Hochschule Kempten
University of Applied Sciences

Bachelorarbeit

Studiengang Informatik - Game Engineering (Bachelor)

Realistische Berechnung von Sound im Spiel mittels Ray Tracing

Yannic Kugler

Aufgabensteller/Prüfer	Dr. S. Kern
Arbeit vorgelegt am	27. September 2021
durchgeführt in der	Fakultät Informatik

Kurzzusammenfassung

Die meisten modernen Tripple A Spiele setzen auf eine hochwertige, annähernd fotorealistische Grafik. Allerdings ist die Grafik nicht nur der einzige Aspekt eines Spieles, den es gilt realistisch darzustellen. Spiele, welche ihren Fokus eher auf den Aspekt der Simulation richten, versuchen häufiger nicht nur eine überzeugende Grafik, sondern z.B. auch ein realistisches Physiksystem implementiert zu haben. Zu diesen Simulationen zählt unter anderen Microsoft Flight Simulator [Fli]. Diese Software weist nicht nur eine beeindruckend realistische Grafik, sondern auch realistische Physik- und Wettersimulationen vor. Sound auf der anderen Seite wird kaum realistisch implementiert.

Abgeleitet davon befasst sich die vorliegende Arbeit mit dem Thema, wie sich Sound realistischer im Spiel darstellen lässt. Um dies zu beleuchten, wird zunächst auf die Ausbreitung von Sound in der Realität eingegangen. Diese Arbeit versucht dabei mithilfe von Ray Tracing Methodiken zu finden und zu formulieren, mit welchen sich Aspekte realistischer Schallausbreitung auf virtuelle Welten umsetzen lassen.

Die Ergebnisse der Arbeit stützen sich darauf, dass Ray Tracing als Prinzip eine potente Lösung für die Berechnung und Darstellung von Sound ist. Um konkrete Algorithmen und Verfahren zur Gewinnung von Daten zu finden, wurde sich stückweise mit den trennbaren Eigenschaften von Sound befasst. Dabei wird jedes Mal die physikalische Grundlage, Mathematik und Umsetzung im Detail aufgezeigt und zusammengeführt.

Nachdem konkrete Algorithmen erarbeitet und zusammengeführt wurden, werden verschiedene Implementationen angefertigt und gegeneinander verglichen. Um zu erarbeiten, ob diese alternative Berechnung von Sound Parametern eine realisierbare Herangehensweise darstellt, wird im Vergleich auch auf die Leistung des Codes der Arbeit eingegangen.

Inhaltsverzeichnis

1 Einleitung

In der Welt der Computergrafik spielen Rendering[1] Verfahren auf Basis von Ray Tracing[2] schon lange eine essenzielle Rolle. Ray Tracing als Technik wurde erstmals 1968 von Arthur Appel [App68] formuliert. Da der Algorithmus im Grunde den nicht relativistischen Aspekt der Ausbreitung von Licht nachbildet, wird dieser bereits seit Jahrzehnten in der Film bzw. allgemeinen Computergrafik Branche genutzt. Allerdings stellt sich die Frage, ob es möglich ist, die Ausbreitung von Schall mit einer Weiterentwicklung performant zu simulieren. Mir zur Verfügung stehende Artikel auf diesem Forschungsgebiet sind im Durchschnitt ziemlich veraltet. Der in diesen Beiträgen verwendete Algorithmus baut meist ebenfalls auf Ray Tracing auf und nennt sich Beam Tracing.

Da ich bereits seit Jahren selbst großes Interesse an Musik, Physik und Computergrafik habe, hat mich dieses Thema sehr angetan. Es bildet die Schnittmenge aller drei an sich großen Bereiche und hat bis jetzt eher wenig Aufmerksamkeit auf sich gezogen. Auch durch die mangelnde Forschung in den letzten Jahren an der Thematik habe ich mich dazu entschlossen sie zu beleuchten. Noch dazu kommt, dass ich in großen Spielen immer wieder feststellen musste, dass die Soundeffekte nicht immer zu der aktuellen Szenerie passen. Beispielsweise wird mit der aktuell verwendeten Technik häufig nicht berechnet, ob der Spieler das Geräusch überhaupt hören kann. Somit kann es zu Situationen kommen, in denen ein Spieler eine Geräuschquelle hinter einer Wand normal hören kann. Auch das Echo und viel mehr der Hall sind in Szenen meist rein statisch. Dies bedeutet, dass große Ereignisse in der Szenerie in der Regel kaum Einfluss auf die Geräuschkulisse nehmen. Dies hängt mit der aktuell verwendeten Technik zusammen. Diese erlaubt zwar eine dynamische Parametrisierung der Einstellungen der Effekte, jedoch werden die Parameter nicht aufgrund einer realistischen Simulation berechnet und gesetzt.

Alle in dieser Arbeit verwendeten Grafiken wurden mit Quellen versehen, insofern sie nicht selbst erstellt wurden. Für die Erstellung der Software wurden teilweise vorgefertigte Pakete verwendet. Diese sind in den Credits der Software aufgeführt und einsehbar.

1.1 Forschungsfragen

Da auf Ray Tracing aufbauende Techniken bereits für die Berechnung der Eigenschaften eines Raumes verwendet werden, stellt sich die Frage, ob es möglich ist, solch ein Verfahren im Kontext eines Spieles zu nutzen. Interaktive Software belastet Computer auf eine ganz bestimmte und schwere Weise. Aktuelle Spiele von großen Publishern[3] (Tripple A Games) beanspruchen annähernd alle Ressourcen eines modernen Computers. Daraus abgeleitet

[1] Rendering oder Bild Synthese ist der Prozess der Erzeugung eines Bildes aus einem 2D- oder 3D-Modell mithilfe eines Programms. [vgl. DA19]

[2] "Diese [...] Technologie simuliert Lichtstrahlen in einer 3D-Umgebung. Da Lichtstrahlen vorhersehbare physikalische Eigenschaften haben, versucht der Ray Tracing Algorithmus, die genaue Färbung jedes Schnittpunkts zwischen Strahl und Objekt zu berechnen." [Sch97]

[3] Ein Video Spiel Verlag ist ein Unternehmen, das Spiele veröffentlicht, welche entweder intern vom Verlag oder extern von einem Entwickler produziert worden sind.

muss der Algorithmus zur Simulation der Ausbreitung von Sound möglichst performant sein.

Die im Mittelpunkt stehende Frage dieser Arbeit lässt sich wie folgt formulieren: Ist es möglich, einen auf Ray Tracing aufbauenden Algorithmus zu entwickeln, welcher eine ausreichend realistische Ausbreitung von Sound simuliert und mindestens genauso gute Ergebnisse erzielt wie die konventionelle Methode?

Eine ebenso valide Frage ist zu erforschen, ob es mit diesem Ansatz möglich ist, eine vom Ergebnis ähnliche Lösung zu finden, welche allerdings wesentlich einfacher zu nutzen ist. Der in dieser Arbeit verwendete Algorithmus versucht einfach replizierbar zu sein. Darunter fällt die benötigte Parametrisierung der Simulation so gering wie möglich, allerdings so groß wie nötig zu halten.

Da sich die Ausbreitung von Sound allerdings in viele einzelne Probleme aufspalten lässt, (Lautstärke, Echo, Dopplereffekt, usw.) kann die im Fokus stehende Frage für jeden Teilaspekt gestellt werden. Sollte sich ergeben, dass sich die neue Herangehensweise äußerst gut für einige, allerdings nicht alle Aspekte von Schall eignet, so stellt sich die Frage, ob sich ein hybrider Ansatz formulieren lässt.

Mit den letzten Jahren haben GPU[4] SoC[5] Designer (AMD & Nvidia: Stand August 2021) neue Grafik Architekturen vorgestellt und veröffentlicht, welche bestimmte Berechnungsschritte des Ray Tracings in Hardware ausführen können. Das bedeutet, ein Teil des Chips ist mit speziellen Rechenwerken ausgestattet, welche es ermöglichen diese Berechnungen wesentlich schneller zu absolvieren als eine CPU[6] oder eine normale GPU. Auch die Frage, ob es möglich ist, die soeben genannte Ray Tracing Hardware in modernen Grafikkarten für die Simulation der Ausbreitung von Sound zu nutzen, wird gestellt.

1.2 Ziel der Arbeit

Das allgemeine Ziel der Arbeit ist, eine möglichst performante Lösung zur Simulation von Sound Ausbreitung zu finden. Auch sollen die errechneten Daten der Simulation leicht auf die benötigten Parameter der Effekte umgesetzt werden können. Dabei muss diese nicht in allen Teilen die aktuell verwendete Technik übertreffen. Die gefundene Methode soll im Idealfall leichter zu nutzen und zu implementieren sein als der zurzeit verwendete Ansatz. Da Beam Tracing auf Ray Tracing aufbaut und bereits in der Architektur als Algorithmus zur Berechnung der Ausbreitung von Sound genutzt wird, sehen die Ergebnisse für diese Thematik vielversprechend aus. [vgl. FTC+04] Leider eignet sich Beam Tracing nicht ganz als Technik für diese Arbeit. Die Gründe hierfür werden in 2.3 erläutert. Da Ray Tracing eine breite Basis für die Entwicklung eines spezialisierten Algorithmus bildet, lässt sich die konkrete Umsetzung auf viele verschiedene Arten realisieren. Aus diesem Grund werden, wie in den Forschungsfragen bereits angesprochen, multiple Versionen des Algorithmus implementiert. Durch diesen Ansatz lassen sich Vor- und Nachteile und folgend daraus Best Practices für die Umsetzung des Algorithmus formulieren. Auch die Leistung der verschiedenen Implementationen werden hierzu beitragen. Ein Aspekt, welcher in der Arbeit nicht behandelt

[4] Graphics Processing Unit (GPU). Auch Grafikkarte genannt.
[5] System on a Chip (SoC). Ein SoC (ausgesprochen "S-O-C") ist eine integrierte Schaltung, die alle erforderlichen Schaltungen und Komponenten eines elektronischen Systems auf einem einzigen Chip enthält
[6] Central Processing Unit (CPU). Auch Prozessor genannt.

wird, ist, ob die vorgestellte bzw. erarbeitete Methode perfekt für Lösung dieser Problematik geeignet ist. Auch wird nicht die direkte Manipulation von Sound Samples im Vordergrund stehen. Darunter fallen tatsächliche Interferenzen multipler Geräuschquellen aufgrund von Unterschieden der Amplituden und Phasen an einem gegebenen Ort zu einer gegebenen Zeit. Auch wird der Algorithmus eine vorhandene Audio Engine[7] nutzen und lediglich die Parametrisierung automatisieren. Das Entwickeln einer eigenen Audio Engine liegt nicht innerhalb des Rahmens einer Bachelorarbeit. Allerdings wird trotzdem auf Möglichkeiten und Potenziale eingegangen, welche leider nicht im Umfang dieser Arbeit im Detail erforscht werden konnten.

1.3 Struktur der Arbeit

Am Anfang dieser Arbeit gilt es zu verstehen, wie sich Schall in der Realität tatsächlich verhält und ausbreitet. Da Bücher, welche dieses Thema in der Tiefe aufgreifen, teils Magnituden mehr Umfang haben als eine Bachelorarbeit, wird dieser Teil der Arbeit nur knapp die wichtigsten Aspekte der Ausbreitung von Schall aufgreifen. Im weiteren Verlauf der Arbeit werden noch benötigte Effekte ebenfalls aufgegriffen und erklärt. Das vertiefte Wissen, welches in dieser Arbeit teils eingeflossen ist, wird aus dem Buch [Gar20] stammen.

Nachdem die Grundlagen gelegt wurden, kann die aktuell verwendete Technik erklärt werden. Mit dem Wissen über den Stand der Industrie lassen sich Stärken und Schwächen der verwendeten Technik extrahieren. Um zu ermitteln, ob ein anderer Ansatz zur Berechnung von Sound gut funktioniert, muss die neue Herangehensweise mindestens die gleichen Stärken vorzeigen. Es wäre kein Fortschritt, wenn eine alternative Methode in einigen Bereichen besser, allerdings im Durchschnitt schlechter ist als der bereits vorhandene Stand der Technik.

Da das verwendete Verfahren auf einem grundsätzlichen Prinzip aus der Computergrafik aufbaut, wird zunächst die ursprüngliche Methode erläutert. Diese Grundlage ist nicht nur nötig für das in dieser Arbeit erarbeitete Verfahren, sondern auch für bereits veröffentlichte Artikel auf diesem Themengebiet. Der Grund für die Beliebtheit von Ray Tracing in der Forschung bezüglich dieses Themas ist, dass Schall ein Wellenphänomen ist. Dadurch eignen sich diese Methoden äußerst gut, um die Ausbreitung von Sound zu modellieren. Da zu dem in dieser Arbeit vorgestellten Thema bereits ein ähnlicher Artikel existiert, wird auf diesen in Kapitel 2.3.1 genauer eingegangen.

Anschließend wird noch der aktuelle Stand der Industrie erläutert, um mögliche Schwachpunkte zu finden. Nachdem auch diese Grundlage gelegt wurde, lassen sich anhand der gefundenen Schwächen mögliche Fokus-Punkte setzen. Auf diese Punkte wird sich im Verlauf im Detail konzentriert.

Anschließend wird darauf eingegangen, wie sich dieses Prinzip nutzen lässt, um einen Sound Ray Tracer zu bauen. Das Herleiten des Algorithmus ist dabei von einer leicht experimentellen Natur. Es werden benötigte Parameter festgelegt und anhand dieser Variablen wird versucht eine Möglichkeit zu finden, diese Parameter zu berechnen. Das gesamte Problem Sound Ray Tracing wird dabei in seine einzelnen Teilprobleme zerlegt.

[7] Eine Audio Engine ist eine Software, welche alle nötigen Teile integriert hat, um Audio zu erstellen, manipulieren, mixen, mastern und auszugeben.

1. Dabei wird zunächst das einfachste Problem gelöst. Die Lautstärke abhängig von der Distanz des Hörers. Aufbauend auf den gewonnenen Erkenntnissen, wird der Algorithmus im Grundsatz so erweitert, dass er sich im Verlauf deutlich von der in der Computergrafik verwendeten Variante unterscheidet.

2. Darauffolgend, wird die wahrgenommene Änderung der Position der Quelle bei Reflexion und Diffraktion[8] behandelt. Dieser Punkt ist einer der größten Vorteile von Sound Ray Tracing. Die benötigten Daten für dieses Problem müssen nicht extra implementiert werden, sie sind direkt vom Algorithmus zur Verfügung gestellt.

3. Ein Wert, für welchen es in der Realität keinen wirklichen Namen gibt, ist der "Spatial Blend" Parameter. Diese Variable arbeitet Hand in Hand mit der virtuellen Position für eine akkurate Darstellung von Spatial Audio. Dieser Parameter bestimmt, wie stark der Hörer identifizieren kann, aus welcher Richtung der Sound kommt. Dieser Wert, in Verbindung mit der wahrgenommenen Richtung, wäre bei einem Audiosystem, welches lediglich Mono wiedergeben kann, überflüssig. Da allerdings annähernd jedes Wiedergabegerät (Kopfhörer, Hi-Fi-Anlage, usw.) mindestens 2 Kanäle zur räumlichen Trennung besitzt, ist es ein nicht zu vernachlässigender Parameter.

4. Mit den bis jetzt gewonnen Daten, lässt sich nun auch der Dopplereffekt berechnen. Bei diesem Punkt werden ebenfalls keine zusätzlichen Daten vom Ray Tracing benötigt.

5. Ein mit der Lautstärke verwandtes Problem ist die Manipulation der Frequenzbänder bei einer Reflexion, oder Diffraktion. Die Implementation davon ermöglicht es mächtige, allerdings nicht zwangsläufig physikalisch korrekte Effekte zu erzeugen. Effekte, welche mit dieser Herangehensweise einfach zu erstellen wären, sind beispielsweise der Bass im Außenbereich einer Diskothek, oder eine sich dumpf anhörende Konversation durch eine geschlossene Tür hindurch.

6. Mit dem Abschluss der Bestimmung der benötigten Parameter für die direkte Darstellung der Quelle werden nun die Effekte in der Audiopipeline beleuchtet. Den Vortritt hat dabei das Echo. Das Echo spielt bei der Berechnung eine besondere Rolle. Es ist ein Effekt, welcher sehr dynamisch und äußerst verschiedene Parameter annehmen kann. Diese lassen sich zum großen Teil ebenfalls von den bereits zur Verfügung gestellten Daten berechnen.

7. Am Ende der Probleme, welcher auf der CPU gelöst werden können, steht der Hall. Dieser ist ein Effekt, welcher am meisten zusätzliche Daten zur Berechnung seiner Parameter benötigt. Um den Hall in einer gegebenen Region zu bestimmen, muss jedes relevante Objekt im Umfeld der Geräuschquelle in Betracht gezogen werden. Interessant ist dabei auch, dass sich die Parameter für Hall bei statischen Quellen nicht ändern müssen. Nur bei sich bewegenden Quellen oder einer nicht statischen Szenerie müssen die Parameter neu berechnet werden. Auch hat die Position des Spielers wenig bis keinen Einfluss auf die Werte des Halls.

[8] Diffraktion beschreibt den Prozess der Beugung von Schallwellen um Objekte.

8. Ein Aspekt der Ausbreitung, welcher mit der CPU nur sehr schwierig mit der verwendeten Implementation umgesetzt werden kann, ist die Transmission von Schall. Der Prozessor kann zwar sehr vielseitige Berechnungen vornehmen, allerdings lassen sich auf einer CPU kaum genug Ray Berechnungen tätigen, um gute Ergebnisse für Transmission zu erzielen. Da in dieser Arbeit allerdings auch noch eine wesentlich leistungsstärkere Variante des Algorithmus auf der GPU implementiert wird, lässt sich dieser Aspekt dennoch umsetzen.

Das Projekt wird mit jedem vorher genannten Punkt um gesagtes Problem im Code erweitert. Die erste Methode zur Berechnung der Rays welche implementiert wird ist die Variante, die lediglich auf der CPU mit dem Physiksystem agiert. Hierfür gibt es einen simplen Grund: Das Debugging[9] von Berechnungen, welche nur auf der CPU getätigt werden, ist wesentlich einfacher und angenehmer, was ein schnelleres Vorankommen an der Arbeit garantiert. Erst nachdem alle benötigten Daten und deren Ursprung definiert wurden, kann der Algorithmus auf der GPU mit Hardware beschleunigtem Ray Tracing implementiert werden.

Nach der Erstellung beider Implementationen des Sound Ray Tracers werden diese anhand von mehreren Tests analysiert. Diese Tests prüfen dabei die Laufzeit, die Güte der berechneten Werte und die Nutzbarkeit. Ist die Evaluation abgeschlossen, so wird darauf eingegangen, wie dieser Algorithmus in der Zukunft verbessert und erweitert werden kann.

[9]Unter Debugging versteht man das Aufspüren und Beseitigen vorhandener und potenzieller Fehler (auch als ”Bugs” bezeichnet) in Softwarecode, welche zu unerwartetem Verhalten oder zum Absturz führen können.

2 Grundlagen

2.1 Ausbreitung von Schall

2.1.1 Erzeugung von Schall

Um einen Algorithmus erstellen zu können, welcher die Ausbreitung von Sound berechnen kann, benötigt man das Wissen, wie sich Sound in der Realität verhält. Dass Schall eine Welle ist, ist nicht fraglich, allerdings ist es eine andere Welle wie Licht. Schall ist eine sich gegebenenfalls periodisch ändernde Abweichung des Drucks vom Equilibrium des Mediums. Vereinfacht gesagt beschreibt die bekannte Darstellungsweise einer Tonspur die Position einer Membran (Abb. 2.1). Wird beispielsweise die Sinusfunktion als Signal gewählt, so ist die Position der Membran p abhängig vom Bogenmaß t. Der Wert t beschreibt hierbei den Fortschritt der Zeit mit einer gegebenen Frequenz. Anhand der Grafik (Abb. 2.1) lässt sich sehen, dass die Position der Membran nur vom Bogenmaß abhängt. Beim maximalen Ausschlag ist die Verdichtung des Mediums am höchsten und beim Minimum am geringsten. Dadurch erzeugen die Schwingungen der Membran entweder einen lokalen Über- oder Unterdruck. Diese lokalen Abweichungen vom normalen Druck breiten sich innerhalb des Mediums aus und können wiederum andere Objekte treffen und in Schwingung versetzen.

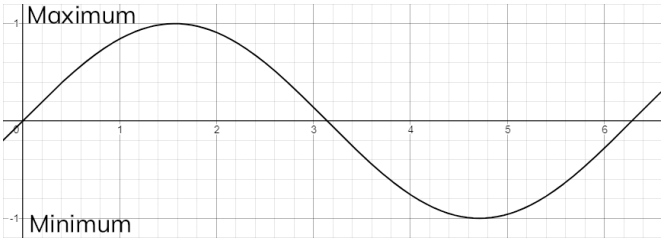


Abbildung 2.1: Funktion der Position der Membran anhand des Beispiels.
$p = \sin(t)$, x $\widehat{=}\, t$, y $\widehat{=}\, p$

2.1.2 Schallgeschwindigkeit in verschiedenen Medien

Die Geschwindigkeit der Schallausbreitung in einem Medium hängt in der Regel von multiplen Parametern ab. Die allgemeine Formel für die Berechnung der Geschwindigkeit in Flüssigkeiten und Gasen c ist folgende:

$$c = \sqrt{\frac{K}{\rho}} \qquad \text{[Hyp]} \tag{2.1}$$

ρ Dichte des Mediums
K Koeffizient der Steifigkeit (Bulk Modulus)

Anhand der Formel 2.1 lässt sich ablesen, dass ein größerer Bulk Modulus[1] eine schnellere Ausbreitung von Schall in einem Medium bedeutet.

[1] "Die elastischen Bulk Eigenschaften eines Materials bestimmen, wie stark es unter einem bestimmten äußeren Druck zusammengedrückt werden kann."[Nav] Je höher der Wert, desto steifer ist das Material.

Wird nur die Schallgeschwindigkeit für Luft benötigt, so lässt sich diese lediglich anhand der Temperatur (θ) berechnen. Dafür existieren eine genaue und eine approximierte Version. [vgl. Sena]

$$c_{akkurat} = 331.5 \cdot \sqrt{1 + \frac{\theta}{273.15}} \tag{2.2}$$

$$c_{approx} \approx 331.5 + 0.6 \cdot \theta \tag{2.3}$$

θ Luft Temperatur in °C

Um die Genauigkeit der Approximation für eine bestimmte Temperatur zu bestimmen, muss zunächst die Differenz der beiden Werte für die gleiche Temperatur gebildet werden.

$$c_{error} = c_{approx} - c_{akkurat} = 331.5 + 0.6 \cdot \theta - 331.5 \cdot \sqrt{1 + \frac{\theta}{273.15}}$$

Wird die Temperatur auf die Extrema, welche auf der Oberfläche der Erde gemessen wurden, limitiert, so ergibt sich als Minimum eine Temperatur von -89.2°C [vgl. And09, S.1] und als Maximum 56.7°C [vgl. Uni12]. Beim Einsetzen dieser Temperaturen für θ erhält man die Abweichungen der Approximation gegenüber der akkuraten Methode in Grad Celsius.

$$c_{error\ min} = 331.5 + 0.6 \cdot (-89.2) - 331.5 \cdot \sqrt{1 + \frac{-89.2}{273.15}} \approx 5.94\,^{\circ}\mathrm{C}$$

$$c_{error\ max} = 331.5 + 0.6 \cdot 56.7 - 331.5 \cdot \sqrt{1 + \frac{56.7}{273.15}} \approx 1.24\,^{\circ}\mathrm{C}$$

Nennenswert bei der maximalen Temperatur ist hierbei, dass der genutzte Wert der formale Rekord ist. Es existieren noch wesentlich höhere Werte, die jedoch nicht verifiziert wurden. Der höchste Wert in dieser Liste ist 93.9°C. Der Fehler der Approximation bei dieser Temperatur liegt bei 3.56 $\frac{\mathrm{m}}{\mathrm{s}}$. Das heißt, der Fehler der minimalen verifizierten Temperatur ist immer noch die größte Differenz der beiden Methoden. Mit dem maximal auftretenden Fehler von 5.94 $\frac{\mathrm{m}}{\mathrm{s}}$ lässt sich nun die maximale prozentuale Abweichung vom akkurat errechneten Wert bestimmen:

$$p_{error} = \frac{5.94}{331.5 \cdot \sqrt{1 + \frac{-89.2}{273.15}}} \cdot 100\% \approx 2.2\%$$

Eine maximale Abweichung von 2.2% von der akkuraten Geschwindigkeit ist klein genug, um es im Kontext des Klimas der Erde zu nutzen. Da vor allem die negativen Temperaturen schneller eine höhere Abweichung erreichen, kann die Temperatur zu einem Bereich von $[-50, 50]$ limitiert werden (Abb. 2.2), was einen maximalen Fehler von 0.62% ergibt. Wird diese Berechnung häufig benötigt, (viele Male jedes Bild) so lässt sich unter normalen Umständen die approximierte Version empfehlen. Wird allerdings die Geschwindigkeit selten neu berechnet, lässt sich ohne Nachteile der Leistungsfähigkeit die akkurate Version verwenden.

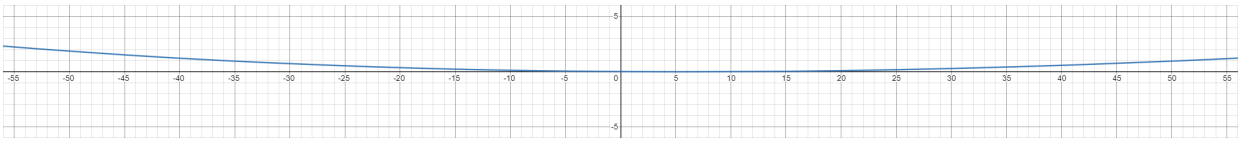


Abbildung 2.2: Funktion der Differenz der akkuraten und der approximierten Schallgeschwindigkeit $-55\,°\mathrm{C} < \theta < 55\,°\mathrm{C}$

2.1.3 Dämpfung von Schall in Luft

Da sich eine, wie in 2.1.1 beschriebene Welle im Raum ausbreitet, muss die initiale Welle mit der Zeit eine immer größere potenzielle Oberfläche bewegen können. Nimmt man eine kugelförmige Quelle mit Radius $r = 1\mathrm{m}$ an, welche eine Welle aussendet, also Energie in die Luft abgibt, so ist die bewegte Oberfläche $A_0 = 4\pi \cdot (1\mathrm{m})^2 = 4\pi\mathrm{m}^2$. Die Welle breitet sich jedoch aus und muss bei einem Radius $r = 4\mathrm{m}$ eine potenzielle Oberfläche von $A' = 4\pi \cdot (4\mathrm{m})^2 = 64\pi\mathrm{m}^2$ bewegen können. Die Energie der Welle hat sich jedoch weder erhöht, noch verringert.

$$E_{\text{ratio}} \propto \frac{A_0}{A'} = \frac{1}{16} = \frac{1^2}{4^2} = \left(\frac{1}{4}\right)^2 = \left(\frac{r_0}{r'}\right)^2 \tag{2.4}$$

Anhand von 2.4 wird ersichtlich, dass die Intensität einer Quelle proportional zu der quadratischen Distanz des Hörers abnimmt.

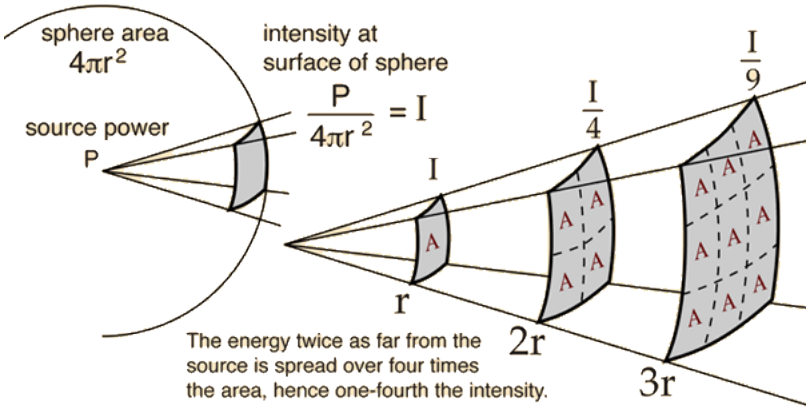


Abbildung 2.3: Eine Grafik zur Veranschaulichung von 2.4
`https://dobrian.github.io/cmp/topics/spatialization/isq.gif`

(Abb. 2.3) zeigt schon eine mögliche Formel für die Berechnung von "I". "I" ist dabei die Intensität der Schallwelle. Die Einheit für dieses Symbol ist $\frac{W}{m^2}$ und bezeichnet die Energie einer sich ausbreitenden Welle. Diese Energie hängt von der Distanz zur Quelle ab, und kann genutzt werden um einen Dezibel Wert, das Sound Intensity Level (SIL), zu berechnen. Für die Berechnung ist noch ein I_0 nötig. Dieser Wert beschreibt die minimale Energie, die ein menschliches Ohr wahrnehmen kann.

$$L_I = 10 \cdot \log_{10}\left(\frac{I}{I_0}\right)\mathrm{dB} \tag{2.5}$$

$I_0 = 10^{-12}\frac{W}{m^2}$

Eine sehr verwandte Einheit von SIL ist dabei das Sound Pressure Level (SPL). SPL hängt dabei nicht von der Intensität der Geräuschquelle, sondern vom Druck der Schallwelle ab. Die resultierenden Werte aus den Formeln beider sind dabei immer gleich. Für die Berechnung des SPL Wertes ist, genau wie beim SIL, ein Referenzwert nötig. Dieser hat nun die Einheit für Druck, also Pascal, und ist der minimale Unterschied von Druck, den ein menschliches Ohr auflösen kann. [vgl. Senb]

$$L_p = 20 \cdot \log_{10}\left(\frac{p}{p_0}\right) \text{dB} \tag{2.6}$$

$$10 \cdot \log_{10}\left(\frac{I}{I_0}\right) \text{dB}_{\text{SIL}} = 20 \cdot \log_{10}\left(\frac{p}{p_0}\right) \text{dB}_{\text{SPL}} \tag{2.7}$$

$p_0 = 2 \cdot 10^{-5}\,\text{P}$

Ein wichtiger Unterschied von Schallintensität und Schalldruck ist, dass die Intensität, wie oben erklärt, quadratisch zur Distanz abnimmt: $I \propto \frac{1}{r^2}$. Allerdings ist dies nicht der Fall für den Druck. Der Druck nimmt nur mit der Distanz ab: $p \propto \frac{1}{r}$! Da in dieser Arbeit häufig ein SIL oder SPL Wert berechnet werden muss, wird dieser Fakt ausgenutzt. Eine Wurzel Operation, was für das Lösen von $\frac{1}{r^2}$ nötig ist, ist eine teure Aufgabe für Hardware. Aus diesem Grund wird so viel wie möglich mit dem Druck von Schall gerechnet, um diese Operation bestmöglich zu umgehen.

Ein wichtiger Wert, welcher vom Algorithmus genutzt wird, um die Ausbreitung der Simulation zu optimieren ist die maximale Distanz, in welcher der Sound gehört werden kann. Um dies zu berechnen, wird noch eine Formel benötigt, welche mit möglichst wenig Aufwand aus einer gegebenen Lautstärke eine Distanz errechnet.

$$p = p_0 \cdot 10^{\frac{L_p}{20}} \tag{2.8}$$

$$r_2 = r_1 \cdot \frac{p_1}{p_2} \tag{2.9}$$

Setzt man nun p von 2.8 in p_1 von 2.9 ein, so ergibt sich folgende Formel:

$$r_2 = r_1 \cdot \frac{p_0 \cdot 10^{\frac{L_p}{20}}}{p_2} \tag{2.10}$$

Da r_2 die Distanz sein soll, bei der das Geräusch nicht mehr wahrnehmbar ist, so muss $p_2 = p_0$ gelten. Dadurch kann nun in 2.10 p_2 durch p_0 ersetzt und gekürzt werden. Da r_1 in dieser Arbeit als Referenz Lautstärke immer bei $1m$ sein wird, lässt sich r_1 ebenfalls aus der Gleichung streichen:

$$r_2 = 1 \cdot \frac{p_0 \cdot 10^{\frac{L_p}{20}}}{p_0} \;\Leftrightarrow\; r_2 = 10^{\frac{L_p}{20}} \tag{2.11}$$

Mit dieser Formel lässt sich zwar die maximale Distanz berechnen, jedoch wird noch eine weitere Formel benötigt. Sie muss anhand einer gegebenen Lautstärke (SPL) und einer Di-

stanz zur Quelle (in Meter) eine tatsächlich hörbare Lautstärke als Resultat ergeben. [vgl. Senc]

$$p_2 = p_1 \cdot \frac{r_1}{r_2} \tag{2.12}$$

p in 2.8 lässt sich nun in p_1 in 2.12 einsetzen.

$$p_2 = p_0 \cdot 10^{\frac{L_p}{20}} \cdot \frac{r_1}{r_2} \tag{2.13}$$

$$L_p = 20 \cdot \log_{10}\left(\frac{p}{p_0}\right) \tag{2.14}$$

Da p_2 von 2.13 den Druck der Welle mit einer gegebenen SPL Zahl, zu einem gegebenen Abstand darstellt, lässt sich p_2 nun in 2.14 einsetzen, um einen neuen SPL Wert zu berechnen.

$$\begin{aligned} L_{\text{dist}} &= 20 \cdot \log_{10}\left(\frac{p_0 \cdot 10^{\frac{L_p}{20}} \cdot \frac{r_1}{r_2}}{p_0}\right) \\ \Leftrightarrow L_{\text{dist}} &= 20 \cdot \log_{10}\left(\frac{10^{\frac{L_p}{20}}}{r_2}\right) \\ \Leftrightarrow L_{\text{dist}} &= 20 \cdot \log_{10}\left(10^{\frac{L_p}{20}}\right) - 20 \cdot \log_{10}(r_2) \\ \Leftrightarrow L_{\text{dist}} &= L_p \cdot \log_{10}(10) - 20 \cdot \log_{10}(r_2) \\ \Leftrightarrow L_{\text{dist}} &= L_p - 20 \cdot \log_{10}(r_2) \end{aligned} \tag{2.15}$$

r_2 = Distanz zur Quelle
L_p = SPL der Quelle bei einem Meter Abstand
L_{dist} = SPL bei gegebener Distanz r_2

Für Systeme, bei denen der Logarithmus von 2 wesentlich schneller berechnet werden kann als der Logarithmus der Basis 10, lässt sich das Ergebnis von 2.15 auch sehr genau mit folgender Formel approximieren:

$$L_{\text{dist}} = L_p - 6.0206 \cdot \log_2(r_2) \tag{2.16}$$

2.2 Ray Tracing

Im Grunde ist Ray Tracing ein Algorithmus, welcher die Ausbreitung eines oder mehrerer Strahlen in einer Szene modelliert. Um mit dieser Technik erfolgreich ein Bild zu generieren, werden mindestens diese drei Objekte benötigt:

1. Eine Lichtquelle: Von diesem Punkt werden die ursprünglichen Strahlen mit der Farbe der Quelle ausgesendet.

2. Ein Objekt: An der Oberfläche des Objekts werden Strahlen reflektiert und mischen die Farbe der Lichtquelle mit der Farbe der getroffenen Oberfläche.

3. Eine Kamera: Auch Auge genannt; Wenn Strahlen auf die "Netzhaut" treffen, werden diese Punkte anhand der Farbe des Strahls gefärbt.

Forward Ray Tracing (Abb. 2.4a) ist der Ansatz, welcher die Realität am genauesten beschreibt. Allerdings hat diese Herangehensweise einen entscheidenden Nachteil. Bei einer Szene mit vielen Objekten muss der Strahl gegebenenfalls viele Male reflektiert werden um zur Kamera zu gelangen. Dies kann dazu führen, dass zu wenig Strahlen die Kamera treffen und dadurch das Rendering[2] eine geringe Qualität aufweist. Ein sehr ähnlicher Ansatz ist das Aussenden der Rays von der Kamera aus. Diese Methode, Backward Ray Tracing (Abb. 2.4b) genannt, garantiert, dass jeder Pixel der Sichtebene immer "getroffen" wird. Dadurch wird garantiert, dass jeder Pixel des Bildes die gleiche Qualität vorweist. Allerdings hat diese Methode auch Nachteile, wie z.B. das Darstellen von Kaustiken von Linsen und Wasser. [vgl. Scr]

(a) Forward Ray Tracing (b) Backward Ray Tracing

Abbildung 2.4: Schematische Darstellungen beider Ray Tracing Algorithmen: Lichtstrahl (gelb), Transparenzstrahl (braun) und Kamerastrahl (grün)

Relevante Punkte für den Algorithmus

Der in dieser Arbeit erarbeitete Algorithmus wird auf Forward Ray Tracing aufbauen, da dies die Möglichkeit gibt, jede Quelle in Relation zum Spieler einzeln zu berechnen. Der Algorithmus soll im Idealfall möglichst wenig Leistung benötigen und Ray Tracing ermöglicht es die Arbeit auf multiple Zeitpunkte (Bilder) aufzuteilen. So lassen sich beispielsweise Quellen, welche weit weg und kaum hörbar sind, weniger oft berechnen, was es ermöglicht viel nähere und dominantere Quellen genauer zu betrachten. Auch der Hall muss von der Quelle aus berechnet werden. Noch dazu kommt, dass sich mit diesem Ansatz eine mögliche Verbesserung des Algorithmus vornehmen lässt. Wenn sich zwei Quellen direkt "sehen" können, dann lassen sich die Daten der einen bereits berechneten Quelle mit der anderen kombinieren. Abgeleitet davon könnte man für weit entfernte Quellen eine Hierarchiestufe einführen, welche die kombinierte Lautstärke aller Quellen berechnet und bestimmt.

[2]Ein Rendering bezeichnet das berechnete Bild des Algorithmus.

2.3 Stand der Forschung

Das Themengebiet realistischer Berechnung von Schall existiert schon lange. Aus diesem Grund ist die Berechnung der Ausbreitung von Sound schon seit langem möglich, leider nur offline[3]. In den letzten Jahren gab es auch einige Forschungsarbeiten im Themengebiet des online[4] Renderings von Schall. Dabei wurde meist nur die CPU für die Berechnungen verwendet. Um diese Berechnungen in Echtzeit durchführen zu können, wurden viele Schritte gegangen, um so viele Daten wie möglich im Vorhinein zu berechnen. Dieser Prozess wird in der Industrie als Baking[5] bezeichnet. Allerdings versucht die Industrie sich vom Baking Prozess zu distanzieren. Dies ist mit der benötigten Arbeit zu begründen, diese Berechnungen auszuführen und zu verwalten. Mit Unreal Engine 5 möchte Epic beispielsweise einige Baking Prozesse obsolet machen. [vgl. Unr] Mit dieser Entwicklung lässt sich erkennen, dass der gewählte Ansatz der Forscher eher weniger den modernen Standards entspricht.

2.3.1 Beam Tracing

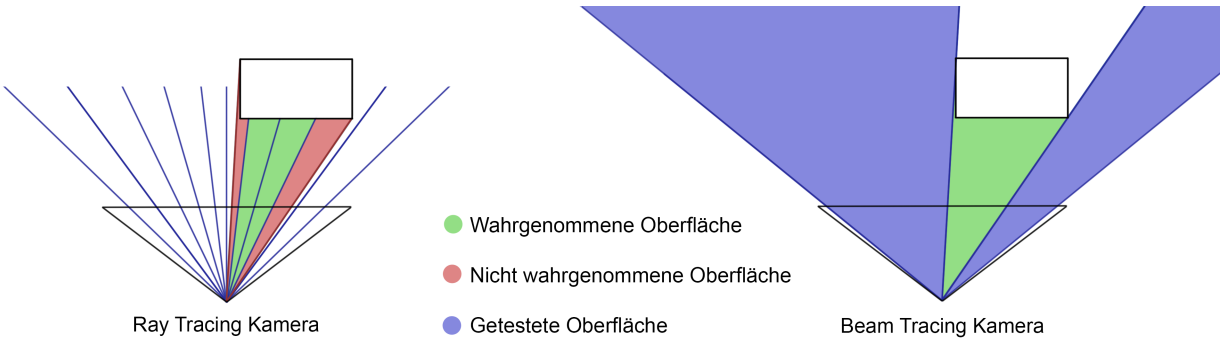


Abbildung 2.5: Vergleich von Ray und Beam Tracing. Links Ray Tracing mit dargestellten Rays, welche von der Kamera ausgesendet wurden. Rechts Beam Tracing, wobei eine Pyramide von der Kamera ausgesendet wird und bei einem Treffer in kleinere Pyramiden aufgeteilt wurde.

Beam Tracing ist ein auf Ray Tracing aufbauender Algorithmus, welcher häufig im Zusammenhang mit der Berechnung von Schall steht. Allerdings kann Beam Tracing auch in der Computergrafik verwendet werden. Dort hat sich dieser Algorithmus nur nicht durchgesetzt, weil die Berechnungen wesentlich aufwändiger sind als bei anderen Verfahren. Bei Beam Tracing werden keine einzelnen Rays pro Pixel ausgesendet, sondern eine pyramidale Struktur, welche das gesamte Frustum[6] abdeckt. Dies erlaubt einen Beam Tracer ein gesamtes Volumen an Raum zu evaluieren, anstelle von einzelnen Punkten. Anhand von (Abb. 2.5) wird veranschaulicht, wie Beam Tracing die gesamte Oberfläche erfassen kann. Ray Tracing kann ohne genug Samples[7] pro Pixel nicht die genauen Umrisse eines Objekts scharf

[3]Offline-Rendering bezeichnet allgemein die Berechnung eines Mediums (Bild, Animation, Musikstück), welches nicht in Echtzeit wiedergegeben werden kann, z.B. Animationsfilme.
[4]Online-Rendering bezeichnet allgemein die Berechnung eines Mediums (Bild, Animation, Musikstück), welches in Echtzeit wiedergegeben werden kann, z.B. Spiele.
[5]Baking beschreibt den Prozess, Teile eines Spieles im Vorhinein offline zu berechnen, sodass das laufende Spiel diese schnell benutzen kann und sie nicht in Echtzeit berechnen muss.
[6]In der Computergrafik ist ein Frustum der Bereich der modellierten Welt, der auf dem Bildschirm erscheinen kann.
[7]Ein Sample ist die Evaluation eines einzigen Rays eines einzigen Pixels.

erfassen. In Abbildung 2.5 wird zur Erläuterung nur ein Sample pro Pixel ausgesendet. Der rote Bereich ist dabei die Oberfläche, welche von keinem Sample wahrgenommen wurde und deshalb "nicht existiert". Bei Beam Tracing wird jedes Mal, wenn ein Dreieck einen Beam in irgendeiner Weise überdeckt, dieser ursprüngliche Beam aufgeteilt in "miss" Beams (blau) und "hit" Beams (grün). Von der getroffenen Fläche der hit Beams lassen sich wieder Beams aussenden. Dies ermöglicht es die Szene genau und vollständig zu analysieren. Allerdings ist die Berechnung der Beams und vor allem der Schnittpunkte wesentlich aufwändiger. Dazu kommt noch, dass Beam Tracing nur sehr schwer für Hardware Ray Tracing auf GPUs implementiert werden kann. Es ist möglich Beam Tracing auf einer GPU zu implementieren, nur kann dieser höchstwahrscheinlich nicht die spezielle Hardware innerhalb des SoCs nutzen. Aus diesen Gründen ist die Leistung von Ray Tracing, zumindest für Echtzeitanwendungen, Beam Tracing weit überlegen. Um Beam Tracing in Echtzeit zu ermöglichen müssen, wie schon erwähnt, viele Daten in der Entwicklung bereits gebaked werden. Aus diesem Grund kann Beam Tracing nicht mit dynamischen Geräuschquellen oder Geometrie zusammen arbeiten. [vgl. Lab, 5.2.3]

2.3.2 Weitere Ansätze

Ein weiteres interessantes Verfahren ist Phonon Tracing. Es ist auch eine verwandte Methode zu Ray Tracing, allerdings werden keine Schallwellen verfolgt, sondern Partikel. Es eignet sich gut um Reflexion und Transmission darzustellen, allerdings lässt sich mit dieser Methode nicht die Beugung von Schallwellen an Objekten berechnen. [vgl. Lab, 5.2.4]

Es gibt auch Methoden, welche ganz auf Ray Tracing als grundlegenden Baustein verzichten. Eine dieser Methoden errechnet anhand der Oberflächen in einer Szene virtuelle Positionen der Geräuschquelle. (Abb. 2.6) Ein großer Vorteil dieser Methode ist, dass sie vollständig geometrisch korrekt ist. Das Problem wird jedoch anhand der exponentiell wachsenden Anzahl an virtuellen Quellen sichtbar. Das bedeutet, dass nur eine relativ geringe Tiefe an Reflexionen erreicht werden kann. [vgl. Sve02, 2.1]

Als Alternative zur Simulation ist es ebenfalls möglich, die Ausbreitung von Schall mit vollständig numerischen Verfahren zu modellieren. Diese Verfahren sind äußerst akkurat und realistisch, jedoch ebenso komplex. Diese hohe Komplexität führt dazu, dass numerische Verfahren kaum für die Berechnung von Sound in Echtzeit geeignet sind. [vgl. Lab, 5.1] Oftmals werden solche Verfahren in der Industrie für die Berechnung der akustischen Eigenschaften von Räumen verwendet.

Relevante Punkte für den Algorithmus

Obwohl viele vorhandene Ansätze auf Ray Tracing aufbauen, heißt das nicht, dass sie alle auf die gleiche Weise funktionieren. Das wohl größte Problem, welches alle Ray Tracing Methoden gemein haben, ist das Beugen der Schallwellen um Objekte. Dieser Effekt ist äußerst schwierig akkurat zu berechnen. Allerdings wird im Algorithmus dieser Arbeit eine alternative Idee verwendet, welche zwar nicht sehr akkurat sein dürfte, jedoch den Rechenaufwand um einiges reduziert. Auch wird die Idee einer virtuellen Sound-Quelle und deren wahrgenommene Position betrachtet.

2.4 Stand der Industrie

Potenziell können in der Industrie viele verschiedene Ansätze verwendet werden. Beispielsweise besitzt jede kommerzielle Game Engine[8] ein eigenes System und viele Studios verwenden sogar ihre selbst entwickelten Lösungen. Aus diesem Grund wird sich im Rahmen dieser Arbeit auf eine in der Industrie weit verbreitete Lösung eines Drittanbieters fokussiert.

2.4.1 Wwise

Wwise ist laut Audiokinetics eigenen Aussagen die umfangreichste Audio Engine auf dem Markt. Sie ist erweiterbar mit Plugins, welche die potenziellen Features umso umfangreicher machen. Betrachtet man die Software gänzlich ohne Plugins, so lässt sich die grundlegende Funktionsweise wie folgt zusammenfassen: Wwise ist eine eigenständige Software und interagiert im Prinzip nicht direkt mit dem Code eines Spieles. Aus diesem Grund lässt sich die Software auch "von außen" an jedes Spiel "anstecken". Damit ist gemeint, dass Wwise auf Events des Spiels reagiert. Ein Event ist dabei vereinfacht gesagt eine Nachricht des Spiels, in der Wwise über Ereignisse im Spiel informiert wird. Wwise nimmt diese Nachrichten auf und reagiert dementsprechend auf das damit verbundene Ereignis. Beispielsweise könnte ein Event Wwise informieren, an welcher Position sich der Spieler befindet. Wwise reagiert auf diese Nachricht und setzt gegebenenfalls die korrekte Lautstärke und Paning[9] der betroffenen Quellen. Mit diesem Prinzip lassen sich alle denkbaren Parameter innerhalb von Wwise anpassen. Dieser Ansatz ist äußerst mächtig und würde sogar mit den Ergebnissen dieser Arbeit gut zusammen funktionieren. Doch hat es einen entscheidenden Nachteil. Wwise kann nur die Informationen erhalten, die von den Entwicklern bewusst preisgegeben werden. Aus diesem Grund ist es für Wwise ohne Erweiterungen oder besonderen Einsatz von Entwicklern nicht möglich physikalisch korrekten Sound wiederzugeben. Reagiert die Software z.B. wieder auf die Position des Spielers, so kann Wwise ohne das Wissen über die Geometrie der Szene nicht auf Objekte und Räume reagieren. Gegebenenfalls könnte der Hörer so Geräusche ohne Probleme durch Wände und somit aus der falschen Richtung bzw. Lautstärke hören.

2.4.2 Wwise Spatial Audio und Wwise Reflect Plugin

Um dieses Problem zu lösen hat Audiokinetic die Plugins Wwise Spatial Audio und Wwise Reflect veröffentlicht. Wwise Reflect ist über die Geometrie der Szene informiert. Dies wird genutzt, um frühe Reflexionen der Sound-Quelle zu berechnen. Diese Reflexionen können noch von sogenannten "Acoustic Textures" manipuliert werden. Diese Texturen beschreiben die verschiedenen Absorptionswerte für eine gegebene Oberfläche oder Materialtyp. [vgl. Audb]

Wwise Spatial Audio ist dabei ein weiteres Plugin, welches in Zusammenarbeit mit Wwise Reflect den Hall und die Position von Geräuschquellen innerhalb von Räumlichkeiten berechnet. Der Hall kann dabei so genau bestimmt werden, dass er sich so anhört, als würde er von einer bestimmten Richtung kommen. Auch das Konzept von "virtual Emitters" (Abb.

[8] Eine Game Engine ist eine Software, welche alle zur Spieleentwicklung nötigen Grundsysteme (Rendering, Audio, Physik, Objektverwaltung, usw.) zur Verfügung stellt. Spiele werden in der Regel auf der Basis einer Game Engine gebaut.

[9] Paning ist die Verteilung eines Audiosignals in ein neues Stereo- oder Mehrkanal-Klangfeld, das durch die Einstellung eines Pan-Reglers bestimmt wird.

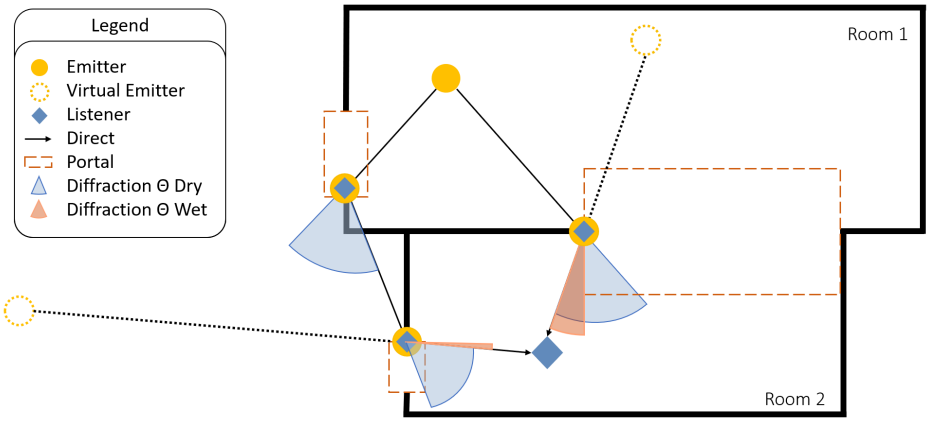


Abbildung 2.6: Grafik von der Audiokinetic Website zur Funktionsweise von Wwise Spatial Audio.
`https://www.audiokinetic.com/images/template/sound_propagation.png`

2.6) wird verwendet. Dabei wird berechnet, aus welcher Richtung man eine Quelle hören würde, wenn sie sich in einem anderen Raum befindet als der Hörer. [vlg. Audc] Anhand der Informationen von Audiokinetic zum Wwise Spatial Audio-Plugin lässt vermuten, dass das Plugin auf einem hybriden Ansatz von mindestens zwei der vier vorgestellten Verfahren aufgebaut wurde. (Kapitel 2.3)

Stärken und Schwächen von Wwise

Stärken:	Schwächen:
- Umfangreich - Industrie Standard - Ziemlich performant - Relativ einfach zu benutzen - Auf vielen Plattformen verfügbar - Einfach erweiterbar mit Plugins	- Kostenpflichtig - Keine Transmission von Sound - Keine bzw. beschränkte Diffraktion von Sound - Berechnungen sind nur auf CPU - Wwise Spatial Audio Räume müssen separat konfiguriert werden

Tabelle 2.1: Stärken und Schwächen von Wwise

Wwise ohne Plugins ist eine performante, recht einfach zu nutzende, umfangreiche Lösung für Spatial Audio in Spielen. Darüber hinaus ist die Software in der Industrie weit verbreitet. Mit dem Zusatz von Plugins wird Wwise zu einer annähernd perfekten Lösung.

Der wohl größte Nachteil, direkt hinter der Kostenpflichtigkeit, dürfte wohl der hohe Aufwand bei der Parametrisierung der einzelnen Plugins sein. Deutlich wird dies durch den Ansatz des Wwise Spatial Audio-Plugins, in welchem der Entwickler die Räume und Portals[10] manuell definieren muss. Dies bedeutet unter Umständen einiges mehr an Arbeit, um die Features von Wwise Spatial Audio nutzen zu können. Auch dass Wwise nur auf der

[10] Portals sind definierte Bereiche, in denen sich keine Wände befinden und sich Schall dadurch frei ausbreiten kann.

CPU agiert, könnte ein potenzielles Limit darstellen. Da Ray Casting[11] eine ziemlich teure Operation für einen CPU ist, könnte das Wwise Spatial Audio-Plugin potenziell von einem schwächeren Prozessor limitiert werden. [vlg. Audc]

2.4.3 Abgeleitete Anforderungen

Der Fakt, dass Wwise auf fast allen Gebieten eine gute Lösung bietet, macht die Zielsetzung für diese Arbeit schwierig. Es gibt lediglich fünf Punkte, welche verbessert werden könnten:

1. Komplette dynamische Berechnung von nicht statischen Schallquellen.
2. Dynamische Geometrie wird in Berechnung berücksichtigt.
3. Weniger Arbeit bei der Parametrisierung der Software.
4. Berechnung der Diffraktion und ggf. Transmission von Schall.
5. Auslagern von Berechnungen auf die GPU.

Die ersten beiden Punkte basieren überwiegend auf den gegebenen Einschränkungen, welche sich vom Beam Tracing Ansatz ableiten lassen. Der dritte Punkt lässt sich leider schlecht in dieser Arbeit gezielt evaluieren. Der Grund dafür ist, dass Wwise eine Softwarelösung für so ziemlich alle Bereiche in der Audioproduktion von Spielen bietet. Trotzdem lässt sich die Benutzung dieser Anwendung mit der Nutzbarkeit von Wwise Spatial Audio-Plugin vergleichen. Sollte die in dieser Arbeit erstellte Anwendung ohne die manuelle Definition von Räumen und Portalen auskommen, so ist dies auf jeden Fall nennenswert. Diese Arbeit wird lediglich den Spatial Audio Aspekt der Vertonung eines Spiels behandeln. Da ein Aspekt dieser Arbeit auf die Auslagerung von Berechnungen auf die GPU abzielt, können wesentlich komplexere bzw. umfangreichere Berechnungen durchgeführt werden als nur auf einem Prozessor. Dies bietet die Möglichkeit für die Berechnung der Transmission von Schall.

[11] Ray Casting ist die Berechnung aller Schnittpunkte einer gegebenen Halbgerade mit allen in der Szene vorhandenen Körpern.

3 Konzeption

3.1 Entwicklung des Algorithmus

3.1.1 Lautstärke der Quelle

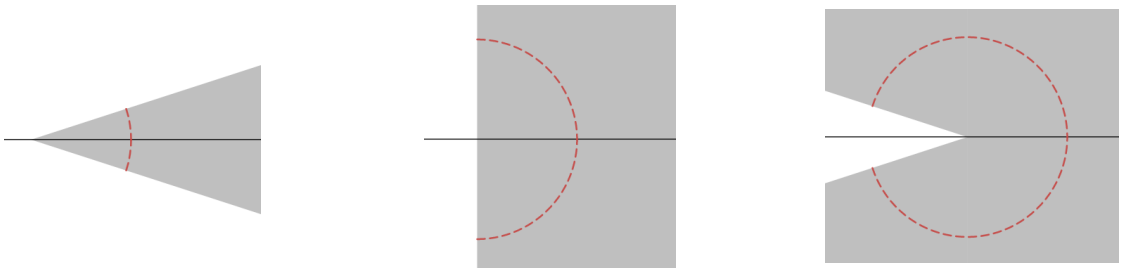
Die wahrgenommene Lautstärke einer Quelle lässt sich recht einfach definieren. Die Lautstärke ist im physikalischen Sinne nicht nur von der Distanz zur Quelle abhängig, sondern auch von den Interferenzen, welche sich auf diesem Weg gebildet haben. Da in dieser Arbeit allerdings nicht auf die Berechnung von Interferenzen eingegangen wird, wie in Kapitel 1.2 beschrieben, wird dieser Aspekt vernachlässigt. Aus diesem Grund besteht die Ausbreitung zunächst aus zwei zusammengesetzten Effekten (Reflexion & Diffraktion), welche im Folgenden einzeln aufgegriffen werden.

Referenzlautstärke und Isometrie

Um die Lautstärke einer Quelle anhand einer gegebenen Distanz berechnen zu können, benötigt die Quelle einen Parameter, der die Referenzlautstärke definiert. Dieser Parameter ist ein Dezibel Wert, welcher die Lautstärke bei einer Distanz von 1m definiert. Um die Lautstärke (hierbei ist die gesamte Lautstärke aller Frequenzbänder zusammen gemeint) anhand einer Distanz zu berechnen, wird die bereits hergeleitete Formel 2.15 verwendet:

$$L_{\text{dist}} = L_p - 20 \cdot \log_{10}(r_2)$$

Allerdings wird eine Geräuschquelle auch von einer Ausrichtung und einem Öffnungswinkel (Isometrie) definiert. Anhand von Abbildung 3.1 werden drei Werte für die Isometrie veranschaulicht. Da die Simulation im dreidimensionalen Raum ausgeführt wird, muss diese Idee angepasst werden, sodass die Isometrie einen Kegel definiert und nicht nur eine Fläche.



(a) Isometrie = 0.1 (b) Isometrie = 0.5 (c) Isometrie = 0.9

Abbildung 3.1: Darstellung verschiedener Werte für die Isometrie. Graue Fläche beschreibt Richtungen, in welche Strahlen ausgesendet werden. Die Ausrichtung ist nach rechts. Der rot gestrichelte Kreis ist dabei der Öffnungswinkel ($0 \leq t \leq \pi$ wobei gilt $t = \text{Isometrie} \cdot \pi$)

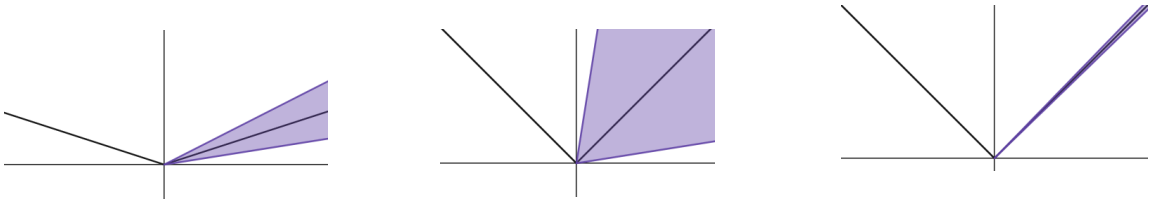
Da allerdings die Verwendung eines strikten Winkels eine Verwendung von trigonometrischen Funktionen verlangt, wird die Richtung eines Strahles mit einer optimierten, nicht auf

Winkeln basierten, Operation berechnet. Dabei wird ein zufälliger Vektor auf der Einheitskugel[1] $v_{\vec{\text{random}}}$ gewählt, mit der Ausrichtung $v_{\vec{\text{forward}}}$ der Quelle anhand der Isometrie i linear interpoliert und schließlich normalisiert.

$$\begin{aligned} \vec{v_d} &= v_{\vec{\text{random}}} \cdot i + v_{\vec{\text{forward}}} \cdot (1 - i) \\ v_{\vec{\text{dir}}} &= \frac{\vec{v_d}}{|\vec{v_d}|} \end{aligned} \tag{3.1}$$

Reflexion

Bei einem Ray Tracing Algorithmus spielt die Reflexion eine essenzielle Rolle in der Ausbreitung der Pfade im Raum. Dabei wird dem Nutzer die Möglichkeit gegeben, jede Oberfläche einzeln mit einem Diffusionswert anzupassen. Dieser Parameter definiert im Grunde wie stark die Oberfläche spiegelt. Ein Wert von 0 bedeutet, dass die Oberfläche perfekt spiegelt und somit eintreffende Rays nur perfekt reflektiert. Ein Wert von 1 bedeutet das genaue Gegenteil und somit eine rein diffuse Reflexion (Matt). Für eine vollständig diffuse Reflexion wird ein eintreffender Ray in eine absolut zufällige Richtung innerhalb der Hemisphäre[2] der Normale[3] weitergeleitet. In Abbildung 3.2 werden verschiedene Werte der Diffusion dargestellt. Die Hemisphäre der Normale ist dabei ein Halbkreis, welcher alle Richtungen mit einem positiven Y Wert einschließt.



(a) Diffusion = 0.1 (b) Diffusion = 0.4 (c) Diffusion = 0.01

Abbildung 3.2: Darstellung der möglichen Richtungen verschiedener Werte der Diffusion mit verschiedenen Einfallswinkeln. Die lila Fläche beschreibt Richtungen, in welche Strahlen ausgesendet werden können. Für die Diffusion gilt $0 \leq$ Diffusion ≤ 1.

Die tatsächliche Berechnung der Richtung des diffus reflektierten Strahls $v_{\vec{\text{diff}}}$ wird, wie schon bei der Isometrie, optimiert. Dabei wird ebenfalls auf die Nutzung von Trigonometrie verzichtet. Eine gute und performante Annäherung an das Verhalten lässt sich erzielen, indem zwischen der reflektierten Richtung $v_{\vec{\text{reflect}}}$ und einem Vektor auf der Hemisphäre $v_{\vec{\text{hemi}}}$ der Normale $\vec{n}$ linear interpoliert wird.

$$\begin{aligned} \vec{v_h} &= v_{\vec{\text{random}}} + \vec{n} \\ v_{\vec{\text{hemi}}} &= \frac{\vec{v_h}}{|\vec{v_h}|} \end{aligned} \tag{3.2}$$

[1] Ein Vektor auf der Einheitskugel beschreibt einen Vektor, der vom Ursprung zu einem Punkt auf der Oberfläche einer Kugel mit dem Radius 1 zeigt und damit normalisiert ist.

[2] Die Hemisphäre einer Normale ist im zweidimensionalen Sinn ein Halbkreis. Diese Hemisphäre ist anhand von Abbildung 3.2 der Halbkreis, dessen Y Werte nur positiv sind.

[3] Eine Normale beschreibt die Ausrichtung einer Oberfläche im dreidimensionalen Raum. Eine Normale wird in der Computergrafik anhand des normalisierten Kreuzprodukts von zwei Vektoren gebildet und beschreibt somit die Ausrichtung eines Dreiecks.

$$\begin{aligned} \vec{v_d} &= \vec{v_{\text{hemi}}} \cdot d + \vec{v_{\text{reflect}}} \cdot (1 - d) \\ \vec{v_{\text{diff}}} &= \frac{\vec{v_d}}{|\vec{v_d}|} \end{aligned} \tag{3.3}$$

Diffraktion

Da ein Bereich der hörbaren Wellenlängen in der gleichen Größenordnung wie das alltägliche Leben liegt, können sich Schallwellen mit diesen Wellenlängen um Objekte beugen. Dieser Prozess der Wellenbeugung wird als Diffraktion bezeichnet. Die Wellenlängen der hörbaren Frequenzen liegen dabei im Bereich von 17.15mm und 17.15m bei einer Temperatur von 20°C. Frequenzen, welche überhaupt von Objekten effektiv abgelenkt werden können, befinden sich bei ca. 500Hz - 400Hz und darunter. Allerdings werden dabei tiefere Frequenzen effektiver und damit stärker umgelenkt. Da auf die Änderung von verschiedenen Frequenzbändern im späteren Verlauf der Arbeit eingegangen wird, kann unter diesem Punkt nur die verwendete Methode für die angenäherte Berechnung der Diffraktion für die tiefsten Frequenzen erläutert werden.

Für die allgemeinen Berechnungen der Schallausbreitung bietet Huygens-Prinzip für jeden der drei Aspekte einen Ansatz. Ursprünglich wurde das Prinzip formuliert, um Diffraktion von Schall zu berechnen. Es beruht dabei auf der Idee, dass jeder Punkt einer sich voran bewegenden Wellenfront[4] ein potenzieller Startpunkt einer kugelförmigen Quelle ist. [vgl. Lab, 2.2.3 Diffraction]

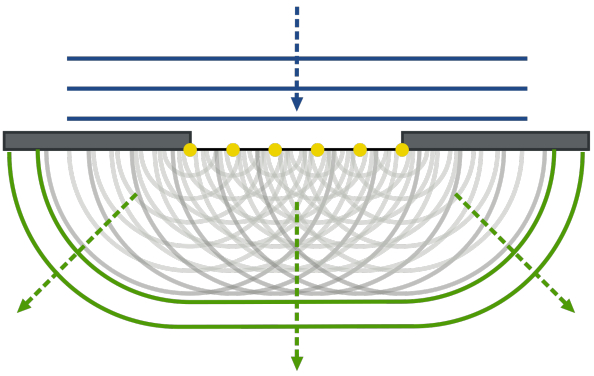


Abbildung 3.3: Huygens-Prinzip für die Berechnung von Diffraktion.
`https://en.wikipedia.org/wiki/Huygens%E2%80%93Fresnel_principle#/media/File:Refraction_on_an_aperture_-_Huygens-Fresnel_principle.svg`

Leider würde dieser von Abbildung 3.3 veranschaulichte Ansatz eine manuelle Definition von Portalen für Schall benötigen. Dies würde allerdings den Aufwand der Parametrisierung auf eine ähnliche Ebene wie die Lösung von Wwise Spatial Audio anheben. Darüber hinaus ist die Berechnung vieler virtueller Emitter äußerst rechenaufwändig. Auch im späteren Fall, wenn die Berechnungen auf der GPU ausgeführt werden, soll der Algorithmus möglichst performant ablaufen. Eine Berechnung von multiplen virtuellen Geräuschquellen in Verbindung mit einer großen Menge an potenziellen Stellen erhöhen den Rechenbedarf für die akkurate Darstellung von Diffraktion dementsprechend exponentiell. Aus diesem Grund wird auf die

[4]Die Wellenfront ist die erste, am weitesten entfernte, Schallwelle einer Quelle.

physikalisch korrekte Berechnung von Diffraktion verzichtet. Allerdings wird eine alternative Idee implementiert.

Grundsätzliche Funktion des Algorithmus

Eine Spielwelt beherbergt zu signifikanten Teilen große, zum Teil von Menschenhand, zum anderen Teil natürliche Strukturen. Diese Objekte sind mögliche Stellen für Diffraktion von Schall. Im grundsätzlichen Prinzip ergibt ein Ray beim Ray Tracing genau drei mögliche Resultate.

- Der Ray trifft ein Objekt und wird reflektiert bzw. gebrochen oder terminiert.
- Der Ray verfehlt jegliche Geometrie der Szene innerhalb seiner maximalen Distanz und terminiert.
- Der Ray trifft den Hörer und terminiert.

Strahlen, welche den Spieler getroffen haben, sind wertvoll für die Berechnung von Parametern. Insofern haben sie ihre Aufgabe erfüllt. Allerdings sind Rays, welche entweder nichts oder nur Geometrie getroffen haben, verschwendeter Rechenaufwand. Die im Algorithmus verwendete Idee beruht darauf, solche "nutzlosen" Pfade zur Berechnung einer angenäherten Form der Diffraktion zu verwenden. Um dies zu erreichen wird der Pfad einfach in eine zufällige Richtung weitergeführt, falls der Ray keine Geometrie trifft. Damit lässt sich der grobe Gesamtablauf mit folgendem Ablaufdiagramm (Abb. 3.4) darstellen.

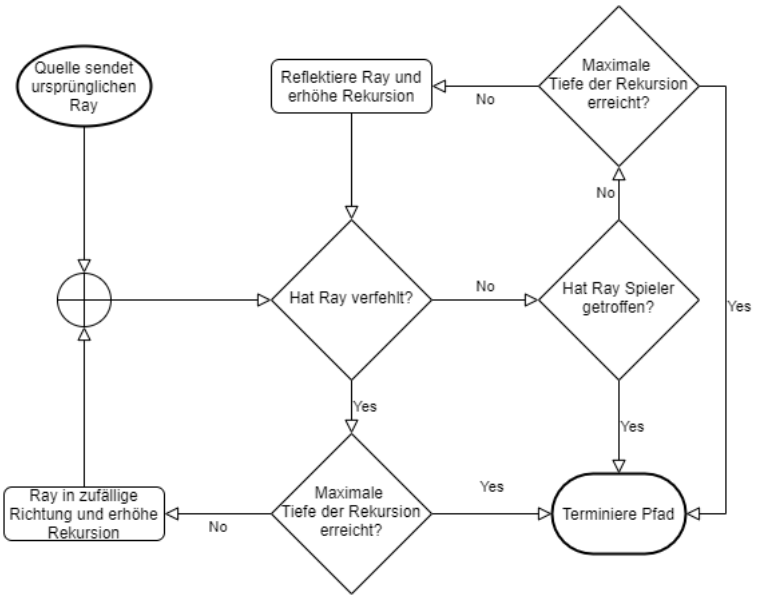


Abbildung 3.4: Vereinfachte Entscheidungsfindung für das Voranschreiten eines Pfades.

In der Praxis hilft dieser Ansatz die Immersion zu erhöhen. Mit dieser Methode werden wesentlich glaubwürdigere Änderungen der Lautstärke bei der Bewegung um eine Wand herum erzielt. Das liegt daran, dass der Schall im Falle des Hörers hinter der Wand entweder reflektiert oder gebeugt werden muss. Bewegt sich nun der Spieler aus dem Schatten der Wand, so wird die Geräuschquelle auf einmal wesentlich lauter. Würde man die Lautstärke in diesem Fall lediglich anhand der Distanz des Spielers zu Quelle berechnen, so würde die Intensität

nicht korrekt reduziert und es wirkt so, als wäre die Wand nicht vorhanden.

Zusammenfassend ist nun für jeden benötigten Schritt zur Berechnung der Lautstärke eine Methode definiert. Allerdings fehlt noch die konkrete Berechnung der Lautstärke. Um sinnvolle Daten von einem Ray Tracing Algorithmus zu erhalten, müssen mehr als nur ein einziger Pfad berechnet werden. Im Grunde ist ein absolut akkurates Ergebnis unmöglich, da ein Ray Tracing Algorithmus dafür alle möglichen Pfade in Existenz getestet haben müsste. In der Regel lässt sich jedoch davon ableiten, dass das Ergebnis besser wird, je mehr Samples berechnet wurden. Da der in dieser Arbeit erstellte Algorithmus zunächst auf einer CPU berechnet wird, ist die Anzahl der Samples sehr begrenzt. Dabei kann ein moderner CPU in wenigen Millisekunden viele tausend Ray Casts ausführen. Allerdings sind tausende Ray Casts verschwindend wenig im Vergleich zu der Anzahl, die bei der Berechnung von Bildern genutzt wird. Aus diesen Gründen existieren für die Verbesserung der Bildqualität sogenannte Denoiser. Diese sind Algorithmen, welche aus den bereits berechneten Pixel Informationen Artefakte herausfiltern. (Abb. 3.5)

(a) Verrauschtes Bild (b) Entrauschtes Bild

Abbildung 3.5: Vergleich von zwei identischen Bildern, welche beide mit 4 Samples pro Pixel gerendert wurden. Datei und Software zur Erstellung beider Bilder: `https://www.blender.org/download/demo-files/`

Da der in dieser Arbeit erstellte Algorithmus ebenfalls auf Ray Tracing aufbaut, sollte für eine stetige und nicht sprunghafte Änderung der Parameter auch eine Methode für Denoising formuliert werden. Um die Ausführung des Algorithmus in einer interaktiven Anwendung zu ermöglichen, muss diese in einem einzigen Bild stattfinden. Darüber hinaus müssen die davon erzeugten Daten gut genug sein, sodass sie eine plausible Geräuschkulisse bilden. Ray Tracing Algorithmen basieren dabei häufig auf leicht zufälligen Ausbreitungsrichtungen (Monte-Carlo Prinzip) der Strahlen, welche von Rahmenparametern abhängig sind. Das bedeutet, dass bei einer geringen Anzahl an Rays die Parameter gegebenenfalls pro Ausführung ziemlich stark fluktuieren können. Aus diesen Gründen basiert die verwendete Denoising Methode auf der Idee, die Daten mehrerer Ausführungen zu speichern und anschließend zu mitteln. Dabei hat sich experimentell eine Speicherung von 20 früheren Ausführungen als ein gutes Mittel zwischen Reaktivität und Stetigkeit herausgestellt. Ein kleiner Nachteil dieser Methode ist, dass die gemittelten Parameter per Definition nicht genau die aktuell berechneten Werte sein können. Dies führt dazu, dass zwar Fluktuationen reduziert, allerdings die gesamte Reaktionszeit erhöht wird.

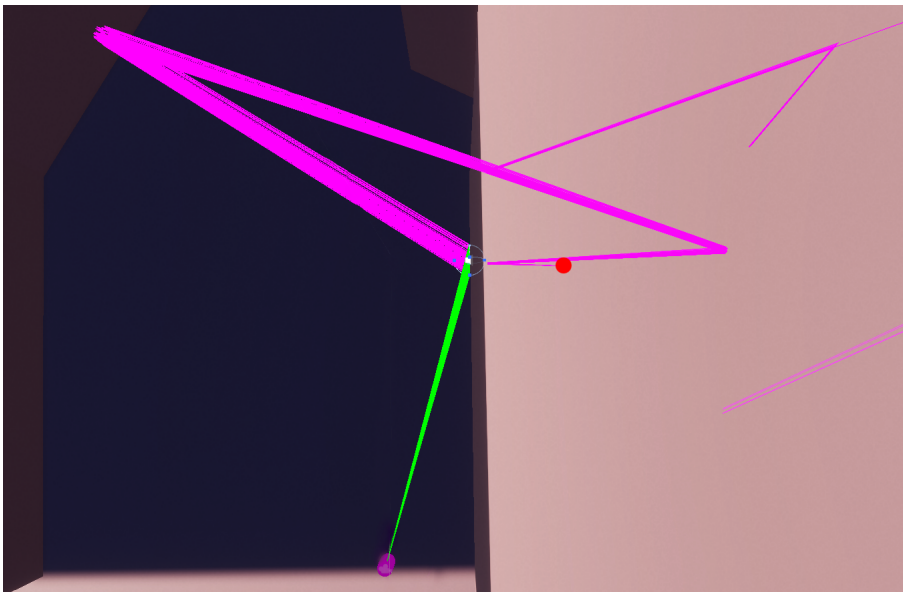


Abbildung 3.6: Die Visualisierung von einer Ausführung des Algorithmus. Die Isometrie der Quelle (roter Punkt) beträgt dabei in diesem Fall 0.01 und die Diffusion der Wände 0.001. Am Ende der grünen Pfade befindet sich der Hörer.

Um die Lautstärke einer Geräuschquelle an einem gegebenen Punkt im Raum zu bestimmen, muss ein Pfad von der Quelle zum Hörer gelangen. Da Wege möglicher Pfade von den Regeln der Reflexion und der Diffraktion beschränkt werden, ist die Chance, dass ein Pfad zufällig den Hörer trifft, ziemlich gering. Dies führt dazu, dass von ca. 200 Pfaden nur ein einziger den Spieler tatsächlich trifft. Um die Anzahl valider Pfade zu erhöhen, wird am Endpunkt jedes Rays ein weiterer zusätzlicher Ray in Richtung des Spielers ausgesendet, wenn die Richtung des Hörers innerhalb von 150% des eigentlich validen Diffusionskegels liegt.

Anhand der Reflexionen in Abbildung 3.6 lässt sich erkennen, dass Rays, welche den Spieler nicht getroffen haben, perfekt reflektiert wurden. Die grünen Strahlen haben den Spieler getroffen. Sie wurden allerdings nicht perfekt reflektiert. Dies ist eine Optimierung, sodass mehr Rays den Spieler treffen. Würde man diese Herangehensweise nicht nutzen, so würde in diesem Fall kein Pfad den Spieler treffen, da die rekursive Tiefe von fünf dafür nicht ausreicht. Wäre die rekursive Tiefe größer, so würden Strahlen in diesem Szenario mit ziemlicher Sicherheit den Spieler treffen. Aus diesem Grund ist es annehmbar diese Approximation vorzunehmen.

Die verwendete Formel, um zu testen, ob der Strahl in Richtung des Spielers innerhalb des vergrößerten Reflexionskegels liegt, ist die Folgende:

$$x_{\text{diff}} = (1 - \vec{v_{\text{reflect}}} \cdot \vec{v_{\text{player}}}) \cdot 1.5 \tag{3.4}$$

Gilt dabei $x_{\text{diff}} \geq d$, wobei d für die Diffusion der Wand steht, so ist die Richtung vom Punkt der Reflexion zum Hörer valide und wird als valider Pfad gespeichert. Fällt die Richtung des Spielers nicht in den definierten Kegel, so wird auf den Test, ob der Hörer getroffen werden kann, verzichtet und der Pfad normal weitergeführt. Nicht nur die grünen Pfade in Abbildung 3.6 haben versucht den Spieler zu treffen. In der rechten oberen Ecke haben ebenfalls Rays versucht den Spieler zu treffen, wurden allerdings von einer Wand aufgehalten.

Optimierung des Algorithmus

Anhand der bereits erarbeiteten Formel 2.10 lässt sich auf simple Weise die theoretisch maximale Ausbreitung einer Quelle errechnen. Diese Information ist äußerst nützlich, da sich mit diesem Wert der Bereich, der vom Algorithmus abgetastet werden muss, stark einschränken lässt. Sollte beispielsweise ein Pfad dieses Limit überschreiten, lässt sich dieser Pfad sofort abbrechen, da die Lautstärke im späteren Verlauf 0 ist. Auch lässt sich die Distanz der Quelle zum Spieler nutzen, um die Ray Längen zu optimieren. Die angenäherte Berechnung der Diffraktion wäre mit einem statischen Limit der maximalen Distanz nicht möglich. Da ein Strahl, wenn er alles verfehlt, bis zur maximalen Distanz voranschreitet. Dadurch wird dieser direkt terminiert und kann nicht testen, ob er den Spieler über Diffraktion treffen würde. Aus diesem Grund hat es sich als nützlich herausgestellt, die maximale Ray Distanz auf die quadrierte Distanz der Quelle zum Spieler zu setzen, insofern es ein Diffraktions- oder Reflexionsstrahl ist. Dieser Wert wurde gewählt, da er äußerst einfach zu errechnen ist. Die Verwendung dieser Optimierungen hilft dabei, die Laufzeit des Algorithmus zu verringern, sowie die allgemeine Qualität des Ergebnisses zu verbessern.

Berechnung der Lautstärke

Mit all den vorhandenen Daten und der festgelegten Methode für die Berechnung der Ausbreitung eines Pfades, lässt sich nun die Vorgehensweise in der Extraktion der Daten definieren. Anhand von Tabelle 3.1 lassen sich benötigte Daten eines Ray veranschaulichen. Dabei wird jedes Modul (Lautstärke, Position der Quelle, Echo, Dopplereffekt, usw.) gegebenenfalls weitere Daten hinzufügen, welche der Strahl mit sich führen muss. Bis jetzt sind lediglich die für die Ausbreitung und Lautstärke relevanten Daten aufgelistet.

Modul benötigt	Ray Information		
Grundsätzlich	Ursprung	Richtung	Ray Länge
Lautstärke	Pfadlänge	wurde gebeugt	Spieler getroffen

Tabelle 3.1: Benötigte Informationen eines Strahls (Ray) für die Berechnung der Lautstärke.

Für schnellen und organisierten Zugriff auf die Daten, werden alle Pfade, welche den Spieler getroffen haben, die Menge der validen Pfade bilden. Alle anderen Pfade, welche den Hörer verfehlt haben, bilden für die Visualisierung eine separate Menge. Um nun von den validen Pfaden die Information für die Lautstärke zu extrahieren, gibt es multiple Ansätze:

- Das Mitteln der Pfadlängen aller validen Pfade. Dies liefert ein zu leises Ergebnis. Die wahrgenommene Lautstärke hängt überwiegend von den kürzesten Pfaden ab.
- Das Mitteln der Pfadlängen, welche kürzer als der Durchschnitt aller Pfadlängen sind. Dies liefert ziemlich stabile Werte für die Lautstärke, hat allerdings das gleiche Problem wie die vorige Lösung.
- Das Selektieren des kürzesten Pfades. Dieser Ansatz liefert in der Praxis ziemlich gute Ergebnisse, verursacht aber instabile Lautstärken bei einer geringen Anzahl an validen Samples.
- Das Definieren einer festen Grenze, ab welcher diese Anzahl der validen Pfade evaluiert wird. Beispielsweise für einen Wert von fünf: Es müssen in einer Ausführung mindestens fünf Pfade zum Spieler finden, sodass aus den Samples überhaupt Daten

errechnet werden. Dies minimiert zwar Undersampling[5], erhöht jedoch bei Grenzfällen (wenn genau die richtige Anzahl an Pfaden, also 3 - 6 den Spieler treffen) wieder die Instabilität.

Bei einer Quelle mit genug Samples ($\geq$ 1000 Rays) bietet die simple Verwendung des kürzesten Pfades ein ziemlich gutes Ergebnis. Der größte Nachteil dieser Methode sind die auftretenden Instabilitäten durch Undersampling. Diese können in extremen Fällen bis zu $\pm$ 20dB erreichen. Aus diesem Grund wird die zuletzt erklärte Methode genutzt. Da die darin beschriebene Grenze in Echtzeit veränderlich ist, ermöglicht dies die Erkennung von Grenzfällen. Mit einer solchen Erkennung ließe sich die Grenze der Situation entsprechend manipulieren.

Da die Charakteristiken des eintreffenden Schalls einer Quelle überwiegend von den Reflexionen abhängen, müssen diese und gebeugte Pfade separat behandelt werden. Gebeugte (und später auch transmittierte) Pfade werden aus diesem Grund jeweils einen separaten Audiomixer nutzen, da diese teilweise deutlich zum Klang der Quelle beitragen können. Dieser Ansatz bringt einige Vorteile mit sich. Beispielsweise lassen sich nun auf simple Weise die Ergebnisse der reflektierten und gebeugten Pfade separat berechnen und anschließend kombinieren. So können sie im Mixer zusammenarbeiten, um die Geräuschkulisse zu formen. Auch haben transmittierte, gebeugte und reflektierte Pfade unter Umständen sehr unterschiedliche Längen. Die daraus resultierenden Unterschiede der Dissipation (Kapitel 3.1.5) von Schall in Luft kann somit ebenfalls getrennt betrachtet und berechnet werden.

Verzögerung von Schall über Distanz

Ein weiterer Aspekt von Schall, welcher mit einer geometrisch akkuraten Distanz berechnet werden kann, ist die Verzögerung von Schall in Luft. Naive Spatial Audio Engines verwenden dabei die direkte Distanz des Hörers zur Quelle. Wie schon bei der Lautstärke spiegelt dies allerdings nicht die Realität wider. In Verbindung mit der bereits vorgestellten Formel 2.1 ist es mit dieser Methode beispielsweise möglich, in einem First Person Shooter die exakte realistische Verzögerung zwischen einem Schuss und der Ankunft des Geräusches beim Spieler zu berechnen. Durch die Verwendung einer physikalisch akkuraten Funktion für die Berechnung der Schallgeschwindigkeit ist dies auch in diversen Klimas möglich.

3.1.2 Wahrgenommene Position der Quelle

Die Berechnung der Lautstärke einer Geräuschquelle anhand eines Pfades ist lediglich ein Aspekt, welcher die Geräuschkulisse realistisch gestaltet. Wie schon im vorigen Kapitel bietet das Beispiel einer Wand zwischen dem Hörer und der Quelle eine gute Möglichkeit zur Veranschaulichung. Die Lautstärke verhält sich nicht nur anders, wenn eine Wand den direkten Weg blockiert, auch hört sich der Schall so an, als würde die Quelle in einer anderen Richtung liegen. Um diesen Effekt darstellen zu können werden lediglich die Ursprünge p der letzten Strahlen aller validen Pfade n benötigt. Um die wahrgenommene Position p' zu bestimmen, wird im Endeffekt ein Durchschnitt aller Ursprünge der letzten Rays der validen Pfade gewichtet nach ihren Distanzen w_p berechnet.

$$w_p = \sum_{r=1}^{n} \frac{1}{1 + d_r} \tag{3.5}$$

[5]Undersampling bezeichnet den Fall, wenn für das Ergebnis eines Renders qualitativ nicht hochwertig ist, da zu wenig Samples getestet werden.

$$\vec{p'} = \frac{1}{w_p} \cdot \sum_{r=1}^{n} \frac{\vec{p_r}}{1 + d_r} \tag{3.6}$$

Eine einfachere Herangehensweise für die Berechnung der empfundenen Position wäre, alle p der Rays einfach zu mitteln. Jedoch weist die verwendete Methode einen klaren Vorteil auf. Bei der Ermittlung eines klassischen Durchschnitts können keine weiteren Daten des Rays das Resultat manipulieren. Werden die Positionen auf naive Weise gemittelt, so hat jede Position den gleichen Einfluss auf das Ergebnis. Die verwendete Methode gewichtet allerdings den Einfluss der Position anhand der Länge des Rays. Je kürzer ein Ray ist, desto stärker beeinflusst dessen p das Ergebnis. Da die wahrgenommene Lautstärke direkt von der Länge des Pfades abhängt, liefert der gewichtete Durchschnitt ein besseres Ergebnis. Sollten beispielsweise viele valide Pfade von sehr unterschiedlichen Richtungen auf den Spieler treffen, so wird der Hörer meist anhand der Unterschiede der Lautstärke erkennen, welcher der kürzeste Weg zur Quelle ist.

3.1.3 Spatial Blend

Die wahrgenommene Position der Geräuschquelle ist nur ein Teil, um dem Nutzer das Erlebnis von akkurater räumlicher Audiotrennung zu vermitteln. Ebenso wichtig ist der sogenannte Spatial Blend Parameter. Spatial Blend gibt an, wie direktional sich eine Quelle für einen Spieler anhört. Ein Spatial Blend Wert von 0 bedeutet keine räumliche Trennung und 1 bedeutet maximale räumliche Trennung. Der in dieser Arbeit verwendete Ansatz diesen Parameter dynamisch zu berechnen, benötigt keine weiteren Daten als die einfallende Richtung der validen Pfade. In Kombination mit der Spielerposition $\vec{p_p}$ und der bereits berechneten wahrgenommenen Position lässt sich die wahrgenommene einfallende Richtung $\vec{v_p}$ berechnen.

$$\vec{v_p} = \vec{p'} - \vec{p_p} \tag{3.7}$$

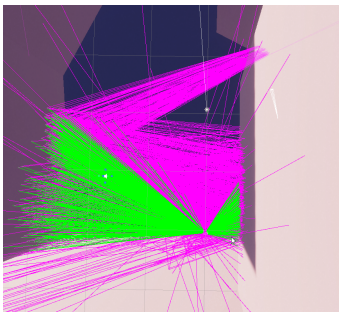
Diese Richtung des wahrgenommenen Schalls zum Spieler lässt sich nutzen, um eine maximale Abweichung davon zu berechnen. Diese Abweichung beschreibt den Öffnungswinkel eines Kegels aller eintreffenden validen Pfaden. Beispielsweise würde der berechnete Wert für Spatial Blend anhand Abbildung 3.6 annähernd 1 betragen. In dieser Abbildung kommen alle validen Pfade aus annähernd der gleichen Richtung. Der Öffnungswinkel des Kegels aller validen Pfade ist somit sehr klein und die maximale Abweichung eines Pfades vom gewichteten Mittel aller ist ebenfalls klein. Der genaue Gegensatz entsteht, wenn alle Pfade aus komplett unterschiedlichen Richtungen auf den Spieler einfallen. Dabei wird immer noch die wahrgenommene Position verwendet, allerdings ist der Öffnungswinkel des Kegels, bzw. des invertierten Kegels[6] sehr groß. Bei einer Abweichung von 180° beträgt der Wert für Spatial Blend $s = 0$. Formel 3.8 definiert dabei den ersten Berechnungsschritt, welcher den Winkel zwischen einem validen Pfad und dem gewichteten Mittel berechnet. Mit Formel 3.9 lässt sich nun der maximale Winkel und somit die maximale Abweichung vom Mittel bestimmen. Da dieser Wert ein Winkel ist und damit maximal 180° betragen kann, muss dieser anhand des maximalen Winkels normalisiert werden.

$$f\left(\vec{d}\right) = \left| \cos^{-1}\left(\frac{\vec{d} \cdot \vec{v_p}}{|\vec{d}| \cdot |\vec{v_p}|} \right) \right| \tag{3.8}$$

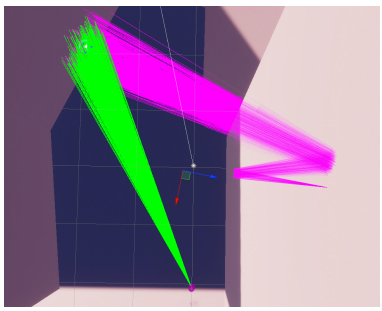
[6]Mit invertiertem Kegel ist eine Kugel gemeint, bei welcher ein kegelförmiges Segment fehlt

$$s = 1 - \frac{\max\left(f\left([N]\right)\right)}{180} \tag{3.9}$$

N Menge aller reflektierten validen Pfade
s Spatial Blend



(a) Spatial Blend = 0.12



(b) Spatial Blend = 0.95

Abbildung 3.7: Visualisierung von hoher und geringer Spatial Blend

3.1.4 Dopplereffekt

Der Dopplereffekt beschreibt die Veränderung einer gegebenen Frequenz bzw. Wellenlänge abhängig von der Bewegung der Quelle und des Hörers. Bedeutet die kombinierte Bewegung eine Annäherung beider, so erhöht sich die Frequenz, andernfalls wird sie verringert. Bewegt sich der Hörer auf die Quelle zu, wird die Geschwindigkeit v_p zu c addiert, andernfalls von c subtrahiert. c repräsentiert dabei die Schallgeschwindigkeit im Medium. Bewegt sich die Quelle auf den Hörer zu, so wird die Geschwindigkeit v_s von c subtrahiert, andernfalls addiert. Da die Tonhöhe (Pitch) in einem Audiomixer im Endeffekt eine direkte Multiplikation aller Frequenzen ist, bedeutet ein Pitch von 200% eine Verdoppelung der Frequenz. Aus diesem Grund reicht die simple Berechnung des Faktors für die Verwendung als Pitch. Für die Berechnung ist es wichtig, dass die Richtung beider verwendeten Geschwindigkeiten genau auf der relativen Richtung beider liegt. [vgl. Lib]

$$f = \left(\frac{c \pm v_p}{c \pm v_s}\right) \cdot f_0 \tag{3.10}$$

Ein resultierendes Problem, welches sich aus der Berechnung der wahrgenommenen Position ergibt, ist, dass sich die Position ständig leicht verschiebt. Diese Änderung verursacht dabei keine merkbaren Schwankungen der Position. Allerdings ergeben diese Schwankungen invalide ”Geschwindigkeiten” der Quelle. Diese würden in Formel 3.10 einen ständigen Faktor ergeben, welcher nicht perfekt 1 entspricht. Schon geringfügige Änderungen der Geschwindigkeit ergeben dabei hörbare Abweichungen von der normalen Tonhöhe. Die Berechnung der Geschwindigkeit des Spielers stellt dabei kein Problem dar. Für diese Geschwindigkeit wird nur die Richtung vom Spieler zur virtuellen Quelle benötigt. Darüber hinaus reagiert die virtuelle Position nicht innerhalb eines einzigen Bildes auf die Bewegung der originalen Quelle. Um dieses Problem zu umgehen, könnte die Quellgeschwindigkeit direkt verwendet werden. Diese Methode wäre allerdings bei weitem nicht akkurat und wurde deshalb nicht implementiert.

3.1.5 Manipulation der Frequenzbänder

Die Veränderung einzelner Frequenzen einer Schallwelle ist im Grunde essenziell für die Berechnung von Ausbreitung von Schall. In der Realität interferieren Wellen auf komplexe Art und Weise auf ihren diversen Wegen zum Hörer. Auch bei der Reflexion werden verschiedene Frequenzen unterschiedlich stark reflektiert bzw. absorbiert. Sogar die Ausbreitungsgeschwindigkeit ist abhängig von der Frequenz. Der physikalisch korrekte Effekt lässt sich im Rahmen einer Echtzeitanwendung nicht umsetzen, da das Betrachten vieler einzelner Frequenzen äußerst aufwändig ist. Allerdings lässt sich ein vereinfachter Effekt mit einigen festen Equalizern (EQs) an festen Frequenzen umsetzen. Die Implementation des Algorithmus wird in diesem Fall vier verschiedene Frequenzbänder behandeln.

- Tiefe Frequenzen bei 40 Hz mit 2 Oktaven Umfang (Low Band)
- Mittlere Frequenzen bei 300 Hz mit 2 Oktaven Umfang (Mid Band)
- Mittelhohe Frequenzen bei 2250 Hz mit 2 Oktaven Umfang (HighMid Band)
- Hohe Frequenzen bei 10000 Hz mit 2 Oktaven Umfang (High Band)

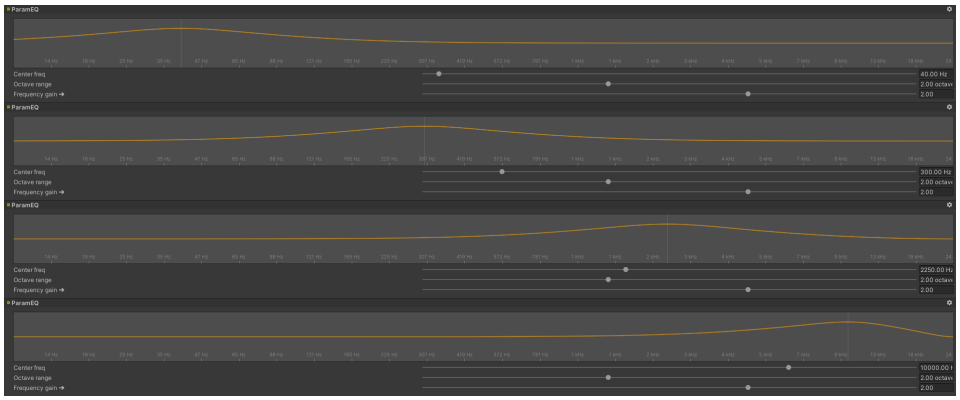


Abbildung 3.8: Darstellung der definierten Frequenzbänder

Absorption bei Reflexion

Kein Spiegel ist perfekt in der Realität und genauso verhält sich die Reflexion von Schall. Verschiedene Materialien brechen Schall nicht nur unterschiedlich, sie absorbieren auch diverse Frequenzen verschieden stark. Aus diesem Grund kann jedes relevante Objekt mit vier weiteren Parametern definiert werden. Entweder mit vier Absorptionswerten für jedes Frequenzband oder mit vier Reflexionswerten. Dabei stehen die Absorption a und die Reflexion r in einem direkten Verhältnis zueinander ($a = 1 - r$ bzw. $r = 1 - a$). Verschiedene Szenarien benötigen entweder den Absorptions- oder den Reflexionskoeffizienten. Mit diesem Zusammenhang ist es einfach zwischen den beiden Werten zu konvertieren.

Um die verschiedenen Frequenzbänder zu berechnen, benötigt der Strahl mehr Informationen auf seinem Pfad. Da ein Pfad auf seinem Weg multiple Male reflektiert werden kann, müssen all die gesammelten Informationen auf simple Weise beschrieben werden. Aus diesem Grund werden die Informationen des Strahls um vier Werte erweitert. (Tabelle 3.2) Diese Frequenzbandinformationen der Strahlen beginnen alle mit einem Wert von 1. Mit jeder

Reflexion werden die Bänder anhand der getroffenen Oberfläche mit dem definierten Reflexionskoeffizienten r multipliziert. Dieser Prozess der Multiplikation wiederholt sich so oft, bis die maximale Rekursionstiefe n des Pfades erreicht wurde. (Abb. 3.4 & Formel 3.11) Jeder Audio Mixer der Quelle wird ebenfalls diese vier EQs besitzen. Solch ein Ansatz ermöglicht die Charakteristiken der verschiedenen Pfade unabhängig voneinander zu berechnen.

Modul benötigt	Ray Information			
Grundsätzlich	Ursprung	Richtung	Ray Länge	
Lautstärke	Pfadlänge	wurde gebeugt	Spieler getroffen	
Frequenzbänder	Low Band	Mid Band	MidHigh Band	High Band

Tabelle 3.2: Benötigte Informationen eines Strahls (Ray) für die Berechnung der Lautstärke und Frequenzbänder.

$$b = \prod_{p=1}^{n} r \tag{3.11}$$

Die Selektion der Pfade, welche die tatsächlichen Gain Parameter der EQs manipulieren werden, sind dieselben, welche bei der Berechnung der Lautstärke verwendet wurden. Das Verfahren der Multiplikation (Formel 3.11) wurde gewählt, weil diese einerseits eine simple, annähernd physikalisch korrekte Lösung darstellt und andererseits die direkte Verwendung des Resultats als Gain Parameter ermöglicht. Ein Wert von 1 beschreibt dabei keinerlei Manipulation des Bandes. Größere Werte verstärken das gegebene Band und kleinere schwächen es. Die Daten der selektierten Pfade werden, genau wie bei der Berechnung der Lautstärke, gemittelt.

Gebeugte Pfade

Das Beugen von Wellen tritt nur bei Objekten in der gleichen Größenordnung wie die Wellenlänge der Frequenz auf. Das heißt, dass nur Objekte, welche ca. so groß wie die Wellenlänge der Frequenz sind, diese überhaupt beugen können. Aus diesem Grund werden in der Regel nur Frequenzen unter 400-500 Hz effektiv in der echten Welt gebeugt. Um das Ergebnis für den Hörer zu verbessern, wird hierbei angenommen, dass Teile der gebeugten Pfade durch Reflexion zum Spieler gelangen können. Aus diesem Grund werden nicht nur Frequenzen unter der 500Hz Marke bei Diffraktion weitergegeben, sondern auch höhere in einer abgeschwächten Form. Falls ein Pfad keine Geometrie trifft, sendet dieser einen gebeugten Pfad aus. (Tabelle 3.4) In diesem Moment wird angenommen, dass dieser Pfad sich um ein Objekt gebeugt hat. Dabei ist es irrelevant, ob dies der Fall ist oder nicht. In diesem Moment werden die bereits gesammelten Daten der Frequenzbänder wie folgt manipuliert.

- Low Band: $b_{lb} = b_l$
- Mid Band: $b_{mb} = 0.75 \cdot b_m$
- HighMid Band: $b_{hmb} = 0.1 \cdot b_{hm}$
- High Band: $b_{hb} = 0$

Dieser Ansatz bietet überzeugende Resultate, da die Lautstärke einer Quelle damit nicht sofort auf null abfällt, falls kein reflektierter Pfad zum Hörer findet. Wie schon erwähnt, ist dieser Ansatz bei weitem nicht realistisch. Allerdings ist das Resultat wesentlich immersiver als keine stark approximierte Berechnung der Beugung.

Dissipation von Schall in Luft

Die sogenannte Dissipation beschreibt die atmosphärische Absorption von verschiedenen Frequenzen über eine gegebene Distanz. Diese wird dabei noch zusätzlich zum regulären Abfall berechnet. Vereinfacht lässt sich sagen, dass höhere Frequenzen schneller absorbiert werden und daher eine wesentlich geringere effektive Reichweite haben. Die Stärke der Absorption einer gegebenen Frequenz f hängt dabei vom Luftdruck p, der Temperatur t_c, t und vor allem der Luftfeuchtigkeit h ab. [vgl. Send]

$$p_r = \frac{p}{p_a} \tag{3.12}$$

$$t_r = \frac{t}{293.15} \tag{3.13}$$

$$p_s = \begin{cases} t_c \leq 0 & \frac{e^{34.494 - \frac{6545.8}{t_c+278}}}{(t_c+868)^2} \\ t_c > 0 & \frac{e^{34.494 - \frac{4924.99}{t_c+237.1}}}{(t_C+868)^{1.57}} \end{cases} \quad \text{[Fia18]} \tag{3.14}$$

$$v_c = \frac{h \cdot p_s}{p_a} \tag{3.15}$$

$$f_O = p_r \cdot \left(24 + \frac{4.04 \cdot 10^4 \cdot v_c \cdot (0.02 + v_c)}{0.391 + v_c}\right) \tag{3.16}$$

$$f_N = \frac{p_r}{\sqrt{t_r}} \cdot \left(9 + 280 \cdot v_c \cdot e^{-4.17 \cdot \left(t_r^{-\frac{1}{3}} - 1\right)}\right) \tag{3.17}$$

$$dis = 8.989 \cdot f^2 \cdot \left(\frac{1.84 \cdot 10^{-11} \cdot \sqrt{t_r}}{p_r} + t_r^{-2.5} \cdot \left(\frac{0.01275 \cdot e^{-\frac{2239.1}{t}}}{f_O + \frac{f^2}{f_O}} + \frac{0.1068 \cdot e^{-\frac{3352}{t}}}{f_N + \frac{f^2}{f_N}}\right)\right) \tag{3.18}$$

p Luftdruck
p_a Luftdruck auf Höhe 0
t Temperatur in Kelvin
t_c Temperatur in Grad Celsius
h Luftfeuchtigkeit in Prozent
dis Dissipation von gegebener Frequenz pro Meter

Da die einzige Formel, welche die Frequenz als Eingabeparameter benötigt, 3.18 ist, lassen sich alle anderen benötigten Parameter vorberechnen. Diese müssen erst erneut berechnet werden, wenn sich die Parameter der Umgebung ändern. Das Resultat dis von 3.18 ist ein $\frac{dB}{m}$ Wert, welcher den zusätzlichen Abfall der SPL Lautstärke über eine gegebene Distanz darstellt. Um den absoluten Abfall der Lautstärke zu berechnen, muss das Ergebnis lediglich mit der Distanz d multipliziert und von der bereits berechneten Lautstärke subtrahiert werden. Die verwendete Distanz ist dabei dieselbe, welche zur Berechnung der Lautstärke genutzt wurde. Da allerdings die Lautstärke für verschiedene Frequenzen unterschiedlich stark abfällt, kann diese Berechnung nicht auf die generelle Lautstärke L angewendet werden. Um einen Faktor für die Manipulation der Frequenzbänder zu berechnen, wird mit Formel 3.19 ein relativer Abfall der Lautstärke des Frequenzbandes berechnet. Das Resultat b_d kann

anschließend einfach mit den bereits berechneten Werten der Frequenzbänder multipliziert werden.

$$b_d = \frac{L_{\text{dist}} - d \cdot dis}{L_{\text{dist}}} : 0 <= b_d <= 1 \tag{3.19}$$

L_{dist} Allgemeine Lautstärke bei Distanz d

Anders als die Berechnung der Bänder der gebeugten Pfade wird die Dissipation für jeden Mixer bestimmt (direkte bzw. reflektierte und gebeugte Pfade). Da die verschieden kategorisierten Pfade unterschiedliche durchschnittliche Distanzen zurückgelegt haben, muss nicht nur die generelle Lautstärke einzeln berechnet werden, sondern auch die Dissipation.

3.1.6 Echo

Sodass ein Echo klar wahrnehmbar ist, müssen in der Umwelt einige Faktoren zusammenspielen.

- Die Quelle sollte am besten direkt hörbar sein.
- Ein Teil der Schallwelle wurde von einer ausreichend großen Fläche uniform reflektiert.
- Der zeitliche Unterschied zwischen der Primär- und der Sekundärwelle beträgt mindestens 10ms.

Um diesen Sachverhalt darstellen zu können, benötigen die Objekte in der Szene und die Strahlen neue Informationen. Da der Algorithmus an sich nur sehr schwer Geometriedaten, wie die Orientierung und Position mehrerer Objekte zueinander, in Echtzeit auswerten kann, muss dafür eine alternative Lösung gefunden werden. Diese verleiht dem Ersteller der Szene die Möglichkeit einzelne Objekte, als sogenannte Wände zu definieren. Nur Strahlen, welche auf ihrem Weg von einer Wand reflektiert wurden, werden in der Berechnung des Echos berücksichtigt. (Tabelle 3.3) In dieser spezifischen Implementation wird nur ein einziges Echo für alle Pfade berechnet. Allerdings ist dies keine Limitation des Systems, sondern einfach eine Wahl. Es spricht nichts dagegen, für jeden Mixer ein separates Echo zu berechnen. Dazu kommt noch, dass das Echo in der Theorie auch einzeln räumlich berechnet werden kann. Da allerdings Unity keine eingebaute Möglichkeit für die Manipulation der Stereokanäle bietet, konnte dies in der Arbeit leider nicht praktisch umgesetzt werden.

Die Berechnung des Echos an sich ist dabei ziemlich simpel. Aus allen Pfaden wird der kürzeste Pfad herausgefiltert, welcher die minimale Verzögerung des Schalls zum Hörer beschreibt. Für die Berechnung der Verzögerung der reflektierten Pfade existieren multiple Ansätze.

- Das Mitteln der Distanz aller reflektierten Pfade. Dies liefert ein gutes Ergebnis für komplexe Räume, allerdings sind die Resultate im simpelsten Fall (Quelle, Hörer und eine Wand) äußerst falsch.
- Das Verwenden der kürzesten Distanz aller reflektierten Pfade, welche von einer Wand umgelenkt wurden. Dies liefert ein akkurates Ergebnis im simplen Fall. Allerdings führt dieser Ansatz dazu, dass bei komplexen Räumen keinerlei Echo berechnet werden kann, da der kürzeste reflektierte Pfad ebenfalls der allgemein kürzeste Pfad ist und die Längendifferenz damit 0.

- Das Mitteln der Distanz aller reflektierten Pfade, um alle Distanzen, welche unter diesem Mittel liegen, zu selektieren und zu mitteln. Dieser Ansatz ist im Grunde ein Kompromiss der beiden vorigen Ideen. Da es sich um eine hybride Methode handelt, ist sie in beiden Fällen nicht perfekt, liefert allerdings zufriedenstellende Ergebnisse in beiden Situationen.

Modul benötigt	Ray Informationen			
Grundsätzlich	Ursprung	Richtung	Ray Länge	
Lautstärke	Pfadlänge	wurde gebeugt		
Frequenzbänder	Low Band	Mid Band	HighMid Band	High Band
Echo	Wand getroffen			

Tabelle 3.3: Benötigte Informationen eines Strahls (Ray) für die Berechnung der Lautstärke, Frequenzbänder und Echo.

Für die Berechnung der Verzögerung der reflektierten Pfade wird die zuletzt aufgezeigte Methode verwendet. Anhand der Subtraktion der beiden Verzögerungen erhält man die zeitliche Verschiebung des Echos. Diese sollte dabei mindestens 10ms betragen. Ist die Verzögerung kleiner als 10ms, so wird die Lautstärke des Echos auf 0dB gesetzt. Dieses Verfahren wurde gewählt, da Menschen ein Echo mit einer Verzögerung von unter 10ms als Hall wahrnehmen und nicht als Echo. [vgl. Ref13] Da das Echo nur durch valide reflektierte Pfade verursacht werden kann, ist der Mixer des Echos dem Mixer der reflektierten Pfade hierarchisch untergeordnet. Dadurch lassen sich die Dissipation der reflektierten Pfade für das Echo verwenden. Zusätzlich dazu wird beim Echo Mixer noch ein High Pass Filter und ein High Roll Off EQ verwendet. Der High Pass Filter simuliert die sehr wahrscheinliche Phasenaufhebung der tiefsten Frequenzen. Das High Roll Off stellt eine zusätzliche, künstliche Dissipation dar. (Abb. 3.9)

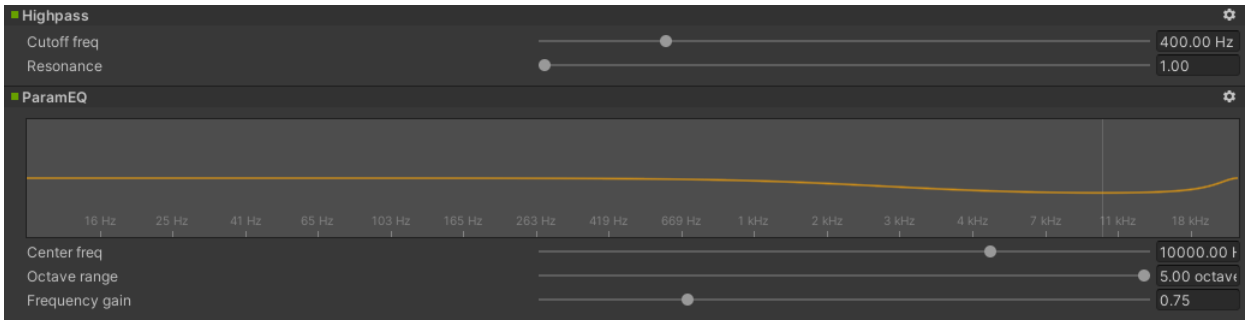


Abbildung 3.9: Darstellung des High Pass Filters und des High Roll Off in Unity.

Die Dissipation könnte auch erneut für das Echo berechnet werden, um ein genaueres Ergebnis zu erhalten. Der High Pass Filter hilft dabei, dass die tiefsten Frequenzen nicht das Profil des Sounds dominieren. Da der Echoeffekt nicht die genaue Realität widerspiegelt, klingt ein Echo mit den vollen tiefen Frequenzen zu stark nach einer direkten Kopie des Sounds und nicht nach einem tatsächlichen Echo. In diesem Fall würde es sich gegebenenfalls sogar lohnen eine Interferenz der tiefen Frequenzen zu berechnen. Bei der gewählten Frequenz des tiefen Bandes beträgt die Wellenlänge 8.5m[7]. Das heißt bei einer Phasenverschiebung von

[7]Akkurat bei 14.2°C; Allerdings nach Formel 2.2 abhängig von Temperatur.

$n \cdot \frac{8.5m}{2}$ würden sich die tiefen Frequenzen gegenseitig aufheben. Allerdings würde die Manipulation des Frequenzbandes annehmen, dass sich alle inbegriffenen Frequenzen von 10 - ca. 300Hz ebenfalls aufheben würden.

3.1.7 Hall

Hall ist der Effekt, welcher am schwierigsten auf eine dynamische Weise parametrisiert werden kann. Darüber hinaus weisen die Halleffekte in diversen Softwarelösungen jeweils unterschiedliche Parametrisierungen vor. Dazu kommt noch, dass die Berechnung für die Zeit des Nachhalls eine der wenigen, in der Wirklichkeit genutzten, Formeln verwendet. Aus diesen Gründen wird in diesem Abschnitt nur die Berechnung der Zeit des Nachhalls beschrieben. Alle weiteren Berechnungen der Parameter werden im Abschnitt der Implementation auf eine experimentelle Weise festgelegt. Die klassische Formel für die Bestimmung des Nachhalls eines Geräusches geht von geschlossenen Räumen aus. Aus diesem Grund verwendet die sabinsche Formel (3.20) für die Berechnung der Zeit ein Verhältnis der Oberfläche zum Volumen des Raums. Das Ergebnis der Formel repräsentiert die Zeit, welche benötigt wird, sodass die Lautstärke des Halls direkt nach dem Verstummen einer Quelle um 60dB abfällt. [vgl. The15, Sabine's Formula]

$$RT_{60} = \frac{24 \cdot \ln 10 \cdot V}{c \cdot S_a} \tag{3.20}$$

Allerdings spiegelt diese Annahme eines geschlossenen Raums bei weitem nicht jede Situation wider. Sind die Quelle und der Hörer gegebenenfalls im Freien, so gibt es kein einfach definierbares Volumen und die Genauigkeit der Formel ist dadurch nicht garantiert. Trotzdem wird eine Möglichkeit zur dynamischen Berechnung des Halls im Tieferen betrachtet. Das verwendete Volumen in der Berechnung wird anhand aller validen Pfade, der Position der Quelle und aller relevanten Objekte bestimmt. Zunächst werden die minimalen und maximalen Werte jeder Achse bestimmt. Alle Werte, welche diese manipulieren, sind die Ursprünge bzw. Treffpunkte der validen Pfade und die ursprüngliche Position der Quelle. Anhand dieser sechs Werten lässt sich eine Bounding Box definieren, welche dazu verwendet werden kann, um alle anderen Körper zu ermitteln, welche dieses Volumen schneiden. Diese Objekte können dabei ein Script besitzen, welches das Volumen und die Oberfläche der Geometrie im Vorhinein berechnet und speichert. Das Volumen eines Meshes[8] lässt sich mit der Summe der Volumen aller Tetraeder berechnen. Ein Tetraeder wird von einem Dreieck der Oberfläche und dem Ursprung des Meshes definiert. Dabei kann das Volumen eines einzelnen Tetraeders entweder positiv bei konvexer Normale oder negativ bei konkaver Normale sein. Da die Formel in mathematischer Darstellung äußert lang und unübersichtlich ist, wird in diesem Fall einfach der verwendete Code gezeigt. (Abb. 3.10)

Vom gesamten Volumen werden alle Volumen der darin liegenden Körper subtrahiert, um das effektive Volumen zu berechnen, welches vom Schall tatsächlich ausgefüllt werden kann. Zusätzlich zum Volumen verlangt die sabinsche Formel auch noch die gesamte Fläche des Raumes multipliziert mit der durchschnittlichen Absorption. Die Berechnung der gesamten Oberfläche eines Meshes ist einfach die aufsummierte Fläche aller einzelnen Dreiecke. Formeln 3.21 und 3.22 stellen eine für Vektor Arithmetik optimierte Lösung für die Flächen-

[8] Ein Mesh beschreibt die Topologie und die Geometrie eines Körpers. Das Mesh definiert Punkte im dreidimensionalen Raum und Verbindungen zwischen diesen. Anhand dieser Daten lassen sich Oberflächen in der Computergrafik darstellen.

```
private float CalculateVolumeOfTriangle(Vector3 a, Vector3 b, Vector3 c)
{
    var v321 = c.x * b.y * a.z;
    var v231 = b.x * c.y * a.z;
    var v312 = c.x * a.y * b.z;
    var v132 = a.x * c.y * b.z;
    var v213 = b.x * a.y * c.z;
    var v123 = a.x * b.y * c.z;
    return (1.0f / 6.0f) * (-v321 + v231 + v312 - v132 - v213 + v123);
}
```

Abbildung 3.10: Berechnung des Volumens mit Vorzeichen.
https://stackoverflow.com/questions/1406029/how-to-calculate-the-volume-of-a-3d-mesh-object-the-surface-of-which-is-made-up

berechnung eines Dreiecks dar. Dabei wird davon ausgegangen, dass $\vec{a}, \vec{b}$ und $\vec{c}$ die Eckpunkte des Dreiecks im Weltkoordinatensystem definieren.

$$\vec{n} = \left(\vec{b} - \vec{a}\right) \times (\vec{c} - \vec{a}) \tag{3.21}$$

$$A = \vec{n} \cdot \frac{2}{|\vec{n}|} \tag{3.22}$$

Da S_a nicht die reine Oberfläche des Raumes darstellt, sondern die Oberfläche des gegebenen Raumes A multipliziert mit der durchschnittlichen Absorption a, so lässt sich S_a des Raumes mit Formel 3.23 bestimmen. Um die Berechnung der Zeit nicht zu stark zu verfälschen, dürfen disproportional große Objekte (riesige Bauten, Boden) im Vergleich zum Hall-Volumen ihre Daten nicht in die Berechnung einfließen lassen.

$$S_a = \sum_{m=1}^{k} A_m \cdot a_m \tag{3.23}$$

k Anzahl der Objekte im Volumen des Raumes V

3.2 Hardware Ray Tracing

3.2.1 Verbesserungen

Da der Algorithmus zu diesem Zeitpunkt komplett definiert ist, können nun mögliche Verbesserungen mithilfe von anderen Implementationen erarbeitet werden. Wie schon im Vorhinein erwähnt, agiert die bestehende Implementation nur auf der CPU. Die größte Limitation für diese Herangehensweise ist die mangelnde Leistungsfähigkeit. Aus diesem Grund sind die berechneten Parameter leicht instabil und komplexe Pfade zum Spieler mit mehr als sechs Reflexionen werden bzw. können nicht gefunden werden. Die Begründung dafür ist, dass mit der limitierten Anzahl an Ray Casts zwischen Parameterstabilität und Pfadlänge abgewogen werden muss. Je öfter ein Pfad reflektieren kann, desto mehr Rays werden in der sogenannten Tiefe benötigt. Bei diesem Ansatz können weniger primäre Samples ausgesendet werden, was die Parameter allgemein instabiler macht. Wird der Fokus andererseits auf die Menge der Grundsamples gelegt, so können komplexe Pfade nicht gefunden werden, da die maximale Tiefe dafür nicht ausreicht.

Mit der Verwendung der Grafikkarte können, auch ohne Hardware Ray Tracing, die maximale Zahl der Rays stark erhöht werden. Ein Limit von ein paar tausend Rays befindet sich nun bei einem Limit von einigen zehn- bis hunderttausend. Da für die Implementation von Ray Tracing auf einer Grafikkarte nicht unbedingt die spezielle Hardware notwendig ist, ermöglicht dies den Algorithmus mit einer weiteren Implementation auf älteren Grafikkarten laufen zu lassen. Allerdings bietet die Benutzung der speziellen Hardware einige Vorteile.

- Gesteigerte Leistung gegenüber Software Ray Tracing auf der GPU.
- Vorgefertigte Berechnung des BSP Trees[9] und Upload auf GPU.
- Vorgefertigtes dediziertes Shader Code[10] System.

Durch die erhöhte Leistung können in einer guten Zeit zum einen 512 anstatt von 50 primären Strahlen und zum anderen eine Rekursionstiefe von 16 anstelle von 6 verwendet werden. Dies verbessert sowohl die Stabilität der Parameter, sowie die Anzahl der gefundenen komplexen Pfade. Da der Algorithmus nun auf der Grafikkarte ausgeführt wird, ist es möglich für die Absorption und die Diffusion Texturen zu verwenden. Dies ist zwar auf der CPU auch möglich, allerdings ist dort das sogenannte Samplen[11] einer Textur eine teure Operation, was die Anzahl der maximalen Ray Casts noch weiter senken würde. Die Möglichkeit Texturen zu verwenden, erlaubt es Objekten komplexere Charakteristiken für die Ausbreitung des Schalls zu verleihen. Beispielsweise können Objekte mit Vertiefungen in diesem Fall stellenweise den Schall tatsächlich absorbieren. Da die Implementation mit Shader Code sich im Prinzip grundsätzlich unterscheidet, benötigen die Strahlen noch zusätzliche Daten, um überhaupt ein Ergebnis liefern zu können.

Modul benötigt	Ray Information				
Grundsätzlich	Ursprung	Richtung	Ray Länge		
Lautstärke	Pfadlänge	wurde gebeugt	Spieler getroffen		
Frequenzbänder	Low Band	Mid Band	MidHigh Band	High Band	
Hardware RT	Spielerposition	TOH Distanz	Ray prüft auf Spieler	Tiefe	Seed

Tabelle 3.4: Benötigte Informationen eines Strahls (Ray) für die Berechnung der Lautstärke, Frequenzbänder und Echo für Hardware Ray Tracing.

3.2.2 Transmission von Schall

Eine weitere Möglichkeit, welche sich durch die gesteigerte Leistung ergibt, ist die Berechnung der Transmission von Schall. Dieser Aspekt der Ausbreitung von Schall hat das Potenzial wesentlich auffälliger als die Beugung von Schall zu sein. Beispielsweise ließe sich mit einer ausreichend guten Methode Schall durch geschlossene Türen oder der Bass außerhalb von Diskotheken dynamisch darstellen. Bei der Berechnung von Transmission wird sich in dieser

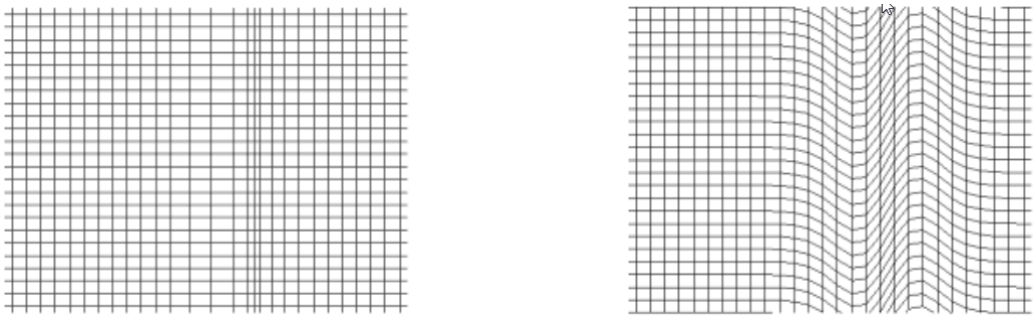
[9] "Binary Space Partitioning ist ein Verfahren zur rekursiven Unterteilung eines Raums in zwei konvexe Mengen unter Verwendung von Hyperebenen als Partitionen. Dieser Prozess der Unterteilung führt zur Darstellung von Objekten innerhalb des Raums in Form einer Baumdatenstruktur, die als BSP Tree bekannt ist." [gee20]

[10] Shader Code sind Instruktionen, welche auf einer GPU ausgeführt werden. Shader Code ist dabei grundsätzlich anders in der Funktionsweise zu normalen Programmcode. Shader Code kann dabei pro Vertex, Fläche, Pixel oder Ray ausgeführt werden.

[11] Das Samplen einer Textur beschreibt den Prozess der Bestimmung der Farbe an einer UV Koordinate.

Arbeit auf zwei Aspekte fokussiert. Das Weitergeben einer eigentlich absorbierten Schallwelle und das Durchdringen einer Welle bei bestimmten Dicken eines Objekts. Die Ausbreitung von Wellen in Festkörpern ist allgemein komplexer als in Gasen und Flüssigkeiten. Beim Eintreffen einer Welle auf ein Objekt kann diese Welle in eine Longitudinalwelle (Abb. 3.11a) und eine Transversalwelle zerlegt werden. (Abb. 3.11b) Diese beiden verschiedenen Fortbewegungsmöglichkeiten haben zusätzlich noch jeweils zwei verschiedene Schallgeschwindigkeiten. [vgl. Gar20, 4.2.2 & 4.2.3] Bei der Berechnung der Transmission werden allerdings diese beiden separaten Aspekte nicht physikalisch komplett korrekt behandelt. Bei jedem Übergang von Schall in ein anderes Medium, wird auch diese, wie jede andere Welle, gebrochen. Für diese Brechung gilt ebenfalls das snelliussche Gesetz. (Formel 3.24)

$$\frac{\sin \delta_1}{\sin \delta_2} = \frac{L_1}{L_2} = \frac{n_2}{n_1} \tag{3.24}$$



(a) Longitudinalwelle
https://upload.wikimedia.org/wikipedia/commons/6/62/Onde_compression_impulsion_1d_30_petit.gif

(b) Transversalwelle
https://upload.wikimedia.org/wikipedia/commons/6/6d/Onde_cisaillement_impulsion_1d_30_petit.gif

Abbildung 3.11: Visualisierung der beiden verschiedenen Ausbreitungsmöglichkeiten von Schall in einem Feststoff.

Ein weiterer Effekt, welcher bei der Transmission von Schall auftritt, ist die widerstandslose Transmission von bestimmten Frequenzen durch bestimmte Materialstärken. Betrachtet man beispielsweise eine Wand zwischen zwei Räumen, so durchdringen Wellen, deren Wellenlänge ein Viertel der Wandstärke d betragen, widerstandslos die Wand. Zusätzlich zu dieser grundlegenden Wellenlänge durchdringen auch Vielfache von aufsummierten halben Wellenlängen die Trennwand. Abgeleitet von diesem Phänomen lässt sich Formel 3.25 herleiten. Für Werte von $n \in \mathbb{N}_0$ erhält man Frequenzen $f(n)$, welche ebenfalls widerstandslos das trennende Medium durchdringen können. [vgl. Gar20, 11.2.2]

$$f(n) = \frac{c}{4d + 2dn} \tag{3.25}$$

c Schallgeschwindigkeit in Luft

Leider kann dieser Effekt in dieser Arbeit nur in einer vereinfachten Form verwendet werden, da die Manipulation von einzelnen Frequenzen nicht möglich ist. Für die Berechnung von Transmission von Schall muss ein eintreffender Strahl zunächst anhand der gegebenenfalls unterschiedlichen Schallgeschwindigkeiten gebrochen werden. Da für eine Brechung eine weitere Schallgeschwindigkeit benötigt wird, erhalten alle relevanten Objekte zusätzlich zur Absorption, Diffusion, Volumen und Oberfläche nun auch noch eine Geschwindigkeit. Für

die Berechnung der gebrochenen Richtung existiert praktischerweise eine dedizierte Funktion, welche anhand des einfallenden Winkels und des Brechungsindexes die Operation auf Validität prüft. Ist der einfallende Winkel zu steil, sodass eine Brechung mit dem gegebenen Index unmöglich ist, so wird an diesem Punkt keine Transmission berechnet. Andererseits wird ein neuer Strahl in Richtung der Brechung geleitet. Am nächsten Schnittpunkt wird bestimmt, ob von diesem Punkt der Spieler erreicht werden kann. Ist dies der Fall, so werden die Frequenzbandinformationen auf folgende Weise modifiziert. Die Länge des letzten Pfades stellt die zurückgelegte Distanz des Strahls innerhalb des Mediums dar. Anhand dieser Distanz wird die höchste Frequenz berechnet ($f(0)$), welche ohne Verluste diese Distanz überqueren kann. Alle Frequenzen der Bänder, welche kleiner oder gleich dieser gegebenen Frequenz sind, werden nicht manipuliert. Die restlichen Frequenzbänder werden mit der Absorption für das gegebene Band multipliziert.

Modul benötigt	Ray Information				
Grundsätzlich	Ursprung	Richtung	Ray Länge		
Lautstärke	Pfadlänge	wurde gebeugt	Spieler getroffen		
Frequenzbänder	Low Band	Mid Band	MidHigh Band	High Band	
Hardware RT	Spieler Position	TOH Distanz	Ray prüft auf Spieler	Tiefe	Seed
Transmission	ist im Objekt	wurde transmittiert			

Tabelle 3.5: Benötigte Informationen eines Strahls (Ray) für die Berechnung der Lautstärke, Frequenzbänder, Echo und Transmission für Hardware Ray Tracing.

3.3 Nicht erforschte Möglichkeiten

Da diese Arbeit versucht eine allgemein valide Herangehensweise für die Berechnung von Schall zu finden, sind die verwendeten Techniken nicht unbedingt die optimalen. Darüber hinaus hat jede genutzte Software Limitationen, welche es eventuell verbieten, bestimmte Vorgehensweisen zu verwenden. Aus diesen Gründen werden in diesem Abschnitt multiple Möglichkeiten, deren Umsetzung nicht möglich war, aufgezeigt.

3.3.1 Beugung von Schall in definierten Zonen

Da die bisherige Berechnung von Beugung eine äußerst grobe Approximation darstellt, könnte die Genauigkeit mit dieser Methode gesteigert werden. Die Idee der definierten Zonen beruht auf der Herangehensweise, welche Wwise Spatial Audio verwendet. Ähnlich zu den dort verwendeten Portalen, können in diesen definierten Bereichen Schallwellen gebeugt werden. Eine einfache, prozedurale Methode für die Bestimmung dieser Zonen wäre es, das Mesh eines Objektes um einen bestimmten Faktor hoch zu skalieren. Tritt ein Strahl in dieses Volumen ein, so kann ein Brechungsindex verwendet werden, um eine Beugung darzustellen. Dabei kann auch die totale interne Reflexion dabei helfen, den Strahl um ein Objekt herumzuleiten. (Abb. 3.12) Dieser feste Brechungsindex kann dabei auf eine experimentelle Weise bestimmt werden.

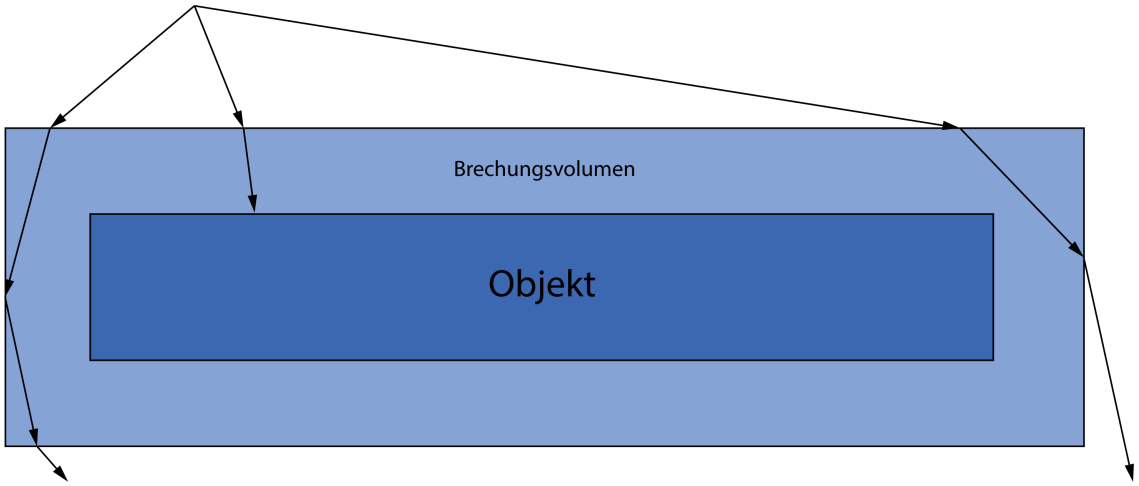


Abbildung 3.12: Darstellung zweier gebeugten Pfade mithilfe eines Brechungsvolumen eines Objekts.

Dieser Ansatz würde die Beugung wesentlich akkurater darstellen, allerdings bringt diese Methode einige Probleme mit sich. Zum einen sind für die Berechnung der Beugung in diesem Fall bis zu drei Rays notwendig. Dies würde eine wesentlich höhere Tiefe an Rekursion benötigen, um genauso komplexe Pfade wie die nun verwendete Methode zu finden. Zusätzlich muss die doppelte Menge an Meshinformationen verwaltet werden. Darüber hinaus würden Pfade, bevor sie auf das Objekt treffen, auf eine falsche Weise abgelenkt werden.

3.3.2 Verwendung von Portalen

Diese Idee beruht ebenfalls auf den Portalen von Wwise Spatial Audio-Plugin. Mit der Verwendung von Portalen können bewusst Hinweise vom Nutzer gesetzt werden, wo komplexe Pfade in der Szene entstehen können. Mit dieser Technik könnten weniger Samples benötigt werden, um genauso gute Ergebnisse zu erzielen, wie eine Berechnung mit vielen Samples. Allerdings würde die Verwendung dieser Technik die Arbeit für den Nutzer wieder steigern. Damit geht der Vorteil der geringen Parametrisierung in dieser Arbeit verloren.

3.3.3 Multiple Stereo separierte Echos

Da Unity nativ leider keine expliziten Stereo bzw. räumlichen Berechnungen vom Code aus erlaubt, ist diese Technik nur schwer zu realisieren. Allerdings wäre es möglich anhand der validen Pfade zu berechnen, aus welcher Richtung ein Echo auf den Hörer trifft. Dies erlaubt eine realistischere Darstellung von Echo, da das Echo nun nicht mehr zwangsweise aus der gleichen Richtung wie die Quelle kommen muss. Zusätzlich dazu können noch weitere Echos berechnet und räumlich eingeordnet werden.

3.3.4 Eigenständige Hall-Volumen

Ähnlich zu Stereo separierten Echos können auch multiple eigenständige Hall-Volumen berechnet werden. Beispielsweise könnte ein Wald als ein Hall-Volumen definiert werden. Wenn Schall in dieses Volumen gelangt, wird ein Hall berechnet und von dort wieder die Lautstärke zum Hörer. Diese Technik würde die Berechnung des Halls nicht nur auf die direkte Umgebung der Quelle limitieren. Diese Methode verlangt allerdings entweder eine manuelle Einstellung der Hall-Volumen oder eine dynamische Berechnung der Hall Parameter. Die

erste Methode würde, wie bei der Verwendung von Portalen, die Arbeit für den Nutzer erhöhen. Zusätzlich liefert sie keine Möglichkeit auf Veränderungen der Szene zu reagieren. Würde allerdings die dynamische Berechnung verwendet, so muss für jedes Hall-Volumen, sobald es getroffen wurde, dessen Parameter berechnet und zusätzlich wieder ein Pfad zum Spieler gefunden werden.

4 Implementation

4.1 Genutzte Software

Um den in dieser Arbeit erstellten Algorithmus entwickeln zu können, wird bereits vorgefertigte Software benutzt. Allerdings muss die verwendete Game Engine einigen Anforderungen gerecht werden.

1. Ein Physiksystem mit Ray und Box Casting.
2. Ein Grafiksystem, welches erlaubt Daten auf der GPU zu berechnen und davon zu extrahieren.
3. Ein Grafiksystem mit der Möglichkeit Hardware Ray Tracing GPUs anzusprechen.
4. Ein Grafiksystem dessen Shader Code frei editier- und erweiterbar ist.
5. Ein Audiosystem mit Spatial Audio.
6. Ein Audiosystem mit Send, Receive, Pitch, EQ, Hall und Echo Effekten.
7. Ein Audiosystem mit frei parametrisierbaren Effekten und Sound Quellen.

Diese Liste an Voraussetzungen ist keine Selbstverständlichkeit. All diese Anforderungen an einen Engine beschränkt die Auswahl beträchtlich. In der Regel sollten alle großen Game Engines diese Features implementiert haben.

Die verwendete Software in dieser Arbeit wird Unity Engine sein. Diese Engine hat alle benötigten Features, um den Anforderungen gerecht zu werden. Dazu kommt noch, dass ich bereits jahrelange Erfahrungen im Umgang mit dieser Engine habe und aus diesem Grund mit dieser vertraut bin. Allerdings ist dieser Algorithmus mithilfe jeder Umgebung implementierbar, welche den angegebenen Anforderungen gerecht wird.

Leider lässt sich mit Unity Hardware Ray Tracing auf GPUs nur mit der DirectX 12 Application Programming Interface (API) nutzen. Auch werden nur Nvidia Grafikkarten ab der Pascal Generation für dieses spezifisches Feature unterstützt. Das bedeutet, dass weder die Vulkan API noch AMD Grafikkarten im Rahmen dieser Arbeit getestet werden können. Auch muss in Unity (insofern man nicht eine eigene Render Pipeline[1] implementieren möchte) die grafisch aufwändigste Render Pipeline gewählt werden, um überhaupt die Hardware für Ray Tracing auf der GPU nutzen zu können.

[1] Eine Render Pipeline ist die Implementation welche nötig ist um Grafikkarten mit Shader Code ansprechen und nutzen zu können.

4.2 Implementation von Ray Casting

Um die definierte Herangehensweise umzusetzen, muss zunächst ein Ray Tracer implementiert werden. Die populärste Umsetzung ist von einer rekursiven Natur. Diese Methode sendet anfangs die primären Strahlen aus und bei jedem Ende eines Strahls wird wieder eine Funktion aufgerufen, welche erneut Strahlen in die Szene schießt. Dieser Prozess wird so lange wiederholt, bis die Rekursionstiefe n erreicht wurde. (Abb. 4.1) In dieser Implementation ist die Anzahl der Primär- sowie der Sekundärstrahlen pro Rekursion konfigurierbar. Darüber hinaus kann auch die Rekursionstiefe frei angepasst werden.

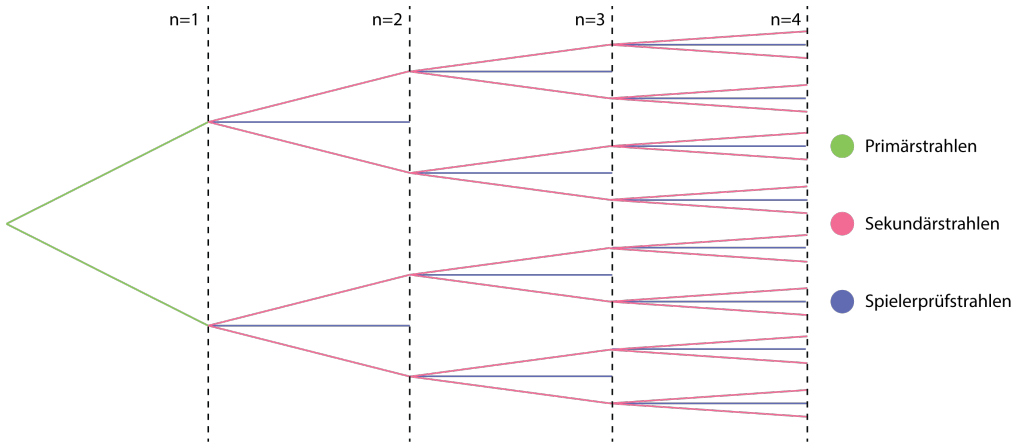


Abbildung 4.1: Darstellung der Rekursion bei 2 Primärstrahlen, 2 Sekundärstrahlen und einer Rekursionstiefe n von 4. Pro Aufruf wird ein weiterer Strahl ausgesendet, welcher testet, ob der Spieler getroffen werden kann.

Die Funktion, welche die Strahlen aussendet, berechnet dabei alle nötigen Parameter für die Strahlen und sendet auch die entsprechenden Typen an Strahlen aus. Um jedoch die Berechnung der Parameter vornehmen zu können wird noch ein Script (SoundObstacleData) benötigt, welches die Parameter der Objekte verwaltet. Trifft ein Strahl ein Objekt, so wird versucht das Script zu referenzieren und dessen Daten mit in die Berechnung einfließen zu lassen (Diffusion, Frequenzbänder). Ist dies nicht möglich, so werden standardisierte Daten verwendet.

Ein Ray wird mit einer Klasse repräsentiert (SoundRayInfo), welche alle nötigen Daten verwaltet, die von Tabelle 3.3 definiert wurden. Diese erzeugten Rays des Ray Tracers werden in drei, sogenannten Lists, gespeichert. Listen ermöglichen es auf eine einfache Weise Elemente an einer beliebigen Stelle einzufügen und zu entfernen. Darüber hinaus können komplexe Sortier- und Suchanfragen ausgeführt werden. Die Rays, welche den Spieler getroffen haben, werden wie schon erwähnt in reflektierte und gebeugte Pfade aufgeteilt. Um allerdings den gesamten Pfad visualisieren zu können, werden alle Rays, welche den Hörer verfehlt haben in der dritten, nicht relevanten Liste, gespeichert.

4.3 Implementation von Hardware Ray Tracing

Die Implementation von Hardware Ray Tracing wurde auf der Basis eines Projektes von Michal Skalsky[Ska] implementiert. Dieser Ansatz musste gewählt werden, da Unity keinerlei Dokumentation für das Erstellen von Shader Code für Ray Tracing Hardware zur Verfügung stellt. Aus diesem Grund wurde dieses Projekt genauestens hinterfragt, um ein Verständnis für den Code zu erhalten. Die Implementation benötigt dabei einen grundlegend anderen Ansatz, da die Daten auf der Grafikkarte berechnet werden. Der einzige Weg Daten von einer Grafikkarte zu erhalten, benötigt Bilder, auch Texturen genannt. Dabei wird der Ray Tracing Shader für jeden Pixel einer Textur ausgeführt. Der Shader besteht dabei aus multiplen Teilen, sogenannten Passes. Im Grunde muss ein Shader erstellt werden, welcher die Pirmärstrahlen aussendet und die Daten dieser Pfade am Ende verarbeitet und ausgibt. Dieser Shader nennt sich *Ray Generation Shader*. Um eine Berechnung durchführen zu können, wenn ein Strahl ein Objekt trifft, braucht der Shader Code des Objekts einen sogenannten *ClosestHit* Shader Pass. Dieser wird ausgeführt, wenn ein Strahl auf ein Objekt trifft. In dieser Funktion werden alle Berechnungen getätigt, welche benötigt werden, um alle relevanten Daten zu erhalten. Das heißt darin werden die Frequenzbänder, die Pfadlänge und auch die Kindstrahlen berechnet. In diesem Pass werden dabei nur die reflektierten, transmittierten und die Strahlen, welche auf den Hörer prüfen, berechnet. Diese werden dann von diesem Punkt ausgesendet und berechnen selbst wieder alle nötigen Daten und Kindstrahlen, bis die Tiefe der Rekursion erreicht ist. Ist ein Kindstrahl fertig, so werden die Daten an den Strahl, der diesen ausgesendet hat, zurückgegeben. Dort können sie erneut manipuliert oder weitergegeben werden. Dazu kommt noch, dass ebenfalls ein Pass existiert, welcher beim Verfehlen von Geometrie aufgerufen wird. Dieser Pass nennt sich *MissShader* und tätigt alle benötigten Berechnen für die Beugung von Strahlen. Auch muss noch ein separater Shader Code für das Treffen des Spielers existieren. Wird dieser getroffen, so terminiert der Pfad sofort und nimmt keine weiteren Berechnungen vor. Ist die Berechnung alle Primärstrahlen abgeschlossen, so wird der Ray Generation Shader weiter ausgeführt und gibt die Daten in Form von fünf Texturen zurück.

- Ursprung des letzten Strahls, oder des Strahls, welcher den Spieler getroffen hat in X, Y und Z.
- Richtung des letzten Strahls, oder des Strahls, welcher den Spieler getroffen hat in X, Y und Z.
- Distanz und Pfadlänge des Strahls bzw. des Pfades in Strahllänge in Meter und Pfadlänge in Meter.
- Informationen über den Pfad des Strahls. Wurde Wand getroffen, wurde Spieler getroffen, wurde transmittiert und wurde gebeugt.
- Frequenzbandinformationen mit High, HighMid, Mid und Low.

Da ein Pixel einen Strahl aussendet, erlaubt dies nur die Rückgabe eines einzigen Datensatzes. Aus diesem Grund müssen im Shader Code einige Entscheidungen getroffen werden, welche die Priorisierung von verschiedenen Pfaden definiert. Die wichtigsten Pfade sind reflektierte Pfade, welche den Spieler getroffen haben und nicht gebeugt wurden. Die spätere Transmission des Pfades spielt hierbei keine Rolle. Hat ein reflektierter Pfad den Spieler unter keinem Umstand getroffen, aber eine Geometrie, so wird ein Pfad transmittiert. Das

heißt, reflektierte Pfade sind allgemein priorisiert. Eine Transmission findet erst statt, wenn es durch Reflexion nicht möglich ist den Spieler zu treffen. Liefert diese Transmission ebenfalls kein Ergebnis, so werden trotzdem die Daten des reflektierten Pfades zurückgegeben, da dieser Pfad eventuell durch eine Beugung den Spieler getroffen hat. Wird keinerlei Geometrie getroffen, so prüft der Code, ob ein Strahl von dieser Position aus den Spieler treffen kann. ist dies der Fall, wird der Pfad erfolgreich terminiert. Alternativ wird der Pfad gebeugt. Diese Priorisierung führt dazu, dass reflektierte Ergebnisse die Mehrheit bilden. Die transmittierten Pfade repräsentieren das Mittelfeld der Priorisierung und die gebeugten Pfade werden am wenigsten priorisierten.

Wurden diese Daten von der Grafikkarte berechnet, so konvertiert der Prozessor die Daten in den Texturen in die vier Listen von reflektierten, gebeugten, transmittierten und invaliden Pfaden. Anschließend können die gleichen Funktionen zur Evaluation der Pfade verwendet werden wie von der Ray Casting Implementation, da sich die Daten in den gleichen Listen befinden.

Um allerdings überhaupt Daten auf der Grafikkarte berechnen zu können, wird noch eine sogenannte *RayTracingAccelerationStructure* benötigt. Diese Datenstruktur ist, wie schon in den Vorteilen von Hardware Ray Tracing genannt, ein BSP Tree. Diese Struktur erlaubt es die Geometrie der Objekte in den Speicher der Grafikkarte zu laden. Wird diese nicht gesetzt, so existiert die Szene nicht im Speicher der Grafikkarte und kann somit nicht berechnet werden.

4.4 Implementation von Scheduling

Da theoretisch jede Schallquelle ihre Parameter dynamisch mit Ray Tracing berechnen kann, wird ein System benötigt, welches diese Berechnungen aufteilt. Ein System in Software, welches Arbeit einteilt und verwaltet, wird Scheduler genannt. Auch ist das Scheduling für die beiden Implementationen unterschiedlich.

4.4.1 Ray Casting Scheduling

Da alle Ray Casts vom Physiksystem ausgeführt werden, ergeben sich Limitationen. In der Regel lassen sich kaum mehr als 4000 Ray Casts in diesem Algorithmus ausführen, sodass die Leistungsfähigkeit nicht stark eingeschränkt wird. Auch können die Anzahl der Casts, welche benötigt werden, anhand der Größe der Rekursionstiefe, Primär- und Sekundärstrahlen eingestellt werden. Aus diesem Grund wird das System, welches die einzelnen Quellen verwaltet, eine maximale Anzahl an Ray Casts vorweisen. Innerhalb eines Bildes werden so viele Quellen berechnet, wie das Limit erlaubt. Ein potenzielles Problem, welches sich mit einem strikten Ansatz wie diesem ergeben kann, ist, dass bestimmte Quellen nur selten berechnet werden, da sie viele Ray Casts benötigen. Um dies zu umgehen, lässt sich eine maximale Zeit pro Quelle definieren, in der sie nicht berechnet werden kann. Wird die Quelle länger als diese Zeit nicht berechnet, so wird sie im nächsten Bild priorisiert behandelt.

4.4.2 Hardware Ray Tracing Scheduling

Das Scheduling von Hardware Ray Tracing ist wesentlich einfacher. Da das Berechnen von Daten auf der Grafikkarte einiges an Overhead erzeugt, kann leider nur ein einziges Mal pro

Bild eine Quelle berechnet werden. Dies reduziert den Aufwand des Schedulings im Grunde auf eine Liste, welche anhand von Prioritäten geordnet ist. Genau wie beim Ray Casting Scheduling, kann die Sortierung durch eine maximale Zeit seit der letzten Berechnung durchgeführt werden. Zusätzlich lässt sich noch ein System einbauen, welches immer mindestens eine Quelle pro Bild berechnet, falls das System nich ausgelastet ist. Dieses System garantiert, dass eine einzige Quelle jedes Bild neu berechnet werden kann.

4.5 Audio Mixer Organisation

Die Organisation des Audiomixers muss die bis jetzt verlangten Rahmenbedingungen erfüllen. Diese definiert jeweils einzelne Mixer für alle Pfadtypen. Zusätzlich einen Echo Mixer als Kind der reflektierten Pfade. Sowie einen Halleffekt, welcher für alle Mixer gilt. Dies lässt sich mit dem in Abbildung 4.2 dargestellten Aufbau umsetzen. Da eine Audioquelle in Unity ihr Signal nur an einen einzigen Mixer weiterleiten kann, wird ein Send Mixer benötigt. Dieser sendet das unbearbeitete Signal an die restlichen vier Mixer, welche dem Mixer des Halls direkt untergeordnet sind und ist selbst dabei komplett lautlos. Die drei Mixer, welche die Frequenzbandinformationen verarbeiten, verwenden dafür jeweils vier EQs. Der Mixer des Echos besitzt alle nötigen Effekte von Kapitel 3.1.6 und sendet sein Signal an *Dry Reflection*. Alle Kind Mixer des *Reverb* senden wiederum ihre Signale an den Mixer des Halls, welcher diesen auf das gesamte Signal anwendet.

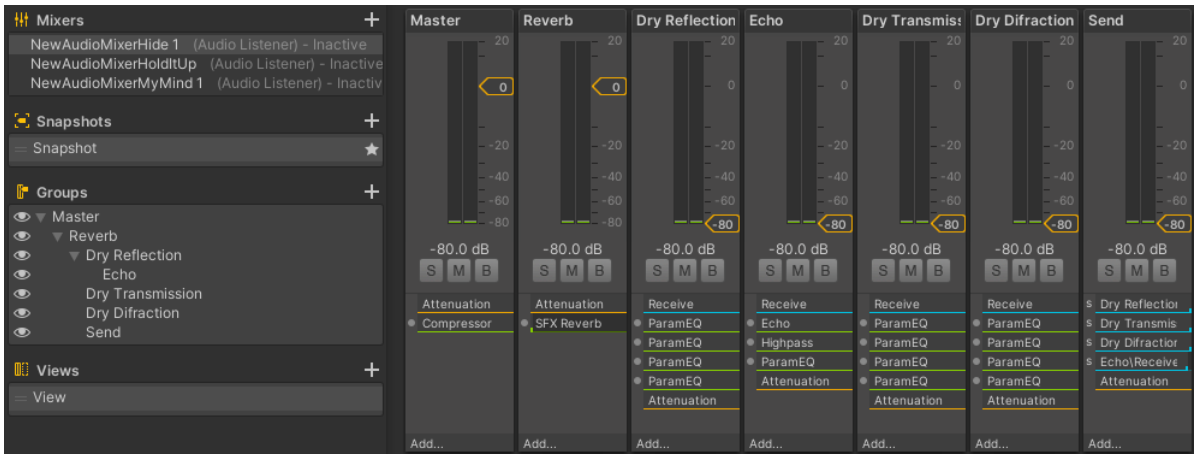


Abbildung 4.2: Audiomixer in Unity

4.5.1 Dynamic Range der Mixer

Eine Limitation des Systems ist der maximal abbildbare Unterschied der Lautstärke (Dynamic Range) in Unity. Die Mixer in Unity haben einen Wertebereich von -80dB zu +20dB. Dies stellt eine Dynamic Range von 100dB dar. Allerdings sind die letzten +20dB unbrauchbar, da diese nur zu einer Übersteuerung des Signals führen. Da Quellen im echten Leben nur in seltenen Fällen 100dB überschreiten, ist diese relativ kleine Dynamic Range kein großes Problem, aber trotzdem eine Limitation. Um diese Limitation zumindest bestmöglich zu umgehen, lässt sich die Dynamic Range künstlich mit einer Multiplikation aller Werte um einen bestimmten Faktor erhöhen. Beispielsweise lässt sich ein Faktor von 0.5 verwenden, um eine doppelt so große Dynamic Range zu erhalten. Die maximal abbildbare Lautstärke ist nun 160dB. Allerdings werden die eigentlich ziemlich lauten 80dB zu 40dB, was einer recht leisen

Unterhaltung entspricht. Aus diesem Grund ist es erforderlich, dass dieser Multiplikator nur angewendet wird, wenn Quellen in der Szene tatsächlich 80dB überschreiten. Diese Herangehensweise garantiert, dass die lautesten Quellen tatsächlich das Klangbild dominieren. Um diesen Faktor zu berechnen, muss eine Reihe der aktuellen Lautstärken, in diesem Fall 100 gespeichert werden. Befindet sich ein Wert unter diesen 100, welcher 80dB überschreitet, so wird der Faktor $f = \frac{80}{L_{\max}}$ berechnet. Ist kein einziger gespeicherter Wert größer als 80dB, so wird der Faktor auf $f = 1$ zurückgesetzt. Um die Funktionen von diesem Ansatz zu garantieren, ist es wichtig, dass jeder Lautstärkewert vor dem Setzen im Mixer immer mit dem Faktor multipliziert wird. In Abbildung 4.3 wird diese Berechnung des Faktors innerhalb der Funktion *GameConstants.CheckForMaxLoudness(splValue);* vorgenommen. *GameConstants.LoudnessMultiplicator* repräsentiert dabei den Faktor.

```
GameConstants.CheckForMaxLoudness(dirSPL);
GameConstants.CheckForMaxLoudness(transSPL);
GameConstants.CheckForMaxLoudness(diffSPL);

if(lastSPLValues.Count > 0)
    audioSource.outputAudioMixerGroup.audioMixer.SetFloat("DirectVolume", Mathf.Min(dirSPL * GameConstants.LoudnessMultiplicator - 80, 0));

if(lastTransmissionSPLValues.Count > 0)
    audioSource.outputAudioMixerGroup.audioMixer.SetFloat("TransVolume", Mathf.Min(transSPL * GameConstants.LoudnessMultiplicator - 80, 0));

if(lastDiffractionSPLValues.Count > 0)
    audioSource.outputAudioMixerGroup.audioMixer.SetFloat("DiffVolume", Mathf.Min(diffSPL * GameConstants.LoudnessMultiplicator - 80, 0));
```

Abbildung 4.3: Code welcher genutzt wird, um die drei Lautstärkewerte zu setzen.

4.6 Implementation der Berechnung der Parameter

Für die Berechnung der Parameter wurden die konkreten Implementationen in sechs separate Funktionen aufgeteilt. Die einzelnen Funktionen berechnen dabei den Dopplereffekt, Lautstärke mit Dissipation und EQ, virtuelle Position, Spatial Blend, Hall und schließlich Echo.

Diese Funktionen berechnen alle benötigten Parameter und speichern ihr Ergebnis in eine sogenannte Queue. Eine Queue ist dabei ein Datencontainer, welcher es erlaubt neue Datensätze am Ende einzufügen und am Anfang zu entfernen. Die Queue besitzt dabei nur diese Funktionalität, was sie performanter macht für dieses Szenario. Um den Aspekt des Denoisings zu implementieren, werden in den Queues die gewünschte Anzahl an Datenpunkten gespeichert. Wird der Parameter gesetzt, so werden alle aktuellen Werte in den Queues gemittelt und angewendet. Diese Mittelung kann dabei mit den gleichen performanten Befehlen ausgeführt werden wie bei Listen.

4.6.1 Lautstärke, Dissipation und EQ

All diese Berechnungen werden in einer Funktion zusammengefasst, da sie alle die Lautstärke manipulieren. Jeder dieser drei Schritte wird jeweils noch für alle drei Wege, auf die ein Pfad den Spieler treffen kann, implementiert. Allerdings gibt es einen Spezialfall für die reflektierten Pfade. Wird der Hörer direkt von der Quelle getroffen, werden die gebeugten und transmittierten Pfade nicht berechnet, da sie sonst die tiefen Frequenzen viel zu stark verstärken würden. Darüber hinaus wird bei den reflektierten Pfaden nur die Dissipation berechnet und die erhaltenen Daten der Reflexionen verworfen. Würden die Daten der Reflexionen verwendet, so findet eine Absorption von Frequenzen statt, welche zum Spieler auf direkten Weg gelangen können.

4.6.2 Wahrgenommene Position

Die wahrgenommene Position lässt sich anhand der definierten Vorgehensweise von Kapitel 3.1.2 auf eine einfache Weise im Projekt implementieren. Allerdings ist es nicht möglich für jeden einzelnen Mixer eine Positionsberechnung durchzuführen, da die Stereokanäle in Unity nicht direkt bearbeitet werden können. Auch ist es nicht möglich einen Audiostream aus dem Mixer erneut durch das Spatial Audio Modul zu leiten. Aus diesem Grund wird in dieser Arbeit immer die Position der reflektierten Pfade gewählt, insofern sie ein Volumen von -40dB überschreiten. Ist dies nicht der Fall, so werden je nachdem, welche Pfade lauter sind, die gebeugten oder die transmittierten Pfade zur Berechnung der wahrgenommenen Position verwendet. Diese gewählten Pfade werden erneut separat in einer Liste hinterlegt.

4.6.3 Spatial Blend

Genau wie die wahrgenommene Position ist die Berechnung des Spatial Blends eine direkte Übersetzung der ermittelten Formeln von Kapitel 3.1.3 in Code. Die dazu verwendeten Pfade befinden sich in der Liste, welche die Berechnung der wahrgenommenen Position hinterlegt hat. Dabei ergibt sich für den Spatial Blend die gleiche Limitation wie für die virtuelle Position. Da nur eine Position berechnet wird, kann auch nur ein einziger Spatial Blend Parameter dafür bestimmt werden. Wäre es allerdings möglich multiple Positionen zu berechnen, so bietet die Berechnung des Spatial Blends verschiedene Möglichkeiten. Einerseits könnte für jede virtuelle Quelle ein direkter Spatial Blend Wert berechnet werden, andererseits ließe sich ein ebenso guter Effekt mit der Manipulation der Stereokanäle erzielen.

4.6.4 Dopplereffekt

Die Berechnung des Dopplereffekts ist ebenfalls eine direkte Implementation der vorgestellten Formeln in Kapitel 3.1.4. Dabei ist die Berechnung dieses Effekts die einzige, welche ohne eine Queue der letzten Parameter auskommt. Dies liegt daran, dass einerseits die Berechnung äußerst simpel ist und andererseits eine Bewegung der Quelle eine sofortige Veränderung der Tonhöhe hervorruft. Bei diesem Effekt ist Reaktivität von höchster Priorität und da dies mit einer Queue nicht unbedingt gegeben ist, wird auf sie verzichtet.

4.6.5 Manipulation der Frequenzbänder

Die Implementation der Frequenzbänder stellt einen häufig verwendeten Codeabschnitt dar, welcher eine schnelle und einfache Nutzung ermöglichen soll. Dadurch ist es eine einzelne Klasse, welche viele Hilfefunktionen vereint, um mit multiplen Bänderdatensätzen schnell und einfach Rechenoperationen durchführen zu können. Auch ist die gewählte Menge der Frequenzbänder wichtig, da Anzahl der Texturkanäle für Hardware Ray Tracing auf vier begrenzt ist (Rot, Grün, Blau, Transparenz = RGBA). Das Beachten dieser Limitation ist wichtig, da die einzige Methode für den Erhalt von Daten von der Grafikkarte mit Texturen arbeitet.

Darüber hinaus muss durch die gewählte Mixer Konfiguration auf den Energieerhalt der Schallquelle geachtet werden. Da die Quelle durch drei eigenständige Audio Mixer manipuliert wird, ist dies nicht automatisch garantiert. Ist die Quelle beispielsweise gut hörbar, allerdings nicht direkt, so wird die Dissipation auch ziemlich laut sein. Dadurch haben sich die tiefen Frequenzen in ihrer Intensität verdoppelt. Um dies zu verhindern, muss der Audiomixer der reflektierten Pfade zusätzlich so angepasst werden, dass die aktuellen Werte des

Diffraktions- und Transmissions-Mixers die reflektierten Frequenzbänder nicht verfälschen. Die Formel subtrahiert dabei einfach die Frequenzen, welche durch Transmission b_{trans} und Beugung b_{diff} zum Hörer gelangen, von den reflektierten b_{refl} Pfaden. Damit die Frequenzen nicht zu stark subtrahiert werden, werden die zu subtrahierenden Frequenzbandinformation noch mit dem Verhältnis der jeweiligen Lautstärken $L_{\text{refl}}, L_{\text{trans}}, L_{\text{diff}}$ multipliziert. Die Formel wurde dabei anhand von Versuchen noch leicht angepasst und für das Erlebnis verbessert.

$$b'_{\text{refl}} = b_{\text{refl}} - \left(\frac{b_{\text{diff}} \cdot L_{\text{diff}}}{2 \cdot L_{\text{refl}}} + \frac{b_{\text{trans}} \cdot L_{\text{trans}}}{2 \cdot L_{\text{refl}}} \right) \tag{4.1}$$

4.6.6 Echo

Die Berechnung der Parameter des Echos ist eine ziemlich simple Prozedur. Im Grunde ist es erneut das einfache Umsetzen der Beschreibung in Code. Allerdings ergibt sich durch die Organisation des Audiomixers eine Besonderheit in der Berechnung der Lautstärke des Echos. Da der Mixer des Echos hierarchisch dem Mixer der reflektierten Pfade untergeordnet ist, muss die Lautstärke des Echos relativ zur Lautstärke der reflektierten Pfade berechnet werden. Um dies umzusetzen, muss lediglich die berechnete Lautstärke des Echos durch die Lautstärke der reflektierten Pfade geteilt und anschließend mit 80 multipliziert werden.

```
if (echoDelay < 10)
    echoSPL = -80;
else
    echoSPL = 80 * (maxRayDistanceDeviationSPL / lastSPLValues.Last());

EnqueueEchoValues(echoDelay, echoSPL);
```

Abbildung 4.4: Berechnung der Verzögerung und der Lautstärke des Echos.

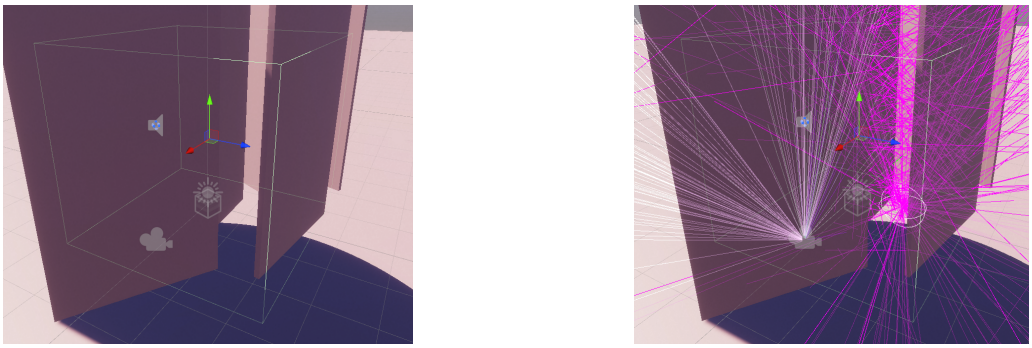
4.6.7 Hall

Die Berechnung des Volumens kann mit der beschriebenen Methode umgesetzt werden. Allerdings stellt sich die Frage, wie alle Objekte ermittelt werden können, welche das berechnete Volumen schneiden. Dafür lässt sich das Physiksystem von Unity verwenden. Mit der Berechnung des Volumens lässt sich ein Box Collider[2] definieren. Dieser Collider kann in Verbindung mit einen sogenannten Box Cast[3] verwendet werden, um alle anderen Objekte zu ermitteln, welche ebenfalls einen Collider besitzen und den gegebenen schneiden. Anhand von Abbildung 4.5 wird das Volumen des Halls visualisiert. Jede Wand, welches dieses Volumen schneidet, wird in der Berechnung berücksichtigt.

Der Halleffekt an sich, welchen Unity zur Verfügung stellt, bietet wesentlich mehr Funktionalität als nur die Zeit des Nachhalls. In diesem Effekt existiert nicht nur ein klassischer diffuser Hall mit Diffusionswert, sondern auch noch die Darstellung von früheren Reflexionen. Darüber hinaus lässt sich noch das Verhältnis der hohen zu den tiefen Frequenzen des Halls bestimmen. Auch kann der sogenannte *Decay* bestimmt werden, welcher die Dauer des Halls definiert. Da der Decay die allgemeine Zeit in Sekunden für das Verstummen des

[2]Ein Box Collider ist ein quaderförmiges Volumen, welches die Oberfläche und das Volumen eines Körpers in einer Physiksimulation definieren.

[3]Ein Box Cast ist in der Funktionsweise verwandt zu einem Ray Cast. Allerdings wird nicht eine Halbgerade für die Schnittpunktberechnung verwendet, sondern ein Quader.



(a) Hall-Volumen (b) Hall-Volumen mit Strahlen zur Visualisierung

Abbildung 4.5: Visualisierung des Hall-Volumens einmal ohne und einmal mit den Pfaden, welche das Volumen definieren.

Echos unabhängig von der Lautstärke darstellt, kann nicht einfach die errechnete Zeit für diesen Parameter verwendet werden. Zusätzlich muss die Zeit noch mit einem sechzigstel der Lautstärke des lautesten Mixers multipliziert werden. Dies verlängert die Zeit, wenn der Mixer lauter als 60dB ist und verkürzt die Zeit bei einer Lautstärke unter 60dB. Das sogenannte *Decay HF Ratio* beschreibt, wie schnell die höheren Frequenzen gegenüber den tieferen verstummen. Ein Ratio von 1 bedeutet, dass sowohl hohe, wie auch tiefe Frequenzen die gleiche Zeit zum Verstummen benötigen. Bei Werten unter eins bleiben die tieferen Frequenzen länger erhalten und bei Werten über eins die höheren. Die Berechnung des Wertes r lässt sich anhand der Formel 4.2 und den Daten der Frequenzbänder b auf eine simple Weise berechnen.

$$r_{\mathrm{HL}} = \frac{(b_{\mathrm{HighMid}} + b_{\mathrm{High}})}{(b_{\mathrm{Low}} + b_{\mathrm{Mid}})} \tag{4.2}$$

Um die Lautstärke verschiedener Frequenzbereiche separat modifizieren zu können, hat dieser Effekt nicht nur einen simplen Lautstärkewert, sondern zusätzlich noch jeweils einen weiteren für die hohen und tiefen Frequenzen. Diese drei Parameter (Room, Room LF und Room HF) sind dabei in einer unerklärten Einheit von mB gegeben. Die Dokumentation von Unity verwenden diese Einheit auf keine klare Weise. Aus diesem Grund musste für die Bestimmung der verwendeten Funktionen eine experimentelle Arbeitsweise verwendet werden. Das bedeutet, dass diese hier gezeigten Funktionen nicht auf naturwissenschaftlicher Basis erstellt wurden, sondern anhand bestimmter Charakteristiken in einem iterativen Prozess entsprechend der Resultate hergeleitet sind. In diesem Fall befindet sich der Wertebereich für alle drei Parameter zwischen -10000 und 0. Da 0 keine Verringerung der Lautstärke darstellt und -10000 die maximale, kann es sich bei der Formel nicht um eine normale SPL Berechnung handeln. Zusätzlich lässt sich von der manuellen Manipulation der Werte ableiten, dass der Einfluss der Parameter sich nicht linear verhält.

$$\begin{aligned} R &= 1 - \frac{4}{b_{\mathrm{Low}} + b_{\mathrm{Mid}} + b_{\mathrm{HighMid}} + b_{\mathrm{High}}} \\ R_{\mathrm{LF}} &= 1 - \frac{2}{b_{\mathrm{Low}} + b_{\mathrm{Mid}}} \\ R_{\mathrm{HF}} &= 1 - \frac{2}{b_{\mathrm{HighMid}} + b_{\mathrm{High}}} \end{aligned} \tag{4.3}$$

Die Diffusion des Halleffekts beschreibt, wie weich sich ein Hall anhört. Eine niedrige Diffusion sorgt dafür, dass sich der Hall stark abgehackt und fast wie ein zu schnelles Echo anhört. Je höher die Diffusion ist, desto weicher ist der Klang des Halls. Dieser Wert wird direkt anhand der durchschnittlichen Diffusion der Objekte innerhalb des Volumens des Raumes bestimmt. Dabei ist es wichtig, dass die Diffusion größerer Flächen stärker gewichtet wird wie die Diffusion kleinerer. Um dies einfach zu berechnen, lässt sich ein ähnlicher Wert zu S_a verwenden. Dieser neue Wert S_d beschreibt die Summe aller Oberflächen multipliziert mit ihren jeweiligen Diffusionen. Nur muss dieser noch zusätzlich durch die gesamte Oberfläche geteilt werden, um S_d zu normalisieren. (Formel 4.4)

$$\begin{aligned} A_t &= \sum_{m=1}^{k} A_m \\ S_d &= \frac{1}{A_t} \cdot \sum_{m=1}^{k} A_m \cdot d_m \end{aligned} \tag{4.4}$$

Zwei weitere, auf mB basierende Werte, sind die Lautstärke der frühen Reflexionen *Reflections Level*, sowie die Lautstärke des Halls *Reverb Level*. Anhand des Wertebereichs von -10000 bis 1000 und dem neutralen Wert bei 0, lässt sich auf eine logarithmische Natur der Parameter schließen. Aus diesem Grund muss die Funktion zur Berechnung der Lautstärkewerte die linearen Eingabewerte mit einem Logarithmus manipulieren. Da in dieser Arbeit die meisten logarithmischen Berechnungen eine Basis 10 verwenden, wird dieser auch in den folgenden Formeln verwendet. Da perfekte Reflexionen eher bei Räumen mit niedriger Diffusion auftreten und Hall eher bei Räumen mit hoher Diffusion, wird der Diffusionswert des Raumes in der Berechnung beachtet.

$$\begin{aligned} b_a &= \frac{b_{\text{Low}} + b_{\text{Mid}} + b_{\text{HighMid}} + b_{\text{High}}}{4} \\ L_{\text{Reve}} &= 200 \cdot \log_{10}\left(b_a \cdot S_d\right) + 20 \\ L_{\text{Refl}} &= 200 \cdot \log_{10}\left(b_a \cdot (1 - S_d)\right) + 20 \end{aligned} \tag{4.5}$$

Der letzte Wert, welcher letztendlich die Stärke des Halleffekts definiert, ist der *Wet* Parameter. Dieser stellt die Lautstärke des Halls in einem Bereich von -80dB bis 0dB dar. Dies ist zwar eine bekannte Einheit, allerdings wird sie relativ zu einem eintreffenden Signal, welches schon von der Lautstärke definiert ist, verwendet. Aus diesem Grund lässt sich in diesem Fall ebenfalls keine Formel aus der Wissenschaft herleiten. Trotzdem wurde in dieser Formel auf die Berechnung eines Dezibel Wertes geachtet und ein logarithmisches Verfahren definiert. Da der Halleffekt mit hohen RT_{60} Zeiten allgemein ziemlich laut wirkt, wird auch dieser Wert in der Berechnung des Wet Wertes verwendet. Auch können sehr hohe RT_{60} Zeiten einen offenen Raum signalisieren, da das Volumen im Vergleich zur Oberfläche dafür um einige Magnituden größer sein muss. Zusätzlich zu der Zeit für das Verstummen werden auch noch die Frequenzbandinformationen verwendet.

$$w = 20 \cdot \log\left(b_a \cdot 10^{-0.005 \cdot RT_{60}}\right) \tag{4.6}$$

4.7 Materialsystem

Werden keine Texturen für die Bestimmung der Absorption und der Diffusion verwendet, so hängen diese Werte nur von fünf Parametern ab. Da diese grundsätzlich vom Material abhängig sind, lässt sich die Benutzung des Systems stark vereinfachen. Mit der Implementation eines Materialsystems können generische Materialien wie Holz oder Beton auf eine simple Weise selektiert werden. Dabei kann der Nutzer ein bestimmtes Material auswählen und die benötigten Daten werden automatisch geladen. Die in dieser Arbeit verwendeten Absorptionskoeffizienten wurden bei [aco] nachgesehen. Die Absorption des High Bands bei 10 kHz wurde dabei grob extrapoliert. Weil für die Diffusion in diesem Anwendungsfall keine echten Daten existieren, wurden sie grob abgeschätzt (kleine Werte bedeuten mehr Reflexion, größere mehr Scattering). Da auch eine Schallgeschwindigkeit für die Berechnung der Transmission benötigt wird, ist auch diese für jedes Material hinterlegt.

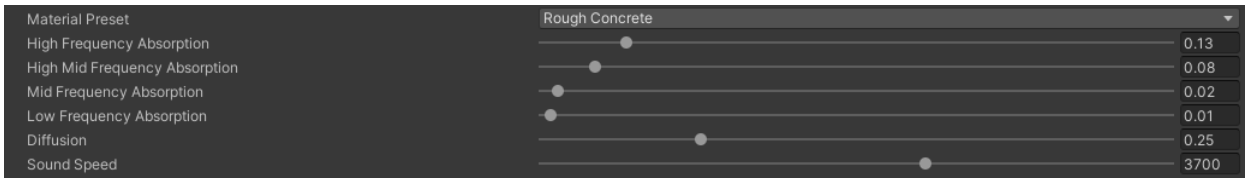


Abbildung 4.6: Darstellung des Materialsystems. Beim Wählen eines Materials werden alle zusätzlich gezeigten Parameter anhand des Materials gesetzt.

5 Leistungsvergleich

Durch die multiplen Implementationen des Algorithmus muss ein Vergleich beider Umsetzungen durchgeführt werden. In diesem Vergleich werden beide Implementationen auf zwei verschiedenen Testsystemen getestet und anschließend analysiert. Dabei hat das eine Testsystem eine Grafikkarte, welche Ray Tracing Hardware integriert hat und ein Weiteres ohne eine solche Karte. Allerdings kann bei dem zweiten System die gleiche Implementation auf der GPU ausgeführt werden. Damit lässt sich grob der Leistungsunterschied von Software Ray Tracing und Hardware Ray Tracing auf einer Grafikkarte feststellen.

5.1 Bestimmung der Testparameter

Um beide Implementationen gegenüberstellen zu können, werden einige Parameter für diesen Vergleich benötigt. Für den Test der Leistungsfähigkeit beider Implementationen muss ein Zeitparameter für die Laufzeit des Algorithmus definiert werden. Zusätzlich zur Laufzeit wird auch noch die Stabilität der Parameter evaluiert. Um die Stabilität, in dieser Arbeit Jitter genannt, aller berechneten Werte zu bestimmen, darf der Jitter beider Implementationen nur mit der gleichen Laufzeit bestimmt werden. Um die Laufzeit jeder Implementation zu evaluieren, wird der Zeitpunkt direkt vor dem Beginn der Ausführung und der Zeitpunkt direkt nach dem Durchlauf des Codes festgehalten. Anhand dieser beiden Zeitpunkte lässt sich die Zeitspanne, welche für die Ausführung benötigt wurde, errechnen. Ein Teil des Codes, welcher in die Analyse mit einfließt, ist die Berechnung der Parameter der Effekte und das Setzen dieser Werte. Diese Funktionen werden dabei von beiden Implementationen verwendet. Trotzdem ist es wichtig, dass diese Funktionen in die Laufzeit einspielen, da beide Implementationen verschiedene Mengen an Daten erzeugen.

Um im Nachhinein noch eine genauere Leistungsanalyse vornehmen zu können, werden die einzelnen Schritte ebenfalls noch separat gemessen und analysiert. So besteht beispielsweise die Ray Casting Implementation auf der CPU aus zwei separierbaren Teilen. Der Simulation der Strahlen und der Analyse der erzeugten Daten, um die Parameter zu bestimmen. Bei Hardware Ray Tracing muss noch ein zusätzlicher Schritt dazwischen vorgenommen werden. Da die Berechnung der Simulation auf der Grafikkarte stattfindet, muss noch ein Teil des Codes der Hardware Ray Tracing Implementation die Konvertierung von Pixel Daten in Schall Strahl Daten vornehmen. Durch diesen Unterschied bringt eine separate Analyse der Bestandteile ein vertieftes Verständnis für die Zusammenarbeit der einzelnen Teile.

5.2 Laufzeitanalyse

Um die Laufzeitanalyse durchführen zu können, muss eine Methode definiert werden. Da weitere Tests auf den Ergebnissen dieser aufbauen, ist es wichtig, dass die Vorgehensweise solide und verlässliche Resultate ergibt. Aus diesem Grund werden jeweils für beide Implementationen zehn Datensätze gesammelt und gemittelt. Diese Datensätze bestehen wiederum aus jeweils 1100 einzelnen Werten. Dabei wurden die Einstellungen der Ray Tracing Algorithmen so getroffen, dass die Laufzeit der Ray Casting Implementation mit der Laufzeit der

Hardware Ray Tracing Variante bestmöglich übereinstimmt. Diese ermittelten Werte bilden die Basis für die Berechnung der Parameterstabilität.

	Ray Casting	Hardware Ray Tracing
Primärstrahlen	65	529
Rekursionstiefe	5	16
1. Lauf	6.694 ms	6.367 ms
2. Lauf	6.713 ms	6.712 ms
3. Lauf	6.639 ms	6.736 ms
4. Lauf	6.732 ms	6.296 ms
5. Lauf	6.709 ms	6.597 ms
6. Lauf	6.652 ms	6.430 ms
7. Lauf	6.635 ms	6.389 ms
8. Lauf	6.716 ms	6.621 ms
9. Lauf	6.674 ms	6.931 ms
10. Lauf	6.678 ms	6.467 ms
Durchschnitt	6.684 ms	6.555 ms

Tabelle 5.1: Vergleich der Laufzeiten beider Algorithmen im ersten Testsystem. (AMD Ryzen 3700X, Nvidia GTX 1070)

Das erste Testsystem repräsentiert dabei ein modernes allerdings nicht topaktuelles System. Die Generation der Grafikkarte kann die Hardware Ray Tracing Implementation zwar ausführen, allerdings besitzt die GPU keine dedizierte Hardware dafür. Dieses System wurde als Referenzsystem gewählt, da es das Hauptsystem darstellt, auf dem diese Arbeit erstellt wurde. Für diese Arbeit wurde ein System bereitgestellt, welches eine Grafikkarte mit spezialisierter Ray Tracing Hardware vorweisen kann. Dadurch auch dieses System analysiert werden. (Tabelle 5.2)

	Ray Casting	Hardware Ray Tracing
Primärstrahlen	65	529
Rekursionstiefe	5	16
1. Lauf	5.692 ms	5.561 ms
2. Lauf	5.721 ms	5.588 ms
3. Lauf	5.689 ms	5.599 ms
4. Lauf	5.644 ms	5.519 ms
5. Lauf	5.616 ms	5.492 ms
6. Lauf	5.687 ms	5.652 ms
7. Lauf	5.705 ms	5.661 ms
8. Lauf	5.721 ms	5.599 ms
9. Lauf	5.639 ms	5.497 ms
10. Lauf	5.649 ms	5.498 ms
Durchschnitt	5.676 ms	5.567 ms

Tabelle 5.2: Vergleich der Laufzeiten beider Algorithmen im zweiten Testsystem. (Intel Core i7 9700K, Nvidia RTX 2080)

Diese Analyse ermöglicht es eine grobe Aussage treffen zu können, ob die dedizierte Hardware die Laufzeit verbessern kann. Um einen direkten Vergleich der beiden Systeme zu berech-

nen, lässt sich ein Faktor des Unterschieds anhand der Durchschnitte der Laufzeiten beider Implementationen errechnen.

	Ray Casting	Hardware Ray Tracing
Erstes System	6.684 ms	6.555 ms
Zweites System	5.676 ms	5.567 ms
Leistungsfaktor	1.17756	1.17749

Tabelle 5.3: Der Leistungsfaktor beschreibt, wie viel stärker das zweite Testsystem im Vergleich zum Ersten ist. 1.177 = 117.7%

Von Tabelle 5.3 lässt sich ablesen, dass beide Implementationen auf dem zweiten System um 17.7% schneller ablaufen. Dieses Ergebnis bedeutet, dass die Laufzeit beider Implementationen mit der Hardware gleichmäßig skalieren. Darüber hinaus heißt dies, dass die Verwendung von dedizierter RTX Hardware, zumindest bei 529 Primärstrahlen, keinen messbaren Einfluss auf das Ergebnis hat. Um die aufwändigsten Rechenschritte bestimmen zu können, kann die globale Laufzeit noch genauer gemessen werden. Tabellen 5.4 und 5.5 stellen diese separaten Daten dar und berechnen den prozentualen Anteil, den jeder einzelne Rechenschritt von der Gesamtzeit jeder Implementation benötigt.

1. System	RC Trace	RC Eval		HRT Trace	HRT Konversion	HRT Eval
1. Lauf	6.367 ms	0.182 ms		0.062 ms	6.391 ms	0.203 ms
2. Lauf	6.385 ms	0.178 ms		0.060 ms	6.141 ms	0.195 ms
3. Lauf	6.309 ms	0.191 ms		0.060 ms	6.143 ms	0.196 ms
4. Lauf	6.342 ms	0.184 ms		0.061 ms	6.225 ms	0.235 ms
5. Lauf	6.372 ms	0.181 ms		0.063 ms	6.144 ms	0.221 ms
Durchschnitt	6.355 ms	0.183 ms		0.061 ms	6.209 ms	0.210 ms
Anteil in Prozent	97.20%	2.80%		0.94%	95.82%	3.24%

Tabelle 5.4: Darstellung der einzelnen Schritte in beiden Implementationen für das erste Testsystem. RC = Ray Casting, HRT = Hardware Ray Tracing, Eval = Evaluation

2. System	RC Trace	RC Eval		HRT Trace	HRT Konversion	HRT Eval
1. Lauf	5.566 ms	0.167 ms		0.057 ms	5.670 ms	0.183 ms
2. Lauf	5.645 ms	0.164 ms		0.055 ms	5.207 ms	0.187 ms
3. Lauf	5.658 ms	0.167 ms		0.052 ms	5.738 ms	0.183 ms
4. Lauf	5.699 ms	0.169 ms		0.056 ms	5.177 ms	0.182 ms
5. Lauf	5.677 ms	0.167 ms		0.056 ms	5.285 ms	0.192 ms
Durchschnitt	5.649 ms	0.167 ms		0.055 ms	5.415 ms	0.185 ms
Anteil in Prozent	97.13%	2.87%		0.98%	95.74%	3.28%

Tabelle 5.5: Darstellung der einzelnen Schritte in beiden Implementationen für das zweite Testsystem. RC = Ray Casting, HRT = Hardware Ray Tracing, Eval = Evaluation

Anhand der Ergebnisse beider Systeme lässt sich ablesen, dass beim Ray Casting auf der CPU der aufwändige Teil die Simulation der Strahlen ist. Die Evaluation der erzeugten Daten benötigt dagegen im Vergleich fast keine Laufzeit. Ein gänzlich anderes Ergebnis hat sich

beim Hardware Ray Tracing auf den GPUs ergeben. Der größte Aufwand hierbei liegt bei der Übersetzung (Konversion) der Daten von der Grafikkarte auf den Prozessor. Die Berechnung der Strahlen auf der Grafikkarte benötigt nur ca. 1% der Laufzeit. Der zeitliche Aufwand der Evaluation ist dabei vergleichbar gering zur Ray Casting Implementation. Auch verstärken diese Ergebnisse die Vermutung, dass die Ray Tracing Hardware im zweiten System keinen messbaren Einfluss auf das Ergebnis nimmt, da die relativen Ergebnisse beider Systeme annähernd identisch sind.

Um zum Abschluss noch ein Verständnis für die zusätzliche Last der Software durch die Berechnung von Schall zu bestimmen, werden von jeder Implementation die Zeit für die Ausführung pro Bild gemessen und die Zeit ohne die Ausführung der Berechnungen. Diese Analyse ist wichtig, da die Laufzeiten nicht die Zeit für die Berechnung des Bildes widerspiegeln.

	Keine Berechnung	Ray Casting	Hardware Ray Tracing
Primärstrahlen	0	65	529
Rekursionstiefe	0	5	16
1. Lauf	5.415 ms	8.396 ms	8.476 ms
2. Lauf	5.653 ms	8.358 ms	8.214 ms
3. Lauf	5.401 ms	8.315 ms	8.623 ms
4. Lauf	5.409 ms	8.364 ms	8.344 ms
5. Lauf	5.584 ms	8.463 ms	8.442 ms
Durchschnitt	5.492 ms	8.379 ms	8.420 ms

Tabelle 5.6: Vergleich der Laufzeiten pro Bild beider und keinem Algorithmus auf dem ersten Testsystem.

Die Ergebnisse von Tabelle 5.6 zeigen, dass beide Algorithmen annähernd die gleiche Zeit für die Ausführung benötigen. Darüber hinaus erhöht sich die benötigte Zeit nur um ca. 3 Millisekunden, wenn eine der Implementationen ausgeführt wird. Diese Ergebnisse erlauben es, dass die Software immer noch mit mehr als 100 Bildern pro Sekunde ausgeführt werden kann, was in der heutigen Zeit von Spielen in der Regel verlangt wird.

5.3 Ressourcenverbrauch

Da die Laufzeit die Grafikkarten nicht differenziert, kann noch eine weitere Analyse durchgeführt werden, um den Ressourcenverbrauch beider Grafikkarten zu bestimmen.

	GTX System	RTX System
GPU Nutzung ohne Berechnung	75%	48%
GPU Nutzung mit Berechnung	92%	54%
Zusätzliche Nutzung Verhältnis	0.227	0.125

Tabelle 5.7: Vergleich des Ressourcenverbrauchs vom ersten zum zweiten Testsystem. Auflösung 1280x720, 24964 Primärstrahlen, Messung mit MSI Afterburner

Anhand der Ergebnisse von 5.7 kann errechnet werden, dass das RTX System im Vergleich ca. nur halb so viel Ressourcen mehr verbraucht für die gleiche Zunahme an Last, wie das GTX System. Dabei ist es wichtig, dass der allgemeine Leistungsunterschied beider Karten

zueinander beachtet wird. Eine RTX 2080 ist im Durchschnitt 20-50% stärker als eine GTX 1070 [tec]. Allerdings beschleunigt die RTX 2080 die Berechnung der Rays 81.6% effizienter als die GTX 1070. Daran lässt sich ableiten, dass die Ray Tracing Hardware tatsächlich die allgemeine Last der RTX Grafikkarte verringert bei der gleichen Berechnung.

5.4 Parameterstabilität

Ein weiterer wichtiger Aspekt der Implementationen ist die Stabilität der berechneten Werte. Dabei wird die Stabilität anhand eines Jitters bestimmt. Der Jitter eines Parameters wird als die maximale Abweichung vom Durchschnitt eines Datensatzes definiert. Der verwendete Datensatz beinhaltet dabei alle Parameter, welche in einer Zeit von einer Minute gesetzt wurden. Um Stärken und Schwächen der einzelnen Implementationen bestimmen zu können, sind zwei verschiedene Konfigurationen der Szene erstellt und gemessen worden. Die erste Konfiguration bietet eine gute Möglichkeit für die Messung der Parameter des Halls und des Echos. Die Begründung hierfür liegt darin, dass die Szene einem großen Raum ähnelt.

Parameter	Ray Casting	Hardware RT	Einheit
Direkte Lautstärke	1.318	1.312	[-80, 20] in dB
Transmission Lautstärke	Nicht Impl.	25.295	[-80, 20] in dB
Diffraktion Lautstärke	15.806	Nicht valide	[-80, 20] in dB
Direkte Frequenzbänder	0.013	0.078	Distanz eines 4D Vektors
Transmission Frequenzbänder	Nicht Impl.	0.016	Distanz eines 4D Vektors
Diffraktion Frequenzbänder	0.042	Nicht valide	Distanz eines 4D Vektors
Wahrgenommene Position	1.462	0.694	Distanz in m
Spatial Blend	0.166	0.139	[0, 1]
Volumen Zentrum	1.444	10.003	Distanz in m
Volumen	53936.650	91388.160	Volumen in m^3
Echo Lautstärke	4.358	1.542	[-80, 20] Lautstärke in dB
Echo Verzögerung	14.104	15.994	Zeit in ms
Hall Low Decay	8.770	25.370	Zeit in ms
Hall High Low Ratio	0	0	[0, 2]
Hall Low Room Level	0.007	0.028	[-10000, 0] in mB
Hall High Room Level	0.014	0.083	[-10000, 0] in mB
Hall Room Level	0.006	0.046	[-10000, 0] in mB
Hall Diffusion	0	0	Prozent
Hall Reverb	0.942	6.017	[-1000, 2000] in mB
Hall Reflections	0.942	6.017	[-10000, 1000] in mB
Hall Wet	1.646	5.059	[-80, 0] in dB

Tabelle 5.8: Vergleich der Stabilität der einzelnen Parameter (Jitter) bei gleicher globaler Laufzeit. Die verwendete Szene, Position des Spielers und Quellenposition werden in Abbildung 5.1 dargestellt.

Im ersten Test von Hardware Ray Tracing kann die Lautstärke der Diffraktion leider nicht bestimmt werden, da die Transmission lauter ist und somit die Lautstärke der Diffraktion auf 0 setzt. Aus dem gleichen Grund sind auch die Frequenzbänder der Diffraktion nicht valide. Die Ergebnisse der Lautstärke sind bei beiden Implementationen von vergleichbarer Güte.

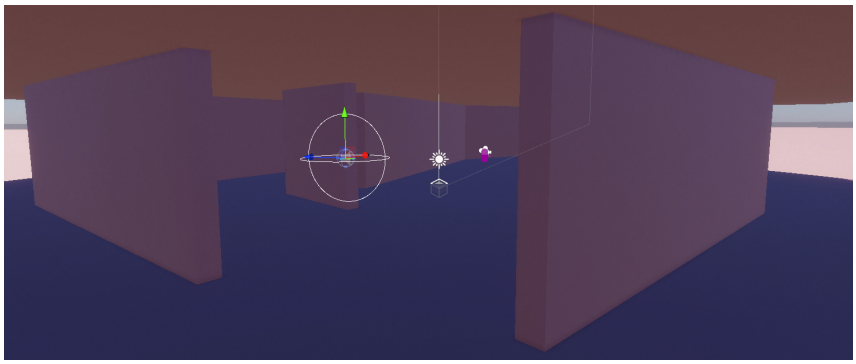


Abbildung 5.1: Verwendete Konfiguration der Szene für die Bestimmung der Jitter Parameter von Tabelle 5.8. Der weiße Kreis stellt die Schallquelle dar und die pinke Kapsel den Spieler.

Die Frequenzbandinformationen sind beim Hardware Ray Tracing zwar um einiges instabiler, allerdings sind beide Werte im Allgemeinen äußerst stabil. Die wahrgenommene Position und der Spatial Blend sind bei beiden Implementationen vergleichbar. Die berechneten Werte des Halls sind zwar beim Ray Casting generell stabiler, allerdings ist der Wertebereich der Parameter so groß, dass der Jitter immer noch ziemlich gering ist.

Die zweite Szene kann verwendet werden, um eine ebene Fläche in einer Stadt widerzuspiegeln. Das Volumen des Halls ist dabei im Vergleich zur Oberfläche so groß, dass die Zeit für das Verstummen des Nachhalls viel zu lang, und somit der Wet Parameter des Halls äußerst klein ist. Aus diesem Grund kann anhand von der zweiten Szenerie die Stabilität des Halls kaum evaluiert werden. Die Verzögerung des Echos ist ein Wert, welcher sich meistens in einem Wertebereich von 100 - 1000ms befindet. Aus diesem Grund ist es wichtig, dass die Lautstärke des Echos stabiler ist als die Verzögerung.

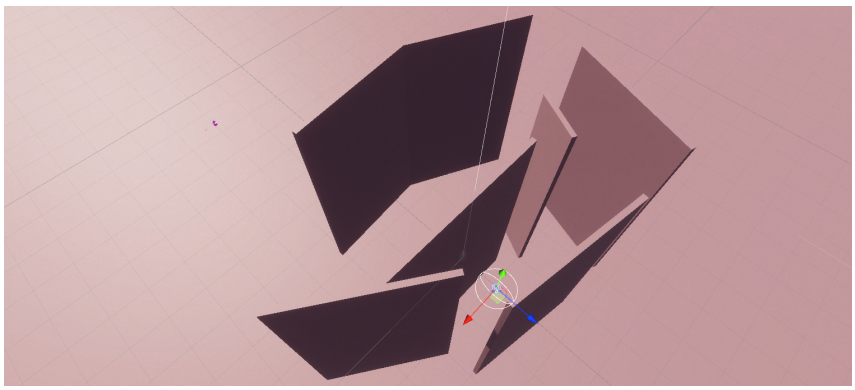


Abbildung 5.2: Verwendete Konfiguration der Szene für die Bestimmung der Jitter Parameter von Tabelle 5.9. Der weiße Kreis stellt die Schallquelle dar und die pinke Kapsel den Spieler.

Wird der Hall im Test der zweiten Szenerie vernachlässigt, bieten erneut beide Implementationen eine kompetente Lösung für die Berechnung der Parameter. Jedoch sind alle Lautstärke- und Frequenzbandberechnungen bei der Hardware Ray Tracing Implementation stabiler. Das Echo ist dabei konsistenter bei der Ray Casting Implementation. Wird andererseits der Hall betrachtet, sind die Werte der Ray Casting Implementation um Längen beständiger als die der Hardware Ray Tracing Implementation.

Parameter	Ray Casting	Hardware RT	Einheit
Direkte Lautstärke	5.604	4.017	[-80, 20] in dB
Transmission Lautstärke	Nicht Impl.	Nicht valide	[-80, 20] in dB
Diffraktion Lautstärke	17.874	6.840	[-80, 20] in dB
Direkte Frequenzbänder	0.280	0.210	Distanz eines 4D Vektors
Transmission Frequenzbänder	Nicht Impl.	Nicht valide	Distanz eines 4D Vektors
Diffraktion Frequenzbänder	0.052	0.038	Distanz eines 4D Vektors
Wahrgenommene Position	6.427	9.154	Distanz in m
Spatial Blend	0.049	0.227	[0, 1]
Volumen Zentrum	5.061	92.946	Distanz in m
Volumen	343390.600	2571938.000	Volumen in m^3
Echo Lautstärke	30.660	23.798	[-80, 20] Lautstärke in dB
Echo Verzögerung	26.908	54.958	Zeit in ms
Hall Low Decay	56.600	615.130	Zeit in ms
Hall High Low Ratio	0	0	[0, 2]
Hall Low Room Level	0.062	6.590	[-10000, 0] in mB
Hall High Room Level	0.061	1.738	[-10000, 0] in mB
Hall Room Level	0.026	2.485	[-10000, 0] in mB
Hall Diffusion	0	0	Prozent
Hall Reverb	4.371	77.572	[-1000, 2000] in mB
Hall Reflections	4.371	77.572	[-10000, 1000] in mB
Hall Wet	12.354	137.910	[-80, 0] in dB

Tabelle 5.9: Vergleich der Stabilität der einzelnen Parameter (Jitter) bei gleicher globaler Laufzeit. Die verwendete Szene, Position des Spielers und Quellenposition werden in Abbildung 5.2 dargestellt.

Insgesamt betrachtet, lässt sich anhand der Tests ableiten, dass die Ray Casting Implementation für die Berechnung von Hall eine bessere Lösung bietet. Für die restlichen Berechnungen sind beide Implementationen äußerst gut geeignet und unterscheiden sich dabei nur minimal in den Ergebnissen. Darüber hinaus lässt sich erkennen, dass bei komplexen Pfaden die Hardware Ray Tracing stabilere Werte berechnet. Dies ist zu erwarten bei der dreimal höheren Rekursionstiefe dieser Implementation im Vergleich zum Ray Casting.

5.5 Menge der Parameter

Die Menge der benötigten Parameter, um eine Szene für den Algorithmus dieser Arbeit vorzubereiten ist ziemlich gering. Jedes Objekt, welches die Berechnungen beeinflussen soll, benötigt lediglich einen Collider und ein Mesh. Diese beiden Komponenten werden sowieso für die korrekte Repräsentation des Objekts benötigt. Zusätzlich müssen diese Körper nur noch ein Script besitzen, welches komplett mit vorgefertigten Materialien eingestellt werden kann. Die benötigte Mixer Struktur kann einfach kopiert und damit wieder verwendet werden. Das Aufsetzen der Stadt Szene in der Software hat lediglich ca. 5 Minuten benötigt. Der Workflow von Wwise ist zwar schnell, allerdings ist es unrealistisch, dass die komplette Parametrisierung so schnell durchgeführt werden kann. Auch das Fehlen von dedizierten Räumen und Portalen kann als Erleichterung für den Workflow gesehen werden.

6 Fazit

Das Fazit dieser Arbeit wird unter dem Einfluss multipler Aspekte stehen. Zum einen müssen die Implementationen vom Feature Set her gegen Wwise verglichen werden, zum anderen muss auch eine Empfehlung für die Art der Implementation gegeben werden. Auch muss eine Voraussicht für beide Implementationen gegeben werden, um potenzielle Verbesserungen hervorheben zu können.

6.1 Realisierte Features

Da sich beide Umsetzungen auf verschiedene Weise nutzen lassen und weil beide ein leicht verschiedenes Feature Set haben, müssen beide Implementationen getrennt betrachtet werden.

6.1.1 Ray Casting Implementation

Da die allgemeine Methode, welche in dieser Arbeit definiert wurde, versucht alle Anforderungen von 2.4.3 zu erfüllen, sind die wichtigsten dieser Forderungen in beiden Implementationen vorhanden. Allerdings kann bei dieser Umsetzung keine Transmission berechnet werden. Dazu kommt noch, dass der fünfte Punkt per Design gar nicht umgesetzt ist.

Die Fähigkeit des Algorithmus, in einem einzigen Bild alle Daten zu erstellen und zu verarbeiten, erfüllt die ersten beiden Anforderungen. Einschließlich der dynamischen Position des Spielers und der Quelle. Auch kann die Geometrie beliebig verändert werden, während die Software ausgeführt wird.

Durch die Abwesenheit von Portalen ist der zeitlich benötigte Aufwand für die Parametrisierung der Szene wesentlich geringer. Auch müssen keine Räume explizit definiert werden, um die Implementation lauffähig zu machen. Die gesamte Parametrisierung der Quelle beschränkt sich dabei auf die Lautstärke, Isometrie, Einstellungen des Ray Tracers und die Konfiguration für die Evaluation der Daten. Dabei können die letzten beiden Einstellungsmöglichkeiten häufig bei den Standardeinstellungen belassen werden. Passt man diese Parameter an, so lässt sich beispielsweise die Laufzeit des Ray Tracers für recht leise Quellen verringern. Darüber hinaus kann bei Objekten eine Datenbank für die gegebenen akustischen Eigenschaften eingebunden werden, sodass der Nutzer einfach nur das Materialsystem von 4.7 verwenden kann. Sollte der Nutzer allerdings dieses System nicht nutzen, müssen nur fünf Parameter für die Konfiguration eines Objekts gesetzt werden. Vier Absorptionswerte und ein Diffusionswert. Ein Aspekt, welcher leider im Rahmen dieser Arbeit nicht automatisiert werden konnte, ist die Erstellung der benötigten Struktur im Audio Mixer. Diese Arbeit ist definitiv der größte Aufwand, welcher mit dem entwickelten System anfallen kann.

Mit der Ray Casting Version des Algorