HAW HAMBURG

BACHELORARBEIT

Realisierung eines Audio-Plugins zur Simulation von Raumimpulsantworten

vorgelegt am 21.06.2024

Niklas Grotzeck

Erstprüferin: Prof. Dr. Eva Wilk
Zweitprüfer: Prof. Dr. Jan Mietzner

HOCHSCHULE FÜR ANGEWANDTE WISSENSCHAFTEN HAMBURG
Department Medientechnik
Finkenau 35
22081 Hamburg

Zusammenfassung

In dieser Arbeit wird ein Audio-Plugin zur Simulation von Raumimpulsantworten entwickelt. Nutzer:innen können Parameter wie Raumgröße, Schallquellen- und Empfängerposition sowie Wandmaterialien anpassen. Die Arbeit konzentriert sich auf die Spiegelschallquellen-Methode für quaderförmige Räume, implementiert in Python und C++ mittels des JUCE-Frameworks. Ein effizienter Algorithmus wird verwendet, der redundante Berechnungen minimiert und das Phänomen der „Sweeping-Echoes“ reduziert. Das Plugin bietet eine benutzerfreundliche Oberfläche und ist plattformübergreifend einsetzbar. Die Verifizierung erfolgt in mehreren Schritten: Zunächst werden die Berechnungszeiten der Code-Implementationen analysiert, um die erzeugte Latenz und den Einfluss der verschiedenen Parameter zu bestimmen. Anschließend werden die Ergebnisse mit dem Akustik-Simulationsprogramm EASE verglichen. Dabei werden Reflexionen und Nachhallzeit untersucht. Ein Hörvergleich dient abschließend dazu, die akustische Qualität der verschiedenen Simulationen zu bewerten.

Abstract

In this work, an audio plugin for simulating room impulse responses is developed. Users can adjust parameters such as room size, source and receiver positions, and wall materials. The work focuses on the image source method for rectangular rooms, implemented in Python and C++ using the JUCE framework. An efficient algorithm is used to minimize redundant calculations and reduce the phenomenon of ”sweeping echoes.” The plugin offers a user-friendly interface and is cross-platform compatible. Verification is conducted in several steps: First, the computation times of the code implementations are analyzed to determine the latency produced and the influence of various parameters. Then, the results are compared with the acoustic simulation program EASE, examining reflections and reverberation time. Finally, a listening test is conducted to assess the acoustic quality of the different simulations.

Inhaltsverzeichnis

Abbildungsverzeichnis

Tabellenverzeichnis

1 Einleitung

Die Raumakustiksimulation ist ein Werkzeug, das darauf abzielt, die Klangumgebung eines Raumes virtuell nachzubilden. Ihr Ziel ist es, die akustischen Eigenschaften eines bestimmten Ortes zu erfassen und sie in einer digitalen Umgebung zu reproduzieren. Diese Simulationstechnologie ist wertvoll für Musiker:innen und Tontechniker:innnen, die ein genaues Verständnis davon benötigen, wie ihre Musik oder ihre Produktion in verschiedenen akustischen Umgebungen klingen wird. So hilft es zum Beispiel, dabei die akustischen Eigenschaften eines Raumes beurteilen zu können, bevor dieser gebaut wird.

Die Entwicklung von Audio-Plugins für Raumakustiksimulationen bietet eine äußerst sinnvolle Lösung für diese Anforderungen. Ein:Eine Musiker:in, der:die beispielsweise ein Stück komponiert hat, möchte oft wissen, wie es in einem bestimmten Veranstaltungsort klingen wird, bevor er:sie es tatsächlich dort aufführt oder aufnimmt. Durch die Integration von Raumakustiksimulationen in ein Audio-Plugin erhalten diese die Möglichkeit, verschiedene akustische Umgebungen virtuell zu testen und das Klangbild seines Stücks entsprechend anzupassen. Dies ermöglicht eine präzisere und realistischere Vorstellung davon, wie die Musik in verschiedenen Szenarien wahrgenommen wird.

Bei der Entwicklung solcher Plugins sind mehrere wichtige Aspekte zu berücksichtigen. Einer davon ist die Echtzeitfähigkeit. Da Musiker:innen und Toningenieure:innen oft sofortiges Feedback benötigen, um ihre Arbeit effektiv zu gestalten, müssen Raumakustiksimulations-Plugins in der Lage sein, die akustischen Effekte in Echtzeit zu berechnen und anzuzeigen. Dies erfordert eine optimierte Codebasis und effiziente Algorithmen, um eine nahtlose Integration in die Produktionsumgebung zu gewährleisten. Darüber hinaus ist eine schnelle Berechnung entscheidend, um die Arbeitsabläufe nicht zu beeinträchtigen und eine reibungslose Interaktion mit anderen Audio-Plugins zu ermöglichen. Die Benutzerfreundlichkeit und die Möglichkeit, die Simulationseinstellungen intuitiv anzupassen, sind ebenfalls entscheidende Faktoren, um sicherzustellen, dass die Plugins effektiv und effizient eingesetzt werden können.

Das Ziel dieser Arbeit ist es, ein leistungsfähiges und effizientes Audio-Plugin zu entwickeln, das es ermöglicht, die Raumakustik in verschiedenen Anwendungsszenarien zu simulieren. Durch die Integration dieses Plugins in gängige Audio-Workstation-Software können Musi-

ker, Toningenieure und Raumakustiker die akustischen Eigenschaften von Räumen virtuell modellieren und optimieren, um bessere Klangbedingungen zu schaffen.

Die Arbeit konzentriert sich hierbei auf einen Ansatz der Spiegelschallquellen-Methode aus der geometrischen Akustik. Zuerst wird ein Überblick über verschiedene Simulationsmethoden gegeben. Danach wird ein für rechteckige Räume spezialisierter Ansatz betrachtet, welcher im nächsten Schritt optimiert wird. Es werden 3 unterschiedliche Implementationen angefertigt. Zum einen werden der unoptimierte und der optimierte Ansatz in Python umgesetzt. Der letztere wird dann abschließend mithilfe von C++ im JUCE-Framework implementiert, mit welchem dann das Plugin entwickelt wird. Die verschiedenen Ansätze werden auf die Abhängigkeit der Berechnungszeiten mit den Parametern untersucht. Außerdem werden die Ergebnisse mit dem Akustiksimulationsprogramm EASE verglichen.

2 Methodik

2.1 Spiegelschallquellen-Methode in rechteckigen Räumen

Es gibt viele unterschiedliche Ansätze, um die Ausbreitung von Schall zu simulieren. Dabei unterscheidet man zwischen der Betrachtung im Frequenzbereich und dem Zeitbereich. Zu den Ansätzen des Frequenzbereichs gehören zum Beispiel die „Boundary Element Method“ (BEM) oder die „Finite Element Method“ (FEM). Im Zeitbereich finden sich Ansätze wie die „Finite Difference Time Domain Method“ (FDTD) oder auch Ansätze der geometrischen Akustik, die in dieser Arbeit im Fokus stehen (Vorländer, 2020, S. 145). Auch diese enthält unterschiedliche Herangehensweisen. Raytracing beschreibt, einen stochastischen Ansatz bei dem eine begrenzte Anzahl an „Rays“ in zufällige Richtungen von der Schallquelle in den Raum gegeben werden. Diese bewegen sich mit Schallgeschwindigkeit durch den Raum und werden an den Wänden reflektiert. Jeder Strahl startet mit der gleichen Energie, die über die Zeit durch die Raumwände und den allgemeinen Energieverlust durch die Luft gedämpft wird. Am Empfänger wird überprüft welche Strahlen zu welchem Zeitpunkt und mit welcher Restenergie eintreffen. Daraus wird die Impulsantwort gebildet (Schröder, 2011, S. 59). Die in dieser Arbeit betrachtete Methode ist die Spiegelschallquellen-Methode.

2.1.1 Allgemeiner Ansatz zur Ermittlung einer Raumimpulsantwort

Das Spiegelschallquellen-Model gehört zu den Ansätzen aus der Gruppe der geometrischen Akustik (Vorländer, 2020, S. 146). Es wird davon ausgegangen, dass sich der Schall von einem definierten Punkt gleichmäßig im Raum in einem Radius verteilt. Mit der Annahme einer ebenen Welle bewegen sich „Schallstrahlen“ durch den Raum und es kommt bei ihrer Ausbreitung zu Energieverlust je höher die zurückgelegte Distanz ist ($\frac{1}{r^2}$). Ein Empfänger wird als Punkt im Raum angenommen. Die Schallquelle wird an jeder Wand des Raumes gespiegelt, um die Spiegelschallquellen zu erhalten. Dieser Prozess kann beliebig oft wiederholt werden, um Spiegelschallquellen höherer Ordnung zu erhalten. Diese werden benötigt, um den Weg der Schallreflexionen zu bestimmen. Die Ordnung entspricht der Anzahl der Reflexionen, die ein Schallstrahl nimmt, bevor er am Empfänger eintrifft. Um die Reflexionen zu bestimmen

wird ein Vektor vom Empfänger-Punkt zu einer Spiegelschallquelle erzeugt. Der Schnittpunkt des Vektors mit der Wand bestimmt den Punkt der Reflexion des Schalls an der jeweiligen Wand. Im Falle einer höheren Ordnung wird der Vektor zwischen den Spiegelschallquellen und dem Schnittpunkt der Wand mit der vorherigen Quelle erzeugt (Vorländer, 2020, S. 195).

Ein großer Anteil der Schallstrahlen sind ungültig, da diese entweder durch andere Wände blockiert werden oder der Schnittpunkt außerhalb der Wandfläche liegt (Schröder, 2011, S. 59). Aus diesem Grund muss für jeden Strahl eine Hörbarkeitsprüfung vorgenommen werden. Hier wird jeder Strahl ab dem Empfänger nachverfolgt. Es wird überprüft, ob er auf dem Weg zur letzten Spiegelschallquelle als Erstes die Wandfläche der letzten Reflexion trifft. Wenn dies der Fall ist, wird der Strahl am Schnittpunkt reflektiert und der Prozess wird mit der nächsten Spiegelschallquelle fortgesetzt. Dieser Prozess wiederholt sich so lange bis der Schallstrahl an der Schallquelle eintrifft. Nur wenn alle Pfade gültig sind ist die Spiegelschallquelle „hörbar“.

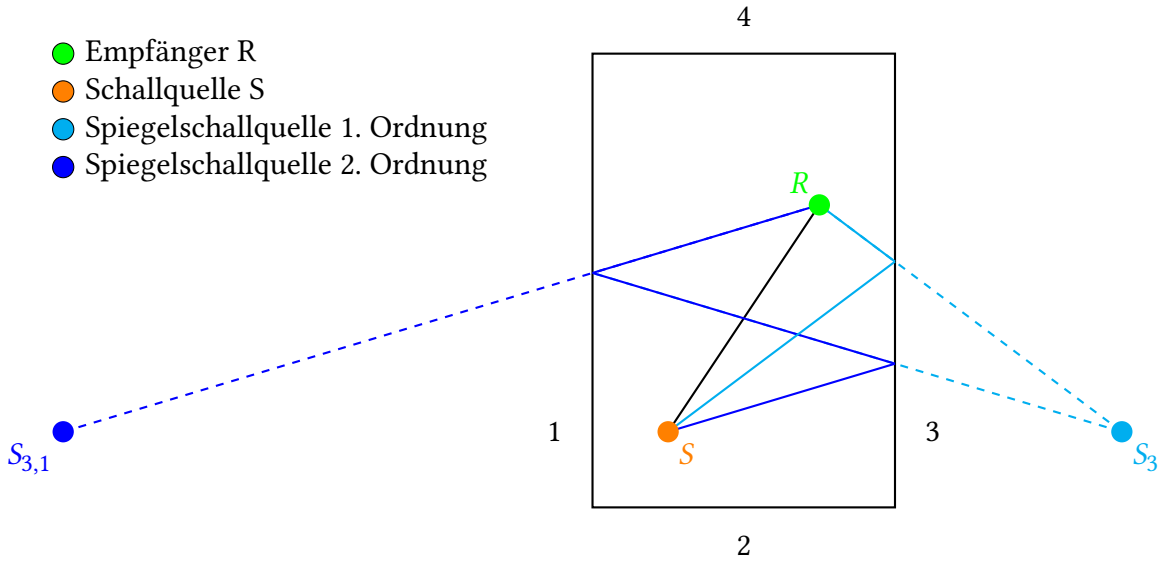

Abbildung 2.1: Konstruktion einer Spiegelschallquelle 1. und 2. Ordnung

In Abbildung 2.1 erkennt man das Prinzip. Die Schallquelle S wird an Wand 3 gespiegelt. Daraus lässt sich eine Reflexion 1. Ordnung bestimmen. Durch erneutes Spiegeln an Wand 1 lässt sich eine Reflexion 2. Ordnung und somit Punkt $S_{3,1}$ erzeugen. In diesem Beispiel sind beide Reflexionen gültig, da die Schnittpunkte innerhalb der jeweiligen Wände liegen und nicht von anderen Wänden blockiert werden. Dieser Prozess wird an jeder Wand des Raumes wiederholt.

Abschließend wir die Länge aller relevanten Pfade bestimmt. Jede gültige Reflexion zeigt sich innerhalb der Impulsantwort als einzelner Impuls. Dieser wird durch die Länge des jeweiligen Pfades verzögert und durch die Absorption an den jeweiligen Wänden und der Luft gedämpft (Schröder, 2011, S. 54).

Nachteile

Die Spiegelschallquellen-Methode ist eine rein deterministische Methode. Ihr liegt der Annahme zugrunde, dass der Reflexionsfaktor winkelunabhängig ist. Auch die Streuung des Schalls an Wänden und durch Objekte wird vernachlässigt (Vorländer, 2020, S. 210). Ebenso wird die Beugung des Schalls, aufgrund der zunehmenden Komplexität der Berechnung nicht mit in Betracht gezogen (Vorländer, 2020, S. 200). Diese Eigenschaften haben jedoch besonderen Einfluss auf die späteren Reflexionen des Nachhalls, weshalb diese Methode nicht optimal für deren Bestimmung geeignet ist.

Außerdem führt die Ermittlung von Schallquellen höherer Ordnung zu einem enormen Anstieg der Berechnungszeit. Wenn keine Maßnahmen zur Vorverarbeitung getroffen werden, steigt die Anzahl der zu berechnenden Quellen exponentiell. Die hörbaren Quellen steigen zudem allerdings nur mit t^3 der Impulsantwort (Vorländer, 2020, S. 198). Viele Berechnungen sind zusätzlich redundant. Somit ist diese Methode für Räume mit komplexen Raumgeometrien und vielen Wänden (Flächen) bei hohen Ordnungen unpraktikabel.

Vorteile

Die deterministischen Eigenschaften dieser Methode bringen dagegen eine sehr gute zeitliche Auflösung der Impulsantwort mit sich. Anders als zum Beispiel Raytracing, eignet sie sich somit besonders zur Ermittlung des Direktschalls und der frühen Reflexionen. Hier sind niedrige Ordnungen gefragt.

Aufgrund dieser Eigenschaften wird oft eine Kombination aus Raytracing und Spiegelschallquellen-Methode verwendet, in denen die jeweiligen Vorteile ausgenutzt werden. Für die Bestimmung der frühen Reflexionen wird meist die Spiegelschallquellen-Methode verwendet, um ein möglichst akurates Ergebnis, bei gleichzeitig kurzer Berechnungszeit, zu erhalten. Die späteren Reflexionen werden dann durch einen Raytracing-Algorithmus bestimmt. Dieser zeigt im Vergleich bei späteren Reflexionen eine höhere Effizienz (Naylor, 1992).

2.1.2 Spezialisierter Ansatz für „Schuhkarton“-Räume

In dieser Arbeit wird sich wie bereits beschrieben auf die Implementation der Spiegelschallquellen-Methode in quaderförmigen Räumen, auch „Schuhkarton“-Räume genannt, konzentriert. Ein solcher Raum besteht aus 6 Wänden, die unterschiedliche Wandmaterialien aufweisen können. Rechteckige Räume sind dabei die am häufigsten vorliegende Art von Raumgeometrien und aus diesem Grund relevant zur weiteren Betrachtung. Ihre Eigenschaften bringen geometrische Besonderheiten mit sich, die den allgemeinen Ansatz zur Bestimmung einer Impulsantwort vereinfachen (Allen & Berkley, 1979).

So kann zum Beispiel die Hörbarkeitsüberprüfung der ermittelten Schallquellen vernachlässigt werden. In rechteckigen Räumen ergibt sich eine regelmäßige Gitteranordnung von Spiegelschallquellen, bei den mehrere Schallquellen auf den selben Punkt fallen. Dies lässt sich in Abbildung 2.2 sehen. Es kann gezeigt werden, dass jeweils nur eine Quelle hörbar ist (Vorländer, 2020, S. 199).

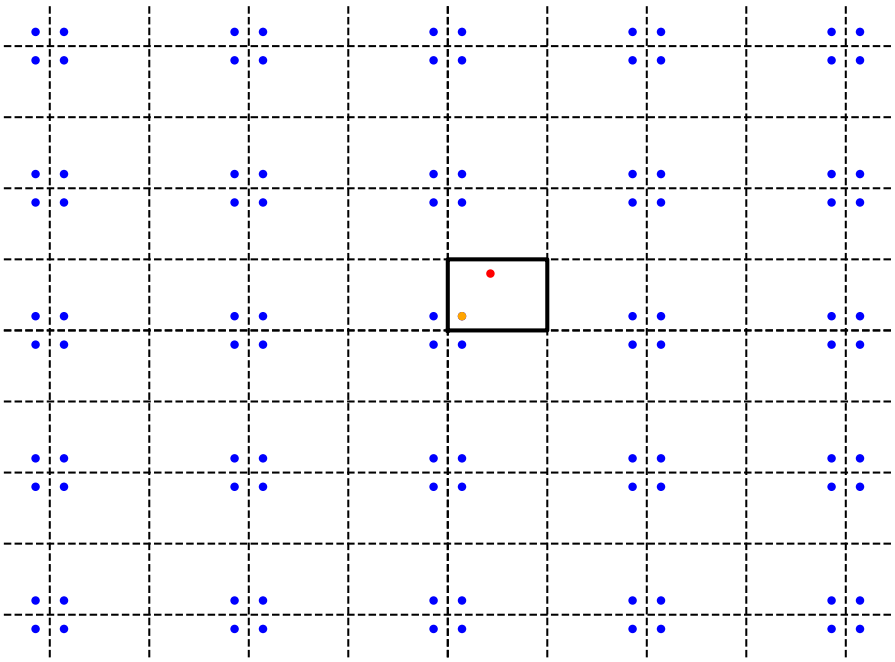

Abbildung 2.2: Gleichmäßige Anordnung der Spiegelschallquellen (gelb: Schallquelle, rot: Empfänger, blau: Spiegelschallquellen)

Außerdem vereinfacht sich die Bestimmung der Pfadlänge eines Schallstrahls. Statt die Vektorlängen zwischen den jeweiligen Wandschnittpunkten und dem Empfänger sowie der Quelle zu bestimmen, ergibt sich die Gesamtlänge der Reflexion aus dem Abstand zwischen dem Empfänger und der jeweiligen Spiegelschallquelle. Aus diesen genannten Gründen fallen viele Berechnungen, die für komplexere Raumgeometrien benötigt werden weg. Es müssen keine Wandschnittpunkte mehr ermittelt werden.

Das Model wird anhand eines Ansatzes nach Fu und Li, 2015 in Python implementiert. Dieser beruht grundlegend auf den Erkenntnissen von Allen und Berkley, 1979.

Zur Bestimmung der Impulsantwort müssen zuerst die Raumeigenschaften festgelegt werden. Der Raum wird in einem 3-dimensionalen Koordinatensystem mit Ursprung (0,0,0) definiert. Die Raumdimensionen sind L_x, L_y, L_z, mit den Reflexionsfaktoren $\beta_{x_1}, \beta_{x_2}, \beta_{y_1}, \beta_{y_2}, \beta_{z_1}, \beta_{z_2}$ der jeweiligen Wände. Die Schallquelle befindet sich am Punkt $(\tilde{x}_0, \tilde{y}_0, \tilde{z}_0)$ und der Empfänger am Punkt (x_0, y_0, z_0). Die Umrechnung der Absorptionsgrade a der Materialien in

Reflexionskoeffizienten erfolgt nach der Formel (Vorländer, 2020, S. 198)

$$\beta = \sqrt{1 - a} \tag{2.1}$$

Die Ermittlung der Impulsantwort erfolgt analog zum allgemeinen Ansatz. Jede Reflexion zeigt sich als verzögerter Impuls. Hierfür wird die Verzögerungszeit τ ermittelt

$$\tau = \frac{|\mathbf{d}|}{c} \tag{2.2}$$

mit der Schallgeschwindigkeit c und der Distanz $\mathbf{d}$ zwischen einer Schallquelle und dem Empfänger.

Der Distanzvektor $\mathbf{d}$ berechnet sich nach

$$\mathbf{d} = \begin{bmatrix} (x_0 - \tilde{x}_0 + 2p_x\tilde{x}_0) + (2m_x + L_x) \\ (y_0 - \tilde{y}_0 + 2p_y\tilde{y}_0) + (2m_y + L_y) \\ (z_0 - \tilde{z}_0 + 2p_z\tilde{z}_0) + (2m_z + L_z) \end{bmatrix} \tag{2.3}$$

bei dem p_x, p_y, p_z gleich 0 ist, falls die Reflexionsordnung gerade ist und andernfalls 1. Außerdem sind $m_x, m_y, m_z \in \mathbb{Z}$. Jede Kombination aus p_n und m_n stellt eine Spiegelschallquelle dar. Um die insgesamt möglichen Reflexionen zu bestimmen, muss über alle möglichen Kombinationen iteriert werden. In Python wird über alle möglichen Spiegelschallquellen iteriert:

Codeblock 2.1: Iteration über alle möglichen Spiegelschallquellen und Ermittlung der dazugehörigen Distanz

```
for mx in range(-Nx,Nx+1):
  for px in range(2):
    # Berechung der x-Distanz
    mpx = (r_pos[0]- s_pos[0] + 2 * px * s_pos[0]) + (2 * mx * Lx)
    for my in range(-Ny,Ny+1):
      for py in range(2):
        # Berechung der y-Distanz
        mpy = (r_pos[1]- s_pos[1] + 2 * py * s_pos[1]) + (2 * my * Ly)
        for mz in range(-Nz,Nz+1):
          for pz in range(2):
            # Berechung der z-Distanz
```

```
12              mpz = (r_pos[2] - s_pos[2] + 2 * pz * s_pos[2]) + (2 * mz * Lz)
13              # Berechnung der Gesamtdistanz
14              abs_vec = np.linalg.norm([mpx,mpy,mpz])
```

Der Abklingkoeffizient α des Impulses berechnet sich zudem nach

$$\alpha = \frac{\beta_{ges}}{4\pi|\mathbf{d}|} \tag{2.4}$$

Er entspricht der resultierenden Amplitude des Impulses. Der Gesamtreflexionskoeffizient β_{ges} eines Strahls ergibt sich aus den einzelnen Koeffizienten der getroffenen Wände:

$$\beta_{ges} = \beta_{x_1}^{|m_x+p_x|}\beta_{x_2}^{|m_x|}\beta_{y_1}^{|m_y+p_y|}\beta_{y_2}^{|m_y|}\beta_{z_1}^{|m_z+p_z|}\beta_{z_2}^{|m_z|} \tag{2.5}$$

Rein theoretisch ist die Anzahl der Reflexionen unendlich. Aus diesem Grund muss bestimmt werden, wie viele Reflexionen relevant sind, also welche Werte m_x, m_y, m_z annehmen müssen. Allgemein ist $-N_u < m_u < N_u$ mit $u \in x, y, z$. Um N_u zu bestimmen, kann die frequenzabhängige Nachhallzeit T_{60} verwendet werden. Diese wird mithilfe einer angepassten Variante der Sabine Formel ermittelt. Dabei wird auch die Frequenzabhängigkeit der Absorption durch Luft mit einbezogen (Kinsler et al., 2000, S. 336). N_u ergibt sich somit

$$N_u = \frac{c[T_{60}(f)]}{2L_u} \tag{2.6}$$

Die Gesamtanzahl der Spiegelschallquellen berechnet sich nach

$$N_{ges} = 2^3(2N_x + 1)(2N_y + 1)(2N_z + 1) \tag{2.7}$$

Die Gesamtimpulsantwort ergibt sich aus der Summe aller einzelnen Reflexionen gedämpft von α_k und vezögert um τ_k:

$$h(t) = \sum_k \alpha_k \delta(t - \tau_k) \tag{2.8}$$

oder in diskretisierter Form mit Verwendung der Abtastrate f_s:

$$h(n) = \sum_k \alpha_k \delta(n - \tau_k f_s) \tag{2.9}$$

Diese Form der Diskretisierung mit Rundung des Zeitwertes an den nächsten Integer Samplewert wird für den Zweck der Auralisation als genau genug angesehen (Fu & Li, 2015).

Um eine möglichst korrekte Darstellung über einen größeren Frequenzbereich zu erhalten, wird die Simulation für jeden Punkt nicht nur einmal für eine Frequenz, sondern über mehrere Frequenzen durchgeführt. Die hierfür ausgewählten Frequenzen sind 125 Hz, 250 Hz, 500 Hz, 1000 Hz, 2000 Hz und 4000 Hz. Die Auswahl der Frequenzen orientiert sich an den Angaben der Absorptionsgrade verschiedener Materialien. Sie decken einen Frequenzbereich ab, der besonders für die Sprach- und Musikwiedergabe relevant ist (Kinsler et al., 2000, S. 338). Man erhält somit sechs Impulsantworten. Daraus ergibt sich die Annahme, dass eine einfache Überlagerung der einzelnen Antworten zu einer besseren Abbildung der akustischen Eigenschaften des Raumes über einen breiteren Frequenzbereich führt.

2.1.3 Optimierung durch Reduktion redundanter Berechnungen

Der in Abschnitt 2.1.2 erläuterte Ansatz beinhaltet viele Berechnungen, die redundant oder irrelevant sind. In McGovern, 2009, wird eine Optimierung der Methode vorgeschlagen. Sie beinhaltet die Verwendung von Lookup-Tabellen, um redundante Berechnungen zu vermeiden. Außerdem werden diese sortiert, um irrelvante Daten zu vernachlässigen. Im vorherigen Ansatz findet eine Bestimmung aller möglichen Reflexionen statt, dabei spielt es keine Rolle ob die gerade bestimmte Reflexion noch einen Einfluss auf die Impulsantwort hat. In diesem Ansatz findet keine Trennung zwischen der Berechnung der Punkte, der Gesamtreflexionskoeffizienten und der Verrechung zur Impulsantwort statt. Dadurch müssen viele Punkte mehrmals berechnet werden. Um dies zu verhindern, werden die Schritte Berechnung und Zusammenführung voneinander getrennt. Es werden Lookup-Tabellen erstellt, die die Distanzen und die korrespondierenden Koeffizienten enthalten. Bei der Erzeugung der Tabellen werden die Werte nach Distanz sortiert. Das bedeutet, dass die kürzeste Distanz (der Direktschall) und der dazugehörige Koeffizient am Anfang der Tabellen stehen. Das ist hilfreich, um unnötige Berechnungen zu verhindern. Dadurch, dass jede Distanz von einer Spiegelschallquelle zum Empfänger durch Multiplikation mit der Schallgeschwindigkeit der Zeit entspricht, die der Schall benötigt, kann auch die Nachhallzeit als eine Distanz angesehen werden. Im 3-Dimensionalen bildet sie also eine Kugel um den Empfänger. Alle Punkte die außerhalb dieser Kugel liegen, haben keinen Einfluss auf die Impulsantwort. Durch die Sortierung der Werte kann die Berechnung der Impulsantwort somit am ersten Wert unterbrochen werden, der außerhalb dieses Radius liegt. Dies führt zu einer deutlichen Abnahme der Berechnungszeit.

Die Impulsantwort berechnet sich nach

$$h(t) = \sum_{i=0}^{2N_x} \sum_{j=0}^{2N_y} \sum_{k=0}^{2N_z} p(t)_{ijk}. \tag{2.10}$$

Der Bereich von v wird jeweils von $-N_u \leq v \leq N_u$ auf $0 \leq v \leq 2N_u$ mit $u \in x, y, z$ und $v \in i, j, k$ verschoben, um negative Werte in den Lookup-Tabellen zu verhindern. Außerdem werden die Raumdimensionen anders verarbeitet. Der Raum wird auf den Koordinaten-Ursprung $(0, 0, 0)$ zentriert. Somit werden die Angaben der Raumdimensionen halbiert und die Positionen respektive angepasst. Am Beispiel der x-Achse ergeben sich diese Raumangaben: x_s beschreibt die Distanz zwischen der Schallquelle und dem Ursprung, x_w die Distanz zwischen den Wänden und dem Ursprung und x_e die Distanz zwischen dem Empfänger und dem Ursprung. Außerdem repräsentieren $B_{x=-x_w}$ und $B_{x=x_w}$ die beiden Reflexionskoeffizienten der Wände.

Um die Sortierung der Werte nach der Distanz vorzunehmen, muss bestimmt werden, welche Schallreflexionen zuerst am Empfänger eintreffen. Der Direktschall C_0 trifft immer als Erstes ein. Bei der Betrachtung im Eindimensionalen kann gezeigt werden, dass der Schall von den Spiegelschallquellen höherer Ordnung in Paaren am Empfänger eintrifft. Es gibt vier mögliche Muster, in denen der Schall zum Empfänger gelangen kann. Jedes der Paare enthält ein Echo von der linken Seite und eines von der rechten. Beide Echos besitzen die selbe Ordnung. Außerdem zeigt sich, dass die Schallquellen höherer Ordnung spätere Echos erzeugen als niedrigere. Das bedeutet, dass zum Beispiel ein Echo einer Schallquelle 3. Ordnung immer ein späteres Echo erzeugt als eines der 2. oder 1. Ordnung. Aus dieser Feststellung lassen sich die folgenden Regeln schlussfolgern:

- **Für Paare mit ungerader Ordnung:** Wenn $x_e + x_s \leq 0$ gilt, trifft das linke Echo zuerst ein. Wenn $x_e + x_s > 0$ gilt, trifft das rechte Echo zuerst ein.
- **Für Paare mit gerader Ordnung:** Wenn $x_e - x_s \leq 0$ gilt, trifft das linke Echo zuerst ein. Wenn $x_e - x_s > 0$ gilt, trifft das rechte Echo zuerst ein.

Mithilfe dieser Regeln lässt sich bestimmen, in welcher Reihenfolge die Reflexionen eintreffen.

Für jede Koordinatenachse werden zwei Tabellen erstellt. Die erste beinhaltet die quadrierten Distanzen der jeweiligen Spiegelschallquelle vom Empfänger und die zweite die resultierenden Reflexionskoeffizienten. Somit ergeben sich für die drei Achsen insgesamt sechs Tabellen. Im Folgenden wird das Vorgehen für eine der Achsen betrachten, um das Prinzip zu veranschaulichen. Die Operationen werden aber auf alle drei Achsen angewendet. Der erste Eintrag enthält den Direktschall:

$$x_{C_0} = x_s - x_e \tag{2.11}$$

$$x_{L_n} = \begin{cases} x_{L_{n-1}} - 2(x_w + x_s) & \text{für ungerade } n \\ x_{L_{n-1}} - 2(x_w - x_s) & \text{für gerade } n \end{cases} \tag{2.12}$$

$$x_{R_n} = \begin{cases} x_{R_{n-1}} + 2(x_w - x_s) & \text{für ungerade } n \\ x_{R_{n-1}} + 2(x_w + x_s) & \text{für gerade } n \end{cases} \tag{2.13}$$

mit Startwerten $x_{L_0} = x_{R_0} = x_{C_0}$

Die Reflexionskoeffiziententabelle startet mit $a_0 = 1$ da der Direktschall nicht durch Wände absorbiert wird.

$$a_{L_n} = \begin{cases} B_{x=-x_w} a_{L_{n-1}} \\ B_{x=x_w} a_{L_{n-1}} \end{cases} \tag{2.14}$$

$$a_{R_n} = \begin{cases} B_{x=x_w} a_{R_{n-1}} \\ B_{x=-x_w} a_{R_{n-1}} \end{cases} \tag{2.15}$$

mit

$$a_{L_0} = a_{R_0} = a_{C_0} \tag{2.16}$$

Mit diesen Werten können nun die Tabellen befüllt werden. Der erste Eintrag der Distanz-Tabelle:

$$(x^2)_0 = (x_{C_0})^2 \tag{2.17}$$

und für die nachfolgenden

$$(x^2)_{2i-1} = \begin{cases} (x_{L_{n=i}})^2 & \text{für ungerade } i \text{ und } x_e + x_s \leq 0 \\ (x_{L_{n=i}})^2 & \text{für gerade } i \text{ und } x_e - x_s \leq 0 \\ (x_{R_{n=i}})^2 & \text{für ungerade } i \text{ und } x_e + x_s > 0 \\ (x_{R_{n=i}})^2 & \text{für gerade } i \text{ und } x_e - x_s > 0 \end{cases} \tag{2.18}$$

$$(x^2)_{2i} = \begin{cases} (x_{R_{n=i}})^2 & \text{für ungerade } i \text{ und } x_e + x_s \leq 0 \\ (x_{R_{n=i}})^2 & \text{für gerade } i \text{ und } x_e - x_s \leq 0 \\ (x_{L_{n=i}})^2 & \text{für ungerade } i \text{ und } x_e + x_s > 0 \\ (x_{L_{n=i}})^2 & \text{für gerade } i \text{ und } x_e - x_s > 0 \end{cases} \tag{2.19}$$

Ähnlich wird bei den Reflexionskoeffizienten vorgegangen. Der erste Eintrag der Tabelle ist

$$a_0 = a_{C_0} \tag{2.20}$$

und für die nachfolgenden

$$a_{2i-1} = \begin{cases} a_{L_{n=i}} & \text{für ungerade } i \text{ und } x_e + x_s \leq 0 \\ a_{L_{n=i}} & \text{für gerade } i \text{ und } x_e - x_s \leq 0 \\ a_{R_{n=i}} & \text{für ungerade } i \text{ und } x_e + x_s > 0 \\ a_{R_{n=i}} & \text{für gerade } i \text{ und } x_e - x_s > 0 \end{cases} \tag{2.21}$$

$$a_{2i} = \begin{cases} a_{R_{n=i}} & \text{für ungerade } i \text{ und } x_e + x_s \leq 0 \\ a_{R_{n=i}} & \text{für gerade } i \text{ und } x_e - x_s \leq 0 \\ a_{L_{n=i}} & \text{für ungerade } i \text{ und } x_e + x_s > 0 \\ a_{L_{n=i}} & \text{für gerade } i \text{ und } x_e - x_s > 0 \end{cases} \tag{2.22}$$

Jeder Eintrag in der Distanz-Tabelle an der Position i kann dem entsprechenden Wert in der Koeffizienten-Tabelle zugewiesen werden. Die Werte können nach der Formel 2.10 verrechnet werden. Für den ersten Wert in den jeweiligen Summen, der den Distanzwert der ermittelten Nachhallzeit übertrifft, wird die Berechnung dieser Summe gestoppt und mit der nächsten Summe fortgeführt.

Codeblock 2.2: Berechnung der Impulsantwort mithilfe der Lookup-Tabellen

```
for i in range(len(x_dist)):
  for j in range(len(y_dist)):
    for k in range(len(z_dist)):
      # Kalkulation der Distanz durch Zugriff auf die Distanz-Tabellen
      dist = np.sqrt(x_dist[i] + y_dist[j] + z_dist[k])
      # Umrechnung in Zeit
      t = dist / speed_of_sound
      # Diskretisierung
      sample = int(t * sample_rate)
      # Überprüfung ob Zeitpunkt im relevanten Bereich liegt
      if sample >= rir_len:
        break
      # Berechnung der Amplitude durch Zugriff auf die Koeffizienten-Tabellen
      rir[sample] += x_coeff[i]*y_coeff[j] * z_coeff[k] / dist
    if k == 0:
      break
  if j == 0:
    break
```

2.1.4 Reduktion von Sweeping Echoes

Bei der Auralisation und Analyse der erzeugten Impulsantworten lässt sich ein besonderes Phänomen feststellen. Es zeigt sich ein linearer Anstieg einiger Frequenzanteile, sogenannte „Sweeping Echoes“. Dieses Echo besteht aus einem Hauptanteil und mehreren kleineren Frequenzanstiegen. Wenn der simulierte Raum hörbar gemacht wird, lässt sich deutlich ein linearer Anstieg der Frequenzkomponenten des Signals hören (Kiyohara et al., 2002).

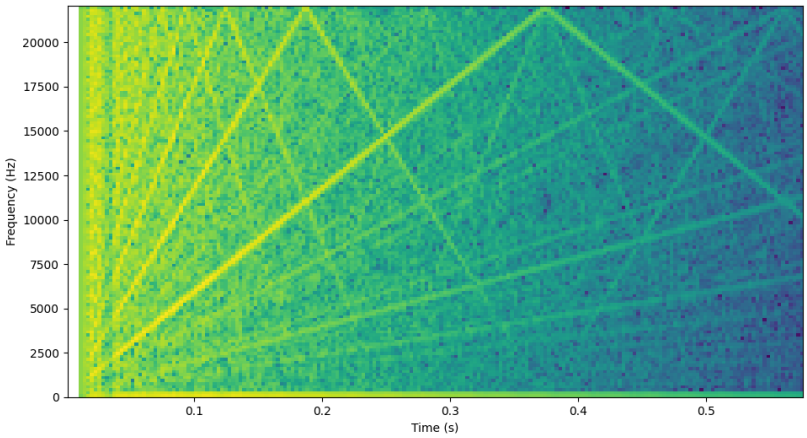

Abbildung 2.3: Spektrogram einer Impulsantwort eines Raumes mit den Dimensionen 5m x 4m x 5m

In Abbildung 2.3 zeigt sich der lineare Anstieg der Frequenzanteile sehr gut. Diese Echos werden durch die zeitliche Regelmäßigkeit der Reflexionen von Spiegelschallquellen höherer Ordnung erzeugt. Dies geschieht aufgrund der exakten Parallelität der gegenüberliegenden Wände und die exakten 90° Winkel zwischen den einzelnen Wänden. Es kann argumentiert werden, dass diese Eigenschaften in der Realität selten anzutreffen sind (Sena et al., 2015). Sena et al., 2015 schlagen einen Lösungsansatz zum Entfernen der sweeping echoes vor. Eine Möglichkeit wäre es, die Simulation mit leichten Unregelmäßigkeiten der Geometrie zu simulieren. Das würde bedeuten, dass eine Berechnung mit der Spiegelschallquellen-Methode nicht mehr möglich wäre. Hierzu müsste ein Ansatz, der komplexere Raumgeometrien beinhaltet, angewendet werden. Dies würde die Berechnungen deutlich komplexer und aufwändiger machen. Die Unregelmäßigkeiten in Räumen lassen sich aber auch durch leichte zufällige Verschiebung γ der Spiegelschallquellen auf der Verbindungslinie zwischen der Quelle und dem Empfänger erzeugen. Dies entspricht einer zufälligen Verzögerung jedes Impulses um $\frac{\gamma}{c}$. In Python lässt sich die Verzögerung einfach umsetzen.

```
1 t += np.random.uniform(-random_factor, random_factor) / speed_of_sound
```

In der Arbeit wird vorgeschlagen eine zufällige Verschiebung der Quellen um $\gamma_{max} = 8cm$ zu wählen. Die Genauigkeit des Ergebnisses wird nicht zu sehr beeinträchtigt, die Sweeping Echoes werden dadurch aber in den meisten Fällen entfernt oder abgeschwächt.

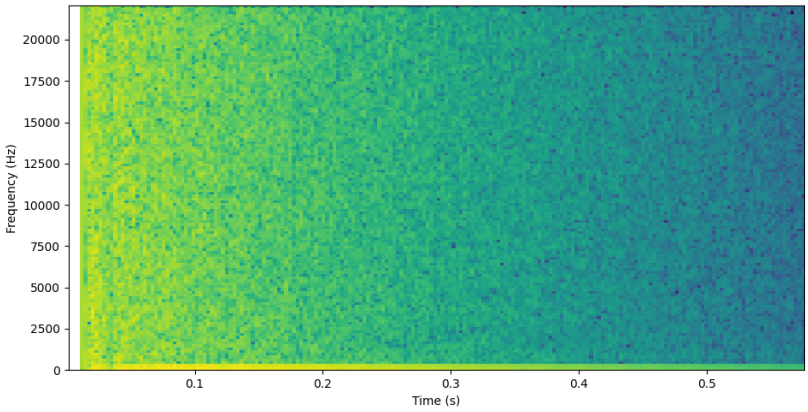

Abbildung 2.4: Spektrogram des Raums aus Abbildung 2.3 nach zufälliger Verschiebung der Spiegelschallquellen

Nach dieser Operation ergibt sich das Spektrum in Abbildung 2.4. Hier lässt sich eindeutig erkennen, dass die linearen Frequenzanstiege nicht mehr vorhanden sind.

2.2 Entwicklung eines Audio-Plugins

In der sich stetig weiterentwickelnden digitalen Welt gewinnt Audio-Software zunehmend an Bedeutung. Ein Audio-Plugin stellt eine Software dar, die in der Lage ist, Audiodateien zu verarbeiten und zu modifizieren. Diese Plugins können eine Vielzahl von Funktionen umfassen, darunter die Bearbeitung eingehender Audiodaten mittels Effekten oder Filtern sowie die Umsetzung virtueller Instrumente. Zur Integration dieser Plugins werden sogenannte Plugin-Host-Anwendungen benötigt, wie beispielsweise eine Digital-Audio-Workstation (DAW), die als Plattform für die Audiobearbeitung dient (Collins, 2003). Zu den bekannten DAWs gehören beispielsweise Logic Pro, Ableton Live, Reaper oder Pro Tools. Alle genannten Programme bieten Softwareumgebungen, die es ermöglichen, Audio- und MIDI-Daten aufzuzeichnen, zu bearbeiten und zu mischen sowie geeignete Signalverarbeitung und Effekte anzuwenden. Wie der Name Plugin schon sagt, sind diese eigenständige Software, die in einer anderen Umgebung aufgerufen werden. Die meisten dieser Anwendungen bieten unterschiedliche Parameter, auf die von der Host-Software zugegriffen werden kann. Diese Parameter können somit vom Host synchronisiert und gespeichert werden um die Einstellungen erneut aufrufen zu können. Jede Instanz eines Plugins in einem Projekt funktioniert unabhängig voneinander.

Dadurch können mehrere Instanzen parallel mit unterschiedlich eingestellten Parametern verwendet werden. Außerdem können die Einstellungen auch in Form von Presets gespeichert werden.

Die Verfügbarkeit unterschiedlicher Plugin-Formate ermöglicht es Entwicklern, Programme plattform- und programmunspezifisch zu entwickeln. Dies bedeutet, dass die Plugins in verschiedenen DAWs verwendet werden können, sofern diese das entsprechende Format unterstützen. Um in einer Host-Umgebung ordnungsgemäß zu funktionieren, müssen sowohl der Host als auch das Plugin ihre jeweiligen Einstellungen und Eigenschaften kennen. Der eingehende Audiostream wird in Blöcken, sogenannten Buffern verarbeitet (Goudard & Muller, 2003).

Ein Audio-Plugin sollte dabei bestimmte Kriterien erfüllen. Ein äußerst entscheidendes Kriterium ist die Berechnungszeit. Da in einer DAW Audiosignale in „Echtzeit“ verarbeitet werden, müssen die Zeiten möglichst gering sein. Um dies zu gewährleisten, läuft die Audioverarbeitung des Plugins über einen priorisierten Thread, der nicht blockiert werden darf. Ansonsten kann es zu Klicken oder anderen Artefakten im Signal kommen. Lange Berechnungszeiten würden das Programm normalerweise verzögern.

Apple bietet mit Audio-Units (AU) ein eigenes Plugin-Format, das ausschließlich mit Apple eigenen Betriebssystemen wie MacOS oder iOS funktioniert. Das bekannteste Format ist das von Steinberg entwickelte Virtual Studio Technology Format (VST). Die Spezifikationen des Formats sowie ein Source Development Kit (SDK) wurden 1996 veröffentlicht. 2008 folgte die Erweiterung auf VST 3.0. VST ist ein offener Standard (Steinberg, n. d.). Außerdem gibt es noch weitere Formate. Zum Beispiel das Avid spezifische Avid Audio eXtension (AAX) Format. Diese Arbeit konzentriert sich auf das VST Format.

2.2.1 Vorstellung des JUCE-Frameworks

Das JUCE-Framework bietet eine Programmierumgebung, mit der sich vor allem Audio-Anwendungen entwickeln lassen. Es bietet die Möglichkeit, Standalone oder Plugin-Anwendungen zu programmieren und unterstützt alle gängigen Plugin-Formate, die in Abschnitt 2.2 besprochen wurden. Das Framework basiert auf einem Open-Source-Code, geschrieben in C++. JUCE vereinfacht das Entwickeln von Audio-Anwendungen, indem es den Prozess der Kompilierung in das gewünschte Format übernimmt. Es muss nur ein Quellcode geschrieben werden, um Anwendungen für verschiedenste Formate und Plattformen bereitzustellen. Außerdem enthält das Framework eine Digital-Signal-Processing (DSP) Bibliothek mit der viele elementare Bearbeitungsfunktionen in der Audioverarbeitung implementiert werden können. Dazu gehört zum Beispiel das Entwerfen und Anwenden von Filtern oder anderen Effekten. Neben der Audioverarbeitung bietet es außerdem eine Bibliothek, mit der sich Grafische Oberflächen (GUI) für die Anwendungen programmieren lassen. Das Framework enthält eine

sogenannten Projucer-Anwendung, mit der sich neue Projekte erstellen und verwalten lassen. Die Grundmodule sind unter einer freien ISC Open Software Licence lizensiert. Weitere Bestandteile laufen unter einer kommerziellen Lizenz, wobei die Personal-Lizenz kostenlos für alle Entwickler ist, die weniger als 50.000 USD pro Jahr mit dem Produkt verdienen („JUCE“, n. d.). Der grundlegende Aufbau eines Plugins wird in zwei Hauptbausteine aufgeteilt. Zum einen die Verabreitung des Audio Signals und zum anderen die Benutzeroberfläche. Diese Trennung findet auch im Aufbau eines JUCE-Projekts statt.

Das DSP in JUCE kann mit dem `juce::DSP` Modul verwendet werden. Diese Klasse kann genutzt werden, um Audio-Buffer zu manipulieren, zu filtern, zu oversamplen oder andere mathematische Funktionen darauf anzuwenden. In der Klasse finden sich weitere Unterklassen. Mit der Klasse `filter-design` lassen sich Filterkoeffizienten generieren. Die Klassen `widget` und `processor` bieten das Anwenden von Effekten, wie Reverb, Echo oder anderen, auf den Buffer. Für diese Arbeit ist besonders die Klasse `frequency` relevant. Hier findet sich das `Convolution` Modul, in das sich Impulsantworten laden lassen, die dann mit dem eingehenden Signal, mittels der `FFT` Klasse im Frequenzbereich gefaltet werden.

Die in Abschnitt 2.2 beschriebene Wichtigkeit der „Echtzeitfähigkeit“ spielt auch im Rahmen des Frameworks eine große Rolle. JUCE übernimmt die Aufteilung des Codes auf unterschiedliche Threads. Die `Convolution` Klasse kann Impulsantworten laden ohne den Audio-Thread zu stören („JUCE Documentation“, n. d.). Die Parameter des Plugins sollten auch automatisiert werden können, um in diesem Fall zum Beispiel eine Bewegung im Raum simulieren zu können. Im Optimalfall sollten die Berechnungen möglichst schnell durchgeführt werden, um das Signal nicht zu unterbrechen.

2.2.2 Nutzerinteraktion durch Verwendung einer Nutzeroberfläche

Ein Plugin benötigt neben der Verarbeitung der eingehenden Signale auch eine Option als Nutzer:in mit der Software zu interagieren. Eine Benutzeroberfläche (UI) ermöglicht es Anwendern:innen, verschiedene Parameter und Einstellungen des Plugins anzupassen, um den Klang nach ihren individuellen Vorstellungen zu formen. Hierdurch entsteht die Interaktion der Nutzenden mit der Berechnung der Impulsantwort im Hintergrund. Die Oberfläche kann unterschiedlichste Bedienelemente enthalten, die die einzelnen Parameter steuern. Dazu gehören unter anderem Slider, Texteingabefelder oder Dropdown-Menüs. Die Benutzeroberfläche sollte übersichtlich gestaltet sein, damit Benutzer:innen leicht erkennen können, welche Parameter verfügbar sind und wie sie eingestellt sind. Klare Beschriftungen und eine logische Anordnung der Elemente tragen dazu bei, die Benutzer:innenerfahrung zu verbessern und die Bedienung des Plugins zu erleichtern.

JUCE bietet eine Reihen von Funktionalitäten, mit denen sich die Gestaltung des UIs beeinflussen und erstellen lässt. Es gibt vorgefertigte Klassen mit denen sich die verschiedenen

Eingabe-Komponenten erstellen lassen. Diese können mit der `LookAndFeel` Klasse nach eigenen Vorstellungen designt werden. Außerdem kann hier auch die Kommunikation der vom User eingestellten Parameter mit dem Verarbeitungscode konfiguriert werden.

In dieser Arbeit wird sich hauptsächlich auf die Funktionalität und nicht auf das Aussehen des Interfaces konzentriert. Das hier entwickelte Plugin besitzt folgende veränderbare Parameter:

- Raumgröße
- Position der Schallquelle
- Position des Schallempfängers
- Materialien der Raumwände

Die Raumgrößen und Positionsangaben erfolgen über Slider für jede Koordinatenachse, respektive Raumgröße. Die verwendete Eingabeeinheit ist Meter. Neben den Slidern gibt es außerdem noch die Möglichkeit, die gewünschten Werte über ein Eingabefeld einzutragen. Die Materialien können für jede der sechs Wände einzeln über Dropdown-Menüs ausgewählt werden. In Abbildung 2.5 erkennt man die Anordnung der Parameter im Userinterface.

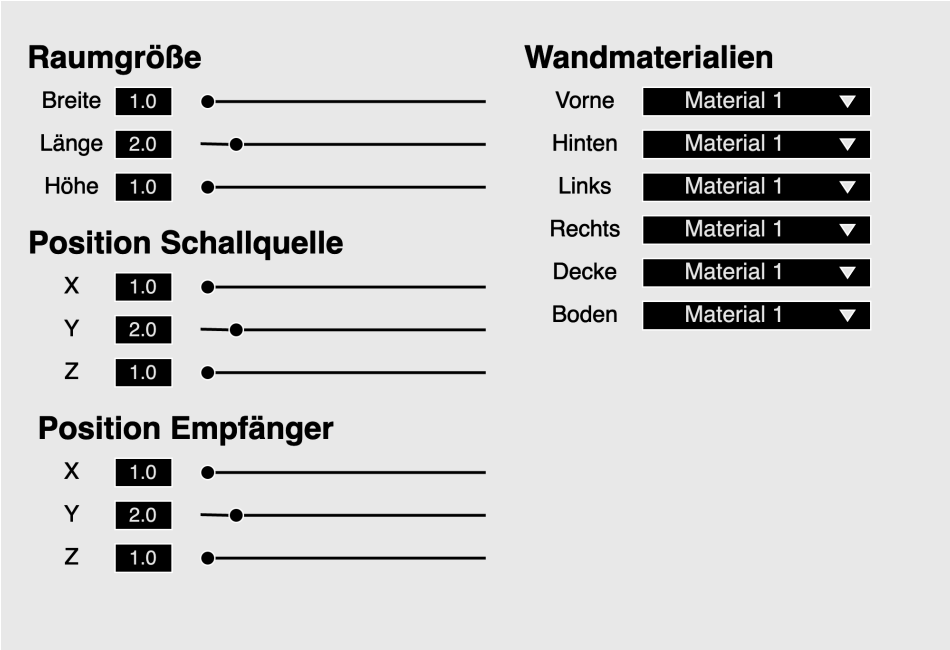

Abbildung 2.5: Das UI-Layout Konzept der Parameter

3 Durchführung

Die Umsetzung des Plugins geschieht wie beschrieben in C++ mithilfe des JUCE-Frameworks. Dieses bietet die Grundlage zur Erstellung des Plugins.

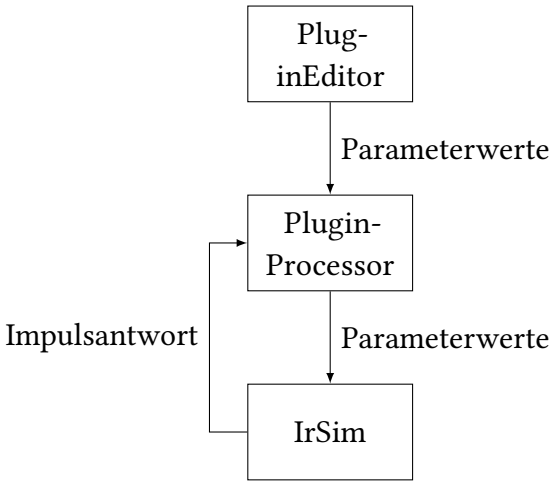

Abbildung 3.1: Aufbau der Codestruktur

Abbildung 3.1 zeigt die Struktur des Plugin-Codes. Der `PluginEditor` enthält alle UI-Elemente und übergibt die eingestellten Parameterwerte an den `PluginProcessor`. Dieser ist für die Verarbeitung der eingehenden Audio-Buffer zuständig. Er gibt die Parameter weiter an den `IrSim`. Hier wird eine neue Impulsantwort erzeugt und zurück an den `PluginProcessor` übermittelt.

3.1 Spiegelschallquellen-Methode in C++

Der Code zur Erzeugung der Impulsantwort beruht auf dem Prinzip des in 2.1.3 beschriebenen optimierten Ansatzes in Kombination mit der in 2.1.4 beschriebenen Auslöschung der Sweeping Echoes. Dieser verspricht die effizienteste Variante der Berechnung. Grundlegend ist das Skript wie folgt aufgebaut: Es enthält ein `RIRDATA` struct zur Speicherung der sechs Lookup-Tabellen. Außerdem gibt es die Klasse `ROOM`, die die Raumdaten speichert und die

Klasse `MATERIALS`, die alle Wandmaterialien mit den Absorptionskoeffizienten enthält. Die Hauptklasse ist die `RIRGENERATOR` Klasse. Hier finden die Berechnungen statt.

Die Funktion `create_full_rir` generiert eine vollständige Impulsantwort für einen Raum basierend auf den übergebenen Raum- und RIR-Daten. Zunächst werden die Frequenzen definiert, bei denen die Impulsantwort berechnet werden soll. 125 Hz, 250 Hz, 500 Hz, 1000 Hz, 2000 Hz, 4000 Hz sind die verwendeten Frequenzen. Anschließend wird die Dauer des Nachhalls (T60) für jede Frequenz berechnet, indem die Funktion `calc_t60` für jeden Frequenzwert aufgerufen wird. Sie berechnet den T60-Wert basierend auf den Raum- und Frequenzparametern. Um die Länge der Impulsantwort zu bestimmen, wird der maximale T60-Wert über alle Frequenzen ermittelt. Dieser Wert wird dann verwendet, um die Gesamtlänge des Arrays zu bestimmen, das die Impulsantwort speichern wird. Ein Array `h` wird erstellt, um die Impulsantwort zu speichern. Die Länge dieses Arrays wird basierend auf der maximalen T60-Zeit und der Abtastrate des Audiosignals berechnet. Für jede Frequenz wird eine Schleife durchlaufen, um die Impulsantwort zu generieren. Dabei werden Lookup-Tabellen für die Distanz und die Koeffizienten entlang der x-, y- und z-Achse des Raums erstellt und mit Werten gefüllt. Diese Lookup-Tabellen werden dann verwendet, um die Impulsantwort für die jeweilige Frequenz mit der `create_single_rir` zu generieren und in das Array `h` zu schreiben. Nach Berechnung der Impulsantwort für jede Frequenz wird `h` normalisiert, indem der maximale Wert berechnet und alle Werte entsprechend skaliert werden. Schließlich wird ein Audiobuffer erstellt, um die Impulsantwort zu speichern und an den Audioprozessor des Plugins zurückzugeben und der Speicherplatz für temporäre Arrays wird freigegeben, um Speicherlecks zu vermeiden.

Die Funktion `create_single_rir` erhält das `RIR_DATA` und das `ROOM` Objekt. Zusätzlich werden T60, das Array zur Speicherung der Impulsantwort und die Werte für N_x, N_y, N_z erwartet. Die Funktion berechnet zunächst die Länge der Impulsantwort in Samples basierend auf der gegebenen Nachhallzeit und der Abtastfrequenz. Anschließend wird eine Schleife durchlaufen, um die einzelnen Reflexionen für die Spiegelschallquellen zu generieren. Innerhalb der Schleife wird auf die Werte in den jeweiligen Lookup Tabellen der einzelnen Achsen zugegriffen. Für jede Position im Raum wird die Entfernung von der Spiegelschallquelle zum Empfänger berechnet. Anschließend wird die Zeit `t` berechnet, die der Schall benötigt, um diese Entfernung zu überwinden. Außerdem wird die leicht zufällige Verschiebung der Spiegelschallquellen zur Auslöschung der Sweeping Echoes angewendet. Die Funktion `rndm` erzeugt eine zufällige Zahl im Bereich von -1 bis 1. C++ verfügt über eine Standard-Funktion, die dies übernimmt. Diese ist aber nicht besonders effizient. Aus diesem Grund wurde eine eigene Funktion definiert, nach dem Vorbild von Thompson, n. d. Dieser Ansatz zeigt ein gutes Verhältnis zwischen statistischer Zufälligkeit und Geschwindigkeit. Die zufällige Verschiebung beträgt maximal 8cm (`random_factor = 0.08`). Für `C` wird ein Wert von 343,6 $\frac{m}{s}$ verwendet. Dieser entspricht der Schallgeschwindigkeit bei einer Temperatur von 20 °C (Weinzierl, 2008, S. 22).

```
        t += rndm() * random_factor / C;
```

Basierend auf der berechneten Zeit `t` wird die entsprechende Abtastposition in Samples im Impulsantwort-Array berechnet. Wenn diese Position innerhalb der Nachhallzeit liegt, wird der Beitrag dieser Reflexion zu dieser Abtastposition im Array `h` akkumuliert. Die Schleifen werden vorzeitig abgebrochen, wenn die generierten Abtastpositionen außerhalb der Länge der Nachhallzeit T60 liegen.

Die Material-Klasse enthält verschiedene Wandmaterialien sowie deren Absorptionskoeffizienten über die oben beschriebenen Frequenzen (Sengpiel, n. d.). Außerdem enthält sie die Funktion `get_ref_coeff`, die für die Umrechnung des Absorptionskoeffizienten in den Reflexionskoeffizienten sorgt.

Die `generate` Funktion ist die Schnittstelle zwischen Plugin und Impulsantworterzeugung. Hier werden die vom Plugin erhaltenen Parameter in die jeweiligen Klassen gesetzt, eine neue Impulsantwort generiert und an das Plugin in Form eines Audio-Buffers zurückgegeben. Die Funktion wird vom `PluginProcessor` in der `updateRIR` Funktion aufgerufen. Dort werden alle benötigten Parameter übergeben.

3.2 Implementation in JUCE

Das Plugin lässt sich in zwei Hauptelemente aufteilen. Das erste Element ist der Audioprocessor. Dieser ist für die Verarbeitung der eingehenden Audio-Buffer zuständig. Das zweite Element ist der Plugin Editor. In diesem lässt sich das UI des Plugins konfigurieren und einrichten.

Die Verbindung der beiden Bereiche lässt sich mit einem sogenannten `AudioProcessorValueTreeState` (`APVTS`) erzeugen. In diesem kann ein Parameterlayout erstellt werden. Hier können die Eigenschaften der Parameter definiert werden.

```
layout.add(std::make_unique<juce::AudioParameterFloat>(
          juce::ParameterID("RoomWidth",1),
          "RoomWidth",
           juce::NormalisableRange<float>(1.f,25.f,0.01f),3.f));
```

Im Codeauszug sieht man beispielhaft die Erzeugung eines Parameters. Der Parameter `RoomWidth` wird dem `layout` hinzugefügt. Es wird ein Parameter vom Typ Float erzeugt. Dieser besitzt die ID „RoomWidth“ und beinhaltet einen Bereich von float-Werten begrenzt von 1 bis 25 mit Schrittweite 0.01 und einem Startwert von 3.

Die im APVTS gespeicherten Parameter können nun vom Editor und Processor aus abgegriffen und verarbeitet oder verändert werden. Im Hintergrund wird die Synchronisation übernommen.

```
1 float roomWidth = apvts.getRawParameterValue("RoomWidth")->load();
```

APVTS wird ebenso verwendet um den aktuellen Status beim Schließen des Plugins zu speichern und diesen beim erneuten öffnen abzufragen. Mithilfe der `Convolution` Klasse wird eine neue Convolution-Engine erzeugt, die das Laden der Impulsantwort sowie die Faltung mit dem eingehenden Signal übernimmt. Eine Veränderung eines Parameters löst eine erneute Berechnung der Impulsantwort aus, die in einem Audio-Buffer an die Engine weitergegeben wird. Wie in Abschnitt 2.2.1 beschrieben, führt eine neue Berechnung nicht zu einer Unterbrechung des Signals, da die neue Impulsantwort im Hintergrund geladen wird. Es können aber Artefakte während des Ladens wahrgenommen werden.

Die Auswahl der Raumgröße beeinflusst die Parameter der Schallquellen- und der Empfängerposition in der jeweiligen Achse, damit die Position nicht außerhalb des Raumes gewählt werden kann. Da aber dynamische Veränderungen der Parameterbereiche nicht unterstützt werden, wird für die Schallquelle und den Empfänger ein normalisierter Wertebereich von 0 bis 1 verwendet. Die eingestellten Parameter werden dann auf ihre eigentlich richtige Position zurückgerechnet. Die Funktion `updateSliderRange` übernimmt das Übersetzen der normalisierten Werte.

3.3 Benutzeroberfläche

Die entwickelte Benutzeroberfläche entspricht dem Konzept aus 2.5. JUCE enthält fertige Komponenten, die für die Umsetzung verwendet wurden. Hierzu gehören die Slider und die Dropdown-Menüs (in JUCE „Combo-Boxen“ genannt). Außerdem wurden Textlabel für die Textanzeigen und die Überschriften verwendet. Um eine UI Komponente hizuzufügen, muss dieses zuerst im `PluginEditor` initialisiert werden werden. Der Ablauf wird an der Slider-Komponente gezeigt.

```
1 juce::Slider roomWidthSlider;
```

Um die Slider mit den richtigen Parameterwerten zu synchronisieren wird ein `Attachement` verwendet, welches an den APVTS gekoppelt wird.

```
1 roomWidthSliderAttach(audioProcessor.apvts, "RoomWidth", roomWidthSlider);
```

Daraufhin kann der Stil und weitere Eigenschaften der Komponente noch weiter verändert werden.

```
// Slider wird als horizontal gesetzt
roomWidthSlider.setSliderStyle(juce::Slider::SliderStyle::LinearHorizontal);
// Anpassung der Textbox des Sliders
roomWidthSlider.setTextBoxStyle(juce::Slider::TextBoxLeft, false, 50, 25);
// macht den Slider sichtbar
addAndMakeVisible(roomWidthSlider);
```

Mit sogenannten `Listener` Objekten können Veränderungen des UIs und somit auch der Parameter abgefangen und verabeitet werden.

```
roomWidthSlider.addListener(this);
```

Dieses Prinzip gilt in ähnlicher Form auch für die anderen Komponententypen. In der `resized` Funktion wird das Layout und die Anordnung der einzelnen Komponenten festgelegt.

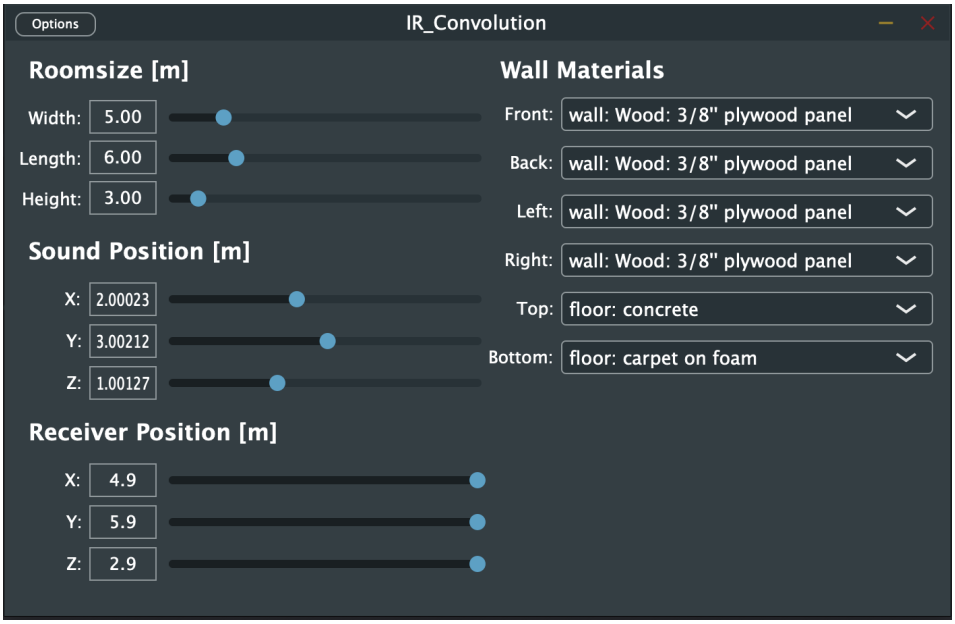

Abbildung 3.2: Die fertige Benutzeroberfläche des Plugins

4 Auswertung

Alle Tests und Analysen wurden auf einem MacBook Pro mit M2 Pro Chip durchgeführt. Der analytische Vergleich wurde ohne Einfluss der in Abschnitt 2.1.4 beschriebenen zufälligen Verschiebung der Spiegelschallquellen durchgeführt.

4.1 Analyse der Berechnungszeiten

Es werden 3 verschiedene Implementationen verglichen. Der Ansatz in Python mit und ohne Optimierung sowie die Implementation in C++ mit Optimierung. Die Ermittlung der Berechnungszeit erfolgt durch eine durchschnittliche Berechnungszeit aus 100 Ausführungen des Codes. Die Berechnungen werden mit einer Abtastrate von 44100 Hz durchgeführt. Die Berechnungszeit wird mithilfe eines Timers gemessen, der vor dem Durchlauf des Programms startet und nach der hundertsten Ausführung stoppt. Aus diesen wird dann der Mittelwert der Berechnungszeit ermittelt. Als Basis für die Analyse der Berechnungszeit wird ein Raum definiert der als Vergleich dient. Parameter des Basisraums:

- Raumgröße: 5m x 7m x 3m
- Position der Schallquelle: X: 4m Y: 3,5m Z: 2m
- Position des Schallempfängers: X: 1m Y: 3,5m Z: 1.8m
- Materialien:

  Wände: Holz

  Decke: Beton

  Boden: Teppich auf Beton

Die Dimensionen des Basisraums entsprechen einem durchschnittlichen Raum. Die Schallquelle wurde zentral an der rechten Wand des Raums platziert. Der Empfänger steht auf der linken Seite des Raums. Der Basisraum wird an verschiedenen Parameter verändert, um zu analysieren welche Parameter Einfluss auf die Berechnungszeit haben. Betrachtet werden 4 vom Basisraum abweichende Varianten:

- **Variante 1:** Alle Wandmaterialien erhalten ein Material mit hohem durchschnittlichen Absorptionskoeffizienten (Foam SDG 4")
- **Variante 2:** Die Schallquelle und der Empfänger werden weit auseinander im Raum platziert

  Schallquelle X: 1m Y: 1m Z: 1m

  Empfänger X: 4,5m Y: 6,5m Z: 2,5m
- **Variante 3:** Die Schallquelle und der Empfänger werden nah beieinander im Raum platziert

  Schallquelle X: 4,5 m Y: 6,4m Z: 2,5m

  Empfänger X: 4,5m Y: 6,5m Z: 2,5m
- **Variante 4:** Der Raum wird vergrößert

  Breite: 10m, Länge: 14m, Höhe: 6m

Tabelle 4.1: mittlere Berechnungszeiten über 100 Iterationen

	Python (ohne Lookup)	Python (Lookup)	C++ (Lookup)	Nachhallzeit
Basis	60,51 s	3,86 s	0,030 s	0,72 s
Variante 1	1,29 s	0,082 s	0,007 s	0,16 s
Variante 2	61,14 s	3,95 s	0,030 s	0,72 s
Variante 3	60,75 s	3,99 s	0,030 s	0,72 s
Variante 4	56,78 s	3,75 s	0,029 s	1,37 s

In Tabelle 4.1 zeigt sich, dass eine Veränderung der Position des Empfängers oder der Schallquelle keinen großen Einfluss auf die Berechnungszeit hat. Die Werte sind annähernd gleich. Bei Veränderung der Raumgröße lässt sich eine kleine Veränderung sehen. Eine klare Abweichung der Zeiten lässt sich allerding bei der Veränderung der Wandmaterialien sehen.

Neben dem Einfluss der einzelnen Parameter zeigen sich außerdem die Auswirkungen der Codeoptimierungen und die Wahl der Programmiersprache. Allein die Berechnung des Basisraums zeigt eine deutliche Veränderung um circa Faktor 16 mit Verwendung der sortierten Lookup-Tabellen im Vergleich zum unoptimierten Code. Eine weitere Verbesserung zeigt sich bei der Implementation in C++. Die Berechnung erfolgt hier circa um Faktor 131 schneller als die Python-Version mit Lookup-Tabellen und circa Faktor 2050 schneller als die unoptimierte Python-Variante.

In der Tabelle werden ebenfalls die Nachhallzeiten (T60) der Räume angegeben. Hier zeigt sich, dass die Berechnungszeit inZusammenhang mit der Nachhallzeit steht. Bei gleicher

Nachhallzeit unterscheiden sich die mittleren Berechnungszeiten nur marginal, aber bei einer kürzen T60 ist auch die Berechnungszeit geringer. Die Nachhallzeit in Kombination mit der jeweiligen Größe der Raumdimension bestimmt die Anzahl N der Spiegelschallquellen (siehe auch Formel 2.6), die berechnet werden müssen. Je höher N, desto mehr Berechnungen müssen durchgeführt werden.

Die Abhängigkeit zwischen den Materialien und der Berechnungszeit wird weiter betrachtet. Hierfür wurde der Basisraum mit unterschiedlichen durchschnittlichen Absorptionskoeffizienten simuliert. In Abbildung 4.1 zeigen sich die Ergebnisse. Der Kurvenverlauf der Berechnungszeit und der jeweiligen Nachhallzeit sinkt exponentiell mit steigendem Absorptionskoeffizient. Dies unterstützt die These des klaren Zusammenhangs zwischen Nachhallzeit und Berechnungszeit.

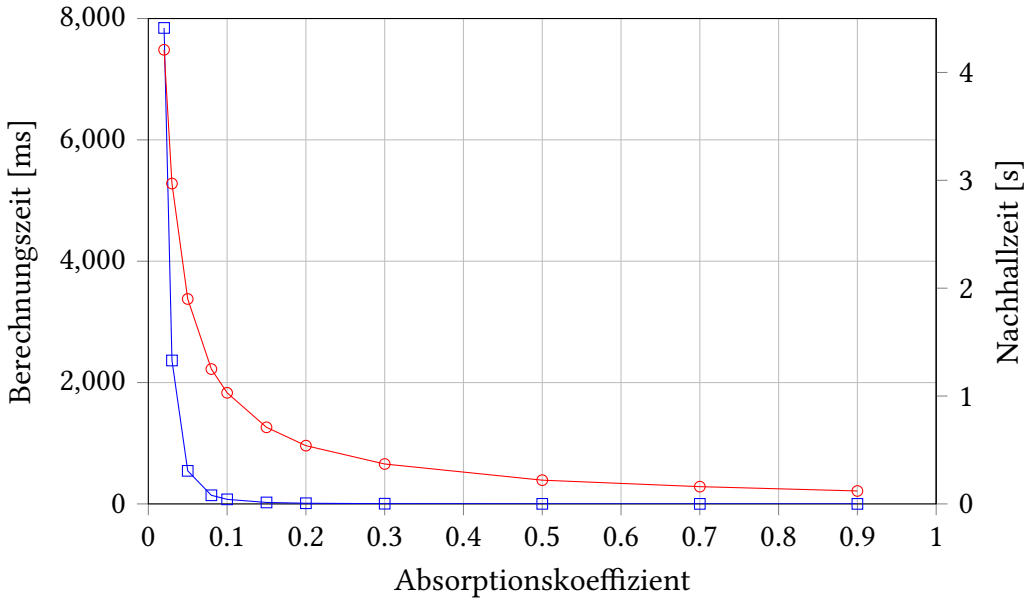

Abbildung 4.1: Nachhallzeit in rot und Berechnungszeit in blau

Dazu zeigt sich in Abbildung 4.2 ein linearer Zusammenhang zwischen der Raumgröße und der Nachhallzeit. Allerdings ist die Änderung der Berechnungszeit nur marginal. In der Betrachtung wurden verschiedene Simulationen mit unterschiedlichen Raumgrößen durchgeführt. Alle Dimensionen wurden um einen Faktor multipliziert und die Nachhallzeit und die Berechnungszeit wurden erfasst. Daraus lässt sich folgern, dass Materialien mit niedrigen Absorptionskoeffizienten einen deutlich höheren Einfluss auf die Berechnung haben als die Veränderung der Raumgröße.

Der Grund für diese Zusammenhänge ist, dass eine Vergrößerung des Raumes zu einer Verlängerung der Reflexionspfade führt. Somit dauert es länger bis Schallstrahlen am Empfänger

eintreffen, was zu einer Erhöhung der Nachhallzeit führt. Bei einem konstanten Absorptionskoeffizienten vergrößert sich die Anzahl der Berechnungen aber nicht signifikant. Dies ist anders bei der Veränderung der Koeffizienten. Hier nimmt die Anzahl der Berechnungen mit niedrigeren Koeffizienten stark zu. Da die Schallstrahlen mit jeder Reflexion nur wenig Energie verlieren, bleiben diese länger am „Leben“.

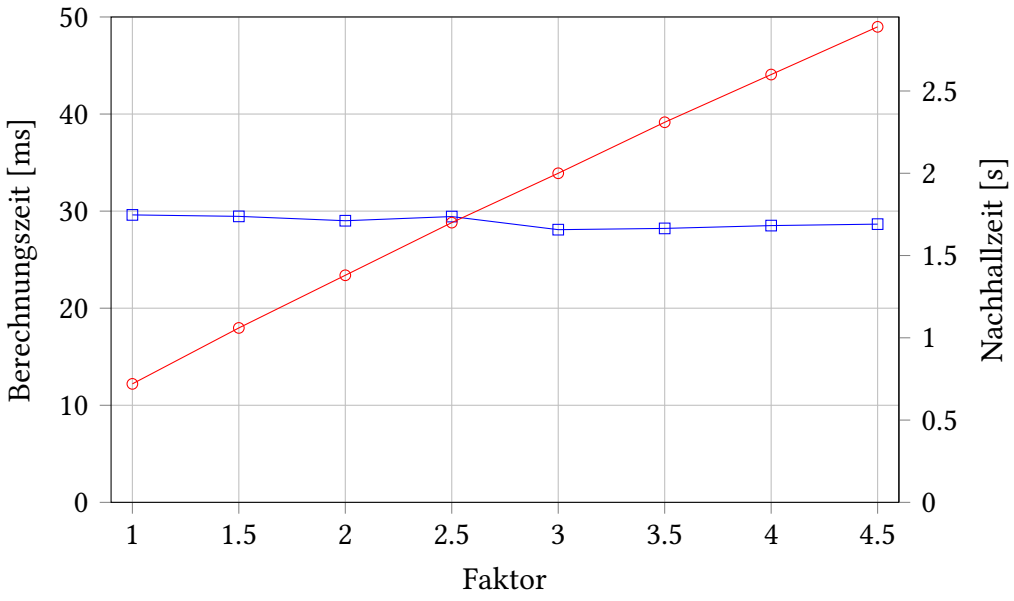

Abbildung 4.2: Nachhallzeit in rot und Berechnungszeit in blau

Um die Berechnungszeiten in einen Kontext zu geben, kann der Optimalfall betrachtet werden. Um eine unterbrechnungslose Berechnung zu erreichen, müsste eine komplette Impulsantwort zwischen zwei Samples berechnet werden. Bei einem Beispiel mit Abtastrate 44100 Hz würde dies einer maximal erlaubten Berechnungszeit von ca. 0,023 ms entsprechen. Wenn dies erreicht wird, könnte zum Beispiel eine kontinuierliche Bewegung im Raum simuliert werden ohne störende Artefakte im Audiosignal zu erzeugen. Mit einer niedrigeren Abtastrate verlänger sich dementsprechend auch die erlaubte Zeit. Diese äußerst niedrigen Werte werden allerdings in den meisten Fällen nicht erreicht (siehe Tabelle 4.1).

4.2 Verifizierung mit Akustiksimulation in EASE

Im Folgenden werden die generierten Impulsantworten miteinander verglichen. Zuerst die einzelnen Implementationen. Daraufhin wird der Basisraum auch mit dem Akustiksimulationsprogramm EASE simuliert und anhand des Ergebnisses verifiziert. EASE bietet vielfältige Möglichkeiten, Räume zu modelieren und sie akustisch zu simulieren. Aus der Simulation

können viele akustische Eigenschaften der Räume abgeleitet werden. Die Impulsantworten werden jeweils mit verschiedenen Signalen gefaltet und einem Hörvergleich unterzogen.

4.2.1 Analytischer Vergleich der Impulsantworten

Zuerst wird ein Vergleich der 3 verschiedenen Implementationen gezogen, um zu verifizieren, dass diese zu gleichen Ergebnissen führen. Verglichen wird der Direktschall, die Anzahl der ermittelten Reflexionen sowie deren zeitliche Position. Als Simulation dient der Basisraum aus Abschnitt 4.1.

Um die Differenzen einschätzen zu können, wurden die Impulsantworten und ihre Differenzen geplottet. In Abbildung 4.3 zeigt sich die Differenz. In Blau ist die Impulsantwort erzeugt mit der Pythonimplementierung ohne Lookup-Tabelle und in Rot das negierte Ergebnis der Implementation mit Lookup-Tabelle zu sehen. In Gelb sind die Differenzen eingezeichnet. Der selbe Prozss wurde zwischen Python ohne Lookup-Tabelle und der C++ Implementation durchgeführt (siehe Abbildung 4.4).

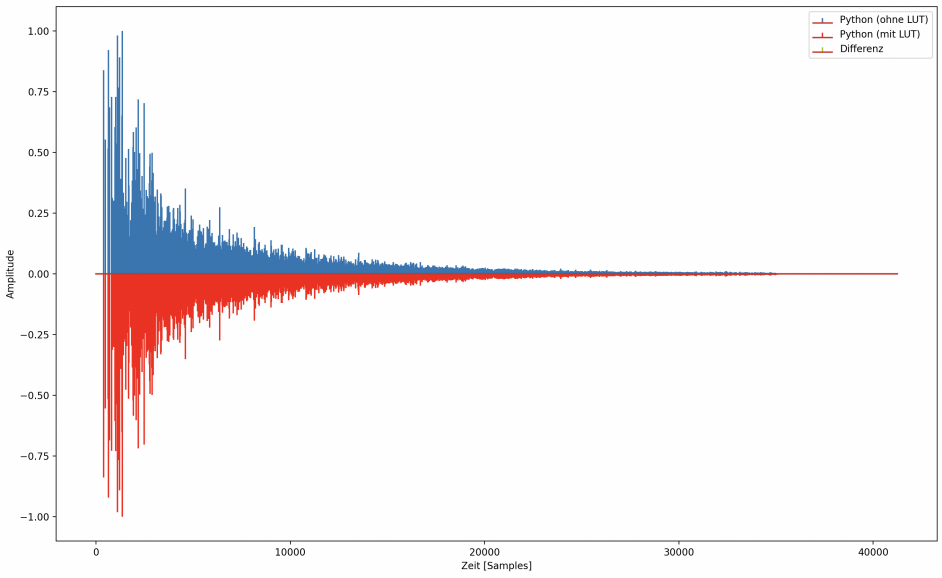

Abbildung 4.3: Differenz: Python mit und ohne Lookup-Tabelle

Zwischen den einzelnen Implementationen sind keine erheblichen Differenzen zu erkennen.

Im Folgenden wurde der Vergleich mit der Simulation in EASE vorgenommen. Dafür wurde der Raum im Programm möglichst nahe an der eigenen Simulation nachgebaut. Für die Betrachtung wurde erneut der Basisraum aus Abschnitt 4.1 verwendet.

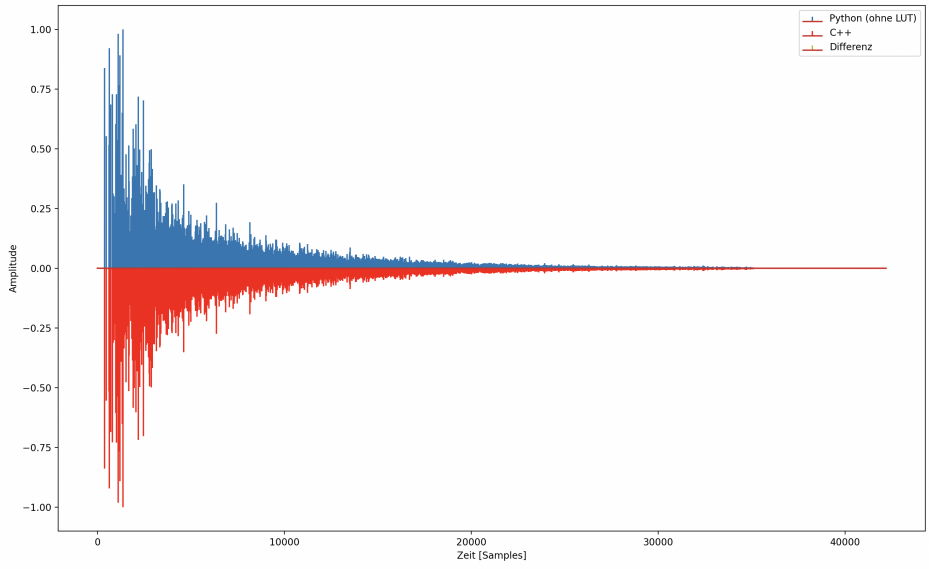

Abbildung 4.4: Differenz: Python mit Lookup-Tabelle und C++

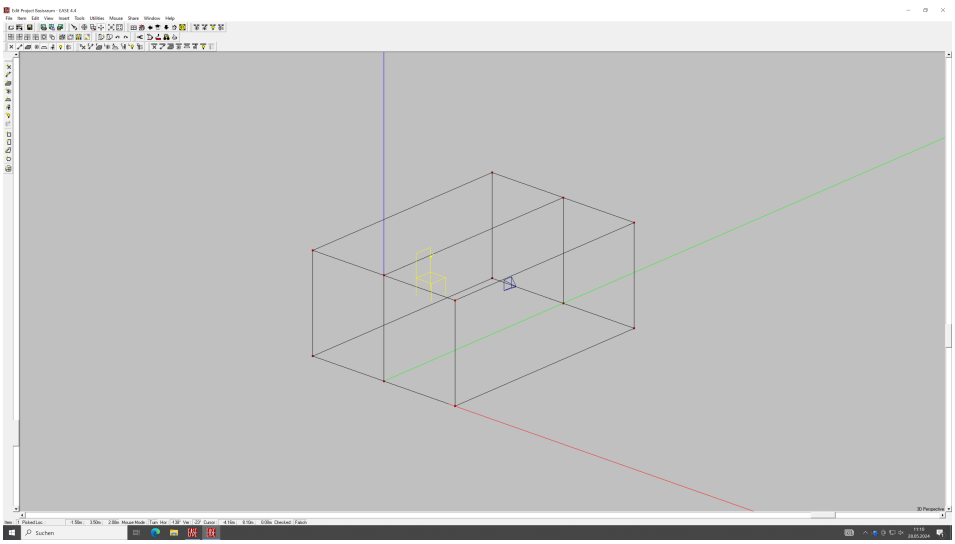

Abbildung 4.5: Der Basisraum im EASE Editor

Es wurden ähnliche Materialien verwendet. Im Anhang befindet sich eine Tabelle,in der die jeweiligen Absorptionskoeffizienten verglichen werden. Anschließend wurde die Simulation durchgeführt. Es wurde ein Reflektogramm erzeugt, welches die am Empfänger angekommenen Reflexionen aufzeichnet. In EASE gibt es hierfür verschiedene Optionen, um die Reflexionen zu bestimmen. Diese sind Raytracing und die Spiegelschallquellen-Methode. Die Simulation wurde einmal nur mithilfe der Spiegelschallquellen-Methode (in EASE Mirror Image Impacts) durchgeführt. Eine Simulation über Schallquellen 20. Ordnung hinaus konnte nicht berechnet werden, da das Programm dabei abstürzte. Dies entspricht nicht der vollen

Nachhallzeit, da aber besonders die frühen Reflexionen wichtig sind, wird dieses Ergebnis weiterhin betrachtet. Die Ergebnisse wurden aus EASE exportiert und mithilfe von Python analysiert.

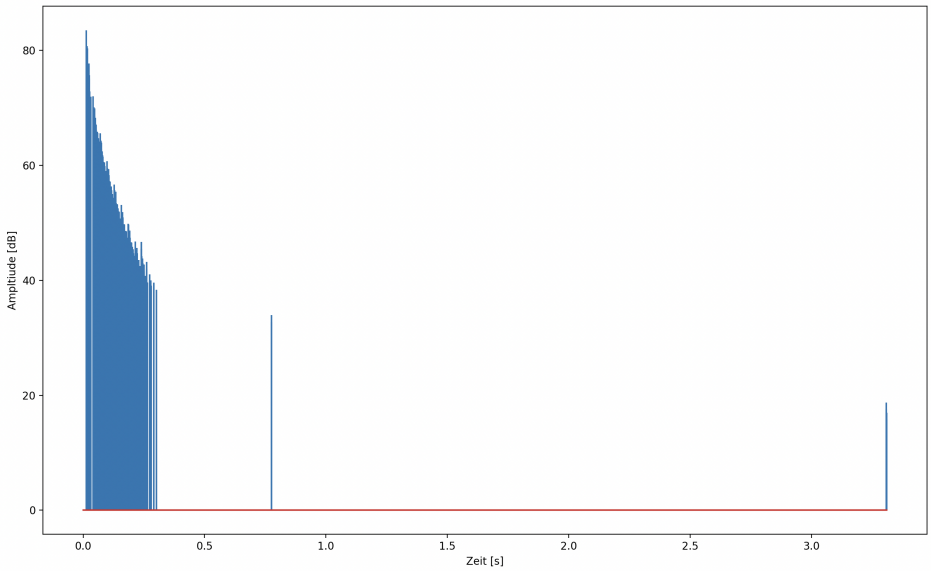

Abbildung 4.6: Reflektogram aus EASE

In Abbildung 4.6 fällt auf, dass ein paar Reflexionen besonders spät auftreten. Um die Simulationsergebnisse zu vergleichen, wurden der eigene Ansatz und die EASE Simulation normiert. Bei direktem Vergleich zeigen sich sehr große Differenzen.

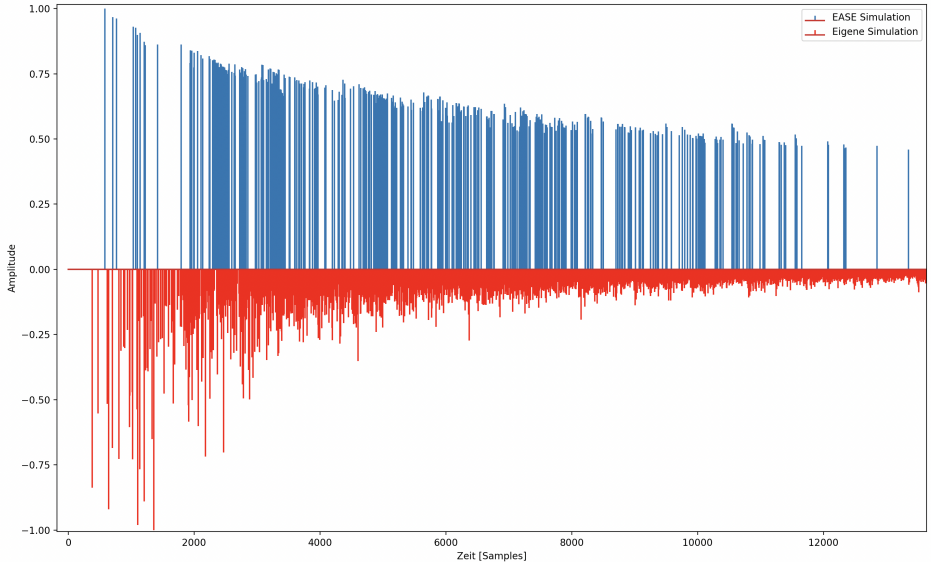

Abbildung 4.7: Vergleich zwischen EASE und eigener Simulation

Bei näherer Betrachtung der früheren Reflexionen zeigt sich schon bei Ermittlung des Direktschalls Unterschiede. EASE errechnet das erste Eintreffen nach 585 Samples. Die eigene Methode errechnet dies auf 385 Samples. Eine händische Berechnung des Direktschalls ergibt 386 Samples. Auch zwischen den nachfolgenden Reflexionen lässt sich keine klare Kohärenz entdecken. Dies gilt auch für den Energieabfall. Die EASE Simulation zeigt hier einen flacheren langsameren Abfall als die eigene Implementation. Die Simulation hat eine durchschnittliche Nachhallzeit von 0,75 s ergeben. Die eigene Implementation kommt auf 0,72 s.

Die Impulsantworten weichen stark voneinander ab. Warum diese Differenzen auftreten ist schwer zu erklären, da die Berechnungsmethoden von EASE nicht öffentlich zugänglich sind. Eine möglicher Ansatz ist vielleicht, dass das Programm mit Berechnungen von Spiegelschallquellen hoher Ordnungen nicht klar kommt, da die eigentliche Simulation normalerweise mit einem Hybrid aus Spiegelschallquellen und Raytracing erfolgt. Nur der ermittelte Nachhallzeitwert ist ähnlich. Dies ist der Fall, da EASE diese ebenso mithilfe der Sabine Formel berechnet.

4.2.2 Hörvergleich

Um die Impulsantworten weiter zu vergleichen wird ein Hörvergleich durchgeführt. Hierfür werden die Antworten jeweils mit zwei verschiedenen Audiosignalen gefaltet und danach abgehört. Das erste Signal ist ein Sprachsignal. Bei der Faltung mit der Antwort der eigenen Simulation ist ein räumlicher Eindruck mit Nahchhall zu hören. Auch die ermittelte Nachhallzeit von 0,72 s lässt sich hier hören. Die hohen Frequenzen sind betont und klingen im Vergleich zum Ausgangssignal schärfer. Die Faltung mit der Impulsantwort aus EASE ergibt einen anderen Höreindruck. Hier ist die räumliche Wirkung nicht so stark und durch die letzten Reflexionsanteile die auch in Abbildung 4.6 erkennen zu sind, klingt es als würde das Signal mehrmals verzögert abgespielt werden. Die Erkenntnisse bestätigen sich bei Verwendung eines anderen Testsignals mit mehr Transienten. Der schlechte räumliche Eindruck der EASE Simulation zeigt sich hier noch deutlicher.

Bei Verwendung des fertigen Plugins können die Effekte der Parameterveränderungen untersucht werden. Bei größeren Räumen ist auch die Nachhallzeit hörbar länger. Wenn Schallquelle und Empfänger nah beieinander zusammen sind, sind Audiosignale deutlicher zu verstehen. Je weiter diese von einander entfernt werden, desto leiser und unklarer wird das Audiosignal. Der diffuse Anteil nimmt also zu. Auch eine Veränderung der Wandmaterialien führt zu hörbaren Unterschieden. Materialien mit niedrigen Absorptionskoeffizienten bewirken klar höhere Nachhallzeiten. Allgemein fällt außerdem auf, dass das Signal deutlich leiser wird sobald das Plugin aktiviert wird. Beim gleichzeitigen Abhören des Signals und Verändern der Parameter ist ein klares Knistern zu vernehmen. Das liegt daran, dass mit jeder kleinsten

Veränderung eine komplett neue Impulsantwort errrechnet wird und nicht nur der Parameter der verändert wurde. Dies ist aufgrund der Software Architektur vorgegeben. Somit gibt es in den meisten Fällen eine kleine Verzögerung bis die neue Antwort erzeugt wurde und in die Convolution-Engine geladen wurde.

5 Zusammenfassung

Die Spiegelschallquellen-Methode bietet durch ihre deterministischen Eigenschaften eine gute Möglichkeit, die Impulsantwort eines Raumes zu simulieren. Dies trifft besonders auf den Direktschall und die frühen Reflexionen zu, da die zeitliche Auflösung der einzelnen Impulse sehr hoch ist. Stochastische Methoden wie Raytracing zeigen hier schlechtere Ergebnisse. Besonders für den geometrischen Spezialfall eines quaderförmigen Raums ist diese Methode eine gute Variante, um eine akurate Impulsantwort zu berechnen.

Ein Audio-Plugin hat hohe Anforderungen an die Geschwindigkeit der Codeausführung. Somit ist die Wahl einer möglichst effizienten und schnellen Programmiersprache wichtig. Mithilfe des JUCE-Frameworks lassen sich diese Plugins umsetzen. C++ gehört zu den effizienteren Sprachen, was im Vergleich mit den beiden Python Implementationen nachgewiesen wurde. Eine Automatisierung der Parameter im Echtzeiteinsatz, wie zum Beispiel die Umsetzung einer Bewegung des Empfängers im Raum, kann nicht störungsfrei simuliert werden, da die Berechnungszeiten zu lang sind. Der Optimalfall wird also mit dieser Methode noch nicht erreicht. Besonders bei Räumen mit niedrigem durchschnittlichen Absorptionskoeffizienten steigen die Berechnungszeiten enorm, sodass auf die Aktualisierung der Impulsantwort lange gewartet werden muss. Solche Bedingung müssten erreicht werden, um eine Akustiksimulation zum Beispiel in der virtuellen Realität umzusetzen. Inwiefern die Ergebnisse mit der Realität übereinstimmen konnte nicht ausreichend verifizert werden, da die Simulation stark von der aus dem professionellen Akustiksimulationsprogramm EASE abweicht. Allerdings wirken die Ergebnisse aus EASE nicht besonders akurat, was zum Beispiel an der Ermittlung des Zeitpunktes des Direktschalls zu sehen ist. Die Simulation wurde mehrfach wiederholt und überprüft. Eventuell wurden in EASE falsche Einstellungen vorgenommen, die zu diesen starken Abweichungen führten. Der räumliche Eindruck bei der Auralisation mithilfe des Plugins und mit Veränderung der Raumparameter klingt aber realistisch.

Um die Berechnungszeiten weiter zu verkürzen, gibt es noch verschiedene Möglichkeiten Optimierungen vorzunehmen. Zum einen könnte die Berechnung paralellisiert werden. Hierfür könnte man die Berechnung über die GPU und nicht die CPU laufen lassen, um die Berechnungen nicht seriell vorzunehmen. Eine weitere Möglichkeit wäre es, eine hybride Methode zur Ermittlung zu verwenden und nicht ausschließlich mit der Spiegelschallquellen-Methode zu arbeiten. Es könnte ein Hybrid aus Raytracing und Spiegelquellen verwendet werden. Eine weitere Optimierung wäre es, nicht mit jeder Veränderung eines Parameters

eine komplett neue Antwort zu erzeugen, sondern Möglichkeiten zu finden nur Teile der Berechnung neu auszuführen. Abschließend kann man sagen, dass das entwickelte Plugin auf jeden Fall dafür geeignet ist, einen Eindruck über die Akustik eines Raumes mit gewünschten Parametern zu erhalten. Bewegungen im Raum können aber aufgrund der Berechnungszeiten nicht realisiert werden.

Literatur

Allen, J., & Berkley, D. (1979). Image method for efficiently simulating small-room acoustics. *The Journal of the Acoustical Society of America*, *65*, 943–950. https://doi.org/10.1121/1.382599

Collins, M. (2003). *A professional Guide to Audio Plug-ins and Virtual Instruments*. Focal Press.

Fu, Z.-H., & Li, J.-w. (2015). GPU-based image method for room impulse response calculation. *Multimedia Tools and Applications*, *75*, 5205–5221. https://doi.org/10.1007/s11042-015-2943-4

Goudard, V., & Muller, R. (2003). *Real-time audio plugin architectures: a comparative study*. IRCAM.

JUCE. (n. d.). Verfügbar 1. April 2024 unter https://juce.com/

JUCE Documentation. (n. d.). Verfügbar 1. April 2024 unter https://docs.juce.com/master/index.html

Kinsler, L. E., Frey, A. R., Coppens, A. B., & Sanders, J. V. (2000). *Fundamentals of Acoustics* (4. Aufl.). Wiley; Sons, Inc.

Kiyohara, K., Furuya, K., & Kaneda, Y. (2002). Sweeping echoes perceived in a regularly shaped reverberation room. *The Journal of the Acoustical Society of America*, *111*(2), 925–930. https://doi.org/10.1121/1.1433808

McGovern, S. (2009). Fast image method for impulse response calculations of box-shaped rooms. *Applied Acoustics*, *70*, 182–189. https://doi.org/10.1016/j.apacoust.2008.02.003

Naylor, G. (1992). Treatment of early and late reflections in a hybrid computer model for room acoustics. *The Journal of the Acoustical Society of America*, *92*. https://doi.org/10.1121/1.404930

Schröder, D. (2011). *Physically based real-time auralization of interactive virtual environments*. Logos Verlag.

Sena, E., Antonello, N., Moonen, M., & Waterschoot, T. (2015). On the Modeling of Rectangular Geometries in Room Acoustic Simulations. *IEEE/ACM Transactions on Audio, Speech, and Language Processing*, *23*(4), 774–786. https://doi.org/10.1109/TASLP.2015.2405476

Sengpiel, E. (n. d.). *Absorption Coefficients α of Building Materials and Finishes*. Verfügbar 22. April 2024 unter https://sengpielaudio.com/calculator-RT60Coeff.htm

Steinberg. (n. d.). *Unsere Technologien*. Verfügbar 13. März 2024 unter https://www.steinberg.net/de/technology/

Thompson, S. (n. d.). *Random number generators for C++ performance tested*. Verfügbar 19. April 2024 unter https://thompsonsed.co.uk/random-number-generators-for-c-performance-tested

Vorländer, M. (2020). *Auralization: Fundamentals of Acoustics, Modelling, Simulation, Algorithms and Acoustic Virtual Reality*. Springer.

Weinzierl, S. (2008). *Handbuch der Audiotechnik*. Springer.

Anhang

Der Code und weitere Anhänge der Arbeit, wie einige Audio- und Analyse-Dateien, befinden sich auf dem beiliegenden USB-Stick.

Tabelle 1: Vergleich der Absorptionskoeffizienten

Material	**Freq (Hz)**	**EASE**	**nach Sengpiel, n. d.**
Plywood 3/8	125	0.28	0.28
	250	0.22	0.22
	500	0.17	0.17
	1000	0.09	0.09
	2000	0.10	0.10
	4000	0.11	0.11
Concrete	125	0.01	0.01
	250	0.01	0.01
	500	0.02	0.015
	1000	0.02	0.02
	2000	0.02	0.02
	4000	0.05	0.02
Carpet on concrete	125	0.04	0.02
	250	0.04	0.06
	500	0.15	0.14
	1000	0.30	0.37
	2000	0.50	0.60
	4000	0.60	0.65

Eigenständigkeitserklärung

Hiermit versichere ich, dass ich die vorliegende Bachelorarbeit mit dem Titel

Realisierung eines Audio-Plugins zur Simulation von Raumimpulsantworten

selbstständig und nur mit den angegebenen Hilfsmitteln verfasst habe. Alle Passagen, die ich wörtlich aus der Literatur oder aus anderen Quellen wie z. B. Internetseiten übernommen habe, habe ich deutlich als Zitat mit Angabe der Quelle kenntlich gemacht.

Hamburg, 16. Juni 2024