehreren Prozessoreinheiten zum Laufen bringen und somit dieses Limit umgehen. Dies benötigt aber eine Art von Context-Management-System, da eine Audioquelle nur mit dem richtigen Kontext angesteuert werden kann. Durch ein solches System könnte man auch steuern, wie viele Prozessoreinheiten man dem Audiosystem zur Verfügung stellen will.

A.1.2 soLoud

1. **Verräumlichung**
   Ergebnis wie erwartet.

2. **Verräumlichung und Mixing**
   Ergebnis wie erwartet.

3. **Stresstess**

Konfiguration	CPU Mittelwert	CPU Max	Speicher Min[a,b]	Speicher Max[b]	Maximale Anzahl[c]
1 Buffer	0,54%	9,86%	11 MB (80)	13 MB (82)	4095[d]
1 Buffer pro Quelle	0,79%	9,85%	11 MB (80)	1291 MB (1360)	4095[d]

[a] Geringster Speicherbedarf, sobald mindestens 1 Quelle spielt
[b] Bereinigter Speicherbedarf, in den Klammern der unbereinigter Bedarf
[c] Maximale Anzahl an spielenden Quellen bevor die Audio stottert (separater Test von Performance Messung)
[d] soLoud unterstützt nur maximal 255 gleichzeitig spielende Quellen. Die Restlichen existieren zwar sind aber 'virtuelle' Quellen und werden nicht abgespielt

Für die Buffer Daten wurde eine Datei mit der Größe 626 KB verwendet.

Bei dem Test zur Maximalen Anzahl an gleichzeitig spielenden Quellen kam es während der Schrittweisen Erhöhung immer wieder zu kleineren Fehlern in der Audioausgabe. Diese haben sich aber von selbst wieder behoben, wenn man die Anzahl weiter erhöht.

4. **Dopplereffekt**
   Ergebnis wie erwartet.

5. Distanzdämpfung Ergebnis meist nicht wie erwartet.
   Es gibt 3 verschiedene Distanz Modelle:

   - Inverse Distance Clamped
   - Linear Distance Clamped
   - Exponent Distance Clamped

   Die Modelle nutzen die selben Formeln wie OpenAL. Das Linearedistanz- und Exponentdistanzmodell haben aber aus bisher unerkenntlichen Gründen keine Dämpfung.

6. **Distanzverzögerung und Schallgeschwindigkeit**
   Ergebnis wie erwartet.

7. **Occlusion**
   Es gibt keine fertigen Tools für Occlusion/Exclusion/Obstruction in soLoud. Die Dokumentation enthält auch keine Information. Der Test im Testprogramm wurde nach dem selben Prinzip wie in OpenAL mittels eines Tiefpassfilters erstellt.

8. **Surround-Sound**
   Ergebnis wie erwartet, aber es ist nötig, die Anzahl an Lautsprechern manuell anzugeben. Des Weiteren ist eine Lautsprecheranzahl von 3, 5 und 7 nicht möglich. Die Anpassung, wie man die Lautsprecheranzahl ändert, ist nicht in der Dokumentation enthalten.

9. **Atmosphärische Eigenschaften**
   Es gibt keine Möglichkeit, um atmosphärische Eigenschaften zu modellieren.

10. **Streaming langer Inhalte**
    Streaming ist sehr einfach umzusetzen, mittels einer fertigen soLoud-Klasse.

11. **HRTF Unterstützung**
    HRTF wird nicht unterstützt.

12. **Stresstest mit HRTF**
    Nicht möglich.

13. **Effekte und Filter**
    SoLoud bietet 10 Effekte/Filter. Wie bei OpenAL lassen sich einige Effekte nicht verräumlichen, da diese aus einem Mono ein Stereo-Signal erzeugen. Dies sind aber nicht die gleichen Effekte wie in OpenAL. Im Testprogramm ist dies nur für den Reverb der Fall. Der Echo- und Flangereffekt werden verräumlicht. Des Weiteren lassen sich Effekte und Filter global setzen, und ein manuelles Routing zum Mixer ist möglich, aber nicht zwingend notwendig.

14. **Stresstest mit Effekten und Filter**

Konfiguration	CPU Mittelwert	CPU Max	Speicher Min[a,b]	Speicher Max[b]	Maximale Anzahl[c]
1 Buffer	0,57%	10,2%	10 MB (79)	13 MB (82)	4095[d]
1 Buffer pro Quelle	0,8%	9,9%	11 MB (80)	1294 MB (1363)	4095[d]

[a] Geringster Speicherbedarf, sobald mindestens 1 Quelle spielt
[b] Bereinigter Speicherbedarf, in den Klammern der unbereinigter Bedarf
[c] Maximale Anzahl an spielenden Quellen bevor die Audio stottert (separater Test von Performance Messung)
[d] soLoud unterstützt nur maximal 255 gleichzeitig spielende Quellen. Die Restlichen existieren zwar sind aber 'virtuelle' Quellen und werden nicht abgespielt

    Für die Buffer Daten wurde eine Datei mit der Größe 626 KB verwendet.
    Benutzte Effekte und Filter: Flanger, Echo, Reverb und Tiefpass.
    Bei dem Test zur Maximalen Anzahl an gleichzeitig spielenden Quellen kam es während der Schrittweisen erhöhung immer wieder zu kleineren fehlern in der Audio Ausgabe. Diese haben sich aber von selbst wieder behoben, wenn man die Anzahl weiter erhöht.

15. **Tonerzeugung**
    Es gibt verschiedene Arten um in soLoud Töne zu erzeugen:

    - Noise: Man kann verschiedene Arten von Noise erzeugen: White, Pink, Brownish, Blueish
    - SFXR: ein Retro-Sound-Effekt-Synthesizer, welcher eher für Retro Video Spiele geeignet ist
    - Sprache: es gibt 2 Sprach-Synthesizer, beide sind alt und klingen auch so

16. **Lautsprecherpositionen konfigurieren**
    Ergebnis wie erwartet. Man kann die Lautsprecherpositionen live anpassen.

17. Anmerkungen

    - soLoud besitzt ein Virtual-Voices-System. Das bedeutet, es gibt ein Maximum an aktiv spielenden Quellen. Wenn diese überschritten wird, werden die restlichen Quellen „virtuell“ und nur noch die lautesten Quellen werden abgespielt. Dieses maximale Limit an aktiven Quellen kann man selbst festlegen, solange es kleiner ist als das Limit der virtuellen Quellen von 4095. Im Testprogramm kann man den Wert der aktiven Quellen aber nur maximal auf 255 setzen, da es darüber hinaus zu einem Programmabsturz kommt, weil eine Quelldatei nicht gefunden wird. Bisher ist nicht ersichtlich, warum diese Quelldatei nicht gefunden wird und warum dieser Fehler nur auftritt, wenn der Wert 255 überschreitet.
    - 3D-Audio-Funktionsaufrufe sind nicht threadsicher, d. h. diese sollten nicht von mehreren Threads gleichzeitig aufgerufen werden.
    - Es ist nicht möglich, Informationen von 3D-Source-Instanzen zu erhalten, wie z. B. Position, Geschwindigkeit, Orientierung, ...
    - Manuelles Routing ist möglich, dadurch kann man Quellen bei Bedarf auch gruppieren.

A.1.3 aeon wave

1. **Verräumlichung**
   Ergebnis wie erwartet.

2. **Verräumlichung und Mixing**
   Ergebnis wie erwartet.

3. **Stresstess**

Konfiguration	CPU Mittelwert	CPU Max	Speicher Min[a,b]	Speicher Max[b]	Maximale Anzahl[c]
1 Buffer	0,59%	55,5%	424 MB (493)	432 MB (501)	>5000
1 Buffer pro Quelle	1,48%	59,4%	ToDo	1714 MB (1783)	>5000

[a] Geringster Speicherbedarf, sobald mindestens 1 Quelle spielt
[b] Bereinigter Speicherbedarf, in den Klammern der unbereinigter Bedarf
[c] Maximale Anzahl an spielenden Quellen bevor die Audio stottert (separater Test von Performance Messung)

Für die Buffer Daten wurde eine Datei mit der Größe 626 KB verwendet.

4. **Dopplereffekt**
   Ergebnis wie erwartet.

5. Distanzdämpfung
   Ergebnis meist wie erwartet.
   Es gibt 5 verschiedene Distanz Modelle in jeweils 2 Version:

   - Exponential Distance (Delay)
   - ISO 9613 Distance (Delay)
   - Inverse Distance (Clamped) - nach OpenAL
   - Linear Distance (Clamped) - nach OpenAL
   - Exponent Distance (Clamped) - nach OpenAL

   Modelle mit einer ‚Delay'-Version bieten eine Verzögerung über die Distanz. Das ISO 9613 Distanzmodell bietet die Möglichkeit, die Dämpfung nach der Temperatur, atmosphärischem Druck und relativer Luftfeuchtigkeit zu berechnen. Das Exponential Distance Modell nach OpenAL ist aus einem bisher unerkenntlichen Grund stumm und erzeugt keinen Ton.

6. **Distanzverzögerung und Schallgeschwindigkeit**
   Die Distanzverzögerung funktioniert, ist aber an die Distanzmodelle 'Exponential Distance Delay' und 'ISO 9613 Distance Delay' gebunden. Die Schallgeschwindigkeit ist nach meinen Tests nur für den Dopplereffekt änderbar.

7. **Occlusion**
   In aeon wave kann man auf zwei Arten Occlusionseffekte nutzen. Entweder nach Beschreibung der OpenAL Effects Extension Dokumentation mittels Effekten und Filtern, wobei man die meisten Occlusion-Berechnungen selbst implementieren muss.Oder über die eingebauten Occlusion Tools mittels der Audio Frames. Diese Methode ermöglicht es aber nur, mit Objekten in Quaderform zu arbeiten.

8. **Surround-Sound**
   Surround-Sound soll dem Render-Modus „surround" möglich sein,

allerdings funktioniert dieser in den Tests nicht. Laut Dokumentation gibt es abseits vom Render-Modus keine andere Einstellungsmöglichkeit.

9. **Atmosphärische Eigenschaften**
   Aeon wave bietet durch das Distanzmodell ISO 9613 eine Möglichkeit, um andere atmosphärische Konditionen zu modellieren. Dieses Modell bietet die Möglichkeit, die Atmosphäre in Temperatur, atmosphärischem Druck und relativer Luftfeuchtigkeit zu beschreiben. Diese Werte haben aber lediglich einen Einfluss auf die Dämpfung über die Distanz eines Klangs.

10. **Streaming langer Inhalte**
    Streaming ist nicht für eine normale Audioquelle im Raum möglich. Es funktioniert aber für ein Playback, eine Möglichkeit, um Hintergrundklänge (Musik) nicht verräumlicht abzuspielen.

11. **HRTF Unterstützung**
    Ergebnis wie erwartet, allerdings kommt es (manchmal) zu einem leichtem Rauschen im Hintergrund.

12. **Stresstest mit HRTF**

Konfiguration	CPU Mittelwert	CPU Max	Speicher Min[a,b]	Speicher Max[b]	Maximale Anzahl[c]
1 Buffer	1,2%	55,8%	423 MB (492)	430 MB (499)	>5000
1 Buffer pro Quelle	2,9%	61,2%	TODO	1712 MB (1781)	>5000

[a] Geringster Speicherbedarf, sobald mindestens 1 Quelle spielt
[b] Bereinigter Speicherbedarf, in den Klammern der unbereinigter Bedarf
[c] Maximale Anzahl an spielenden Quellen bevor die Audio stottert (separater Test von Performance Messung)

    Für die Buffer Daten wurde eine Datei mit der Größe 626 KB verwendet.

13. **Effekte und Filter**
    Aeon wave bietet 12 Effekte und 11 Filter. Davon sind alle verräumlicht, anders als bei soLoud und OpenAL. Aus einem bisher unerkenntlichen Grund kommt es aber zu einer Exception, wenn man einen Reverb-Effekt im Testprogramm deaktiviert. Bei anderen Effekten ist dies nicht der Fall, obwohl das (de-) aktivieren von Effekten bei allen die selbe Vorgehensweise ist.

14. **Stresstest mit Effekten und Filter**

Konfiguration	CPU Mittelwert	CPU Max	Speicher Min[a,b]	Speicher Max[b]	Maximale Anzahl[c]
ohne HRTF					
1 Buffer	0,57%	52,8%	424 MB (493)	431 MB (500)	>5000
1 Buffer pro Quelle	1,2%	55,4%	TODO	1714 MB (1783)	>5000
mit HRTF					
1 Buffer	1,4%	55,8%	424 MB (493)	431 MB (500)	>5000
1 Buffer pro Quelle	2,3%	57%	TODO	1714 MB (1783)	>5000

[a] Geringster Speicherbedarf, sobald mindestens 1 Quelle spielt
[b] Bereinigter Speicherbedarf, in den Klammern der unbereinigter Bedarf
[c] Maximale Anzahl an spielenden Quellen bevor die Audio stottert (separater Test von Performance Messung)

Für die Buffer Daten wurde eine Datei mit der Größe 626 KB verwendet.
Benutzte Effekte und Filter: Flanger, Reverb (Echo), Reverb und Tiefpass.

15. **Tonerzeugung**
    Es gibt 2 Arten, um Töne zu erzeugen. Presets: Es gibt eine Reihe an fertigen synthetischen Sounds, von Natur und Sci-Fi Geräuschen bis hin zu Instrumenten. Davon klingen aber nicht alle authentisch. Und Waveform: Man kann Töne, wie z.B. Sinus, Sägezahnschwingung oder Rauschen in einer bestimmten Frequenz erzeugen.

16. **Lautsprecherpositionen konfigurieren**
    In der Dokumentation ist keine Einstellungsmöglichkeit dafür zu finden.

17. Anmerkungen

    - Looping ist nicht nahtlos.
    - HRTF hat leichte Rausch/Übersteuerungsklänge
    - Aeon wave nimmt den kompletten Audio-Treiber ein, d.h. solange aeon wave läuft, können andere Programme keine Audio ausgeben. Zwar gibt es eine Getter-Funktion, um zu überprüfen, ob der Treiber es unterstützt, Audio von mehreren Programmen anzunehmen, was auch bestätigt wird, allerdings wird dies aus bisher unerkenntlichen Gründen nicht getan. Weitere Einstellungsmöglichkeiten konnten dazu auch nicht gefunden werden.
    - Einige Informationen auf der Webseite sind falsch. Z. B. kann der Mixer laut Webseite nur 256 Quellen entgegennehmen. Dies steht aber im Kontrast zu den Tests, und auch der Entwickler sagte auf Anfrage, dass es kein Hard-Limit gibt für

Anzahl. In Zukunft wollen die Entwickler selbst testen, was das tatsächliche Limit ist.

- Man kann Quellen als Audioframes gruppieren und sehr einfach als Gruppe bearbeiten.
- Man kann Konfigurations-Dateien erstellen, um synthetische Wellenformen zu definieren und um Effekte und Filter an Mixer/Quellen/Audioframes zuzuweisen.
- MIDI Support.
- Die C++-Version verfügt über ein automatisches Buffermanagement, welches erkennt, wenn man die selbe Datei ein weiteres Mal in einen Buffer laden möchte und automatisch den bereits existierenden Buffer im Hintergrund dafür nutzt.

A.2 Erweitereter Anhang

In diesem Abschnitt sind alle angehängten Dateien aufgeführt.

- Testsoftware
  Das Verzeichnis mit dem Namen „Testsoftware“ enthält den gesamten Quellcode der Testsoftware.
- Performancemessungen der Audio Bibliotheken
  Die Exceldatei „Performancemessungen_Audio_Bibliotheken“ enthält die Daten (und eine visuelle Aufbereitung) der Performancemessungen der einzelnen Audiobibliotheken.
- UML Diagramm der Audio Engine
  Die SVG Datei „UML_Audio_Engine“ stellt das gesamt UML-Diagramm der Audio Engine dar.
- Programmcode der Audio Engine
  In dem Verzeichnis „audio_engine“ ist der Quellcode für die Audio Engine zu finden. Um diese auszuführen ist jedoch die gesamte CosmoScout VR Software notwendig. In der ReadMe Datei ist beschrieben, wie man die Audio Engine ausführen kann.
- csp-audio-test
  Das Verzeichnis „csp-audio-test“ enthält den Quellcode des Test-Plugins für die Performancemessung der Audio Engine. Um dieses Plugin auszuführung, muss das Verzeichnis in den Plugin Ordner von CosmoScout VR kopiert werden. Zusätzlich, muss die in der

ReadMe.md aufgezeigt Konfiguration in die Konfiguationsdatei für CosmoScout VR kopiert werden.

- csp-sonification
  Das Verzeichnis „csp-sonification“ enthält den Quellcode des Sonfikations-Plugins. Um dieses Plugin auszuführung, muss das Verzeichnis in den Plugin Ordner von CosmoScout VR kopiert werden. Zusätzlich, muss die in der ReadMe.md aufgezeigt Konfiguration in die Konfiguationsdatei für CosmoScout VR kopiert werden.
- Datensatz Sonifikation
  Die JSON-Datei „air_traffic_emissions_global_volume.json“ enthält den Datensatz, um die Sonifikation zu testen.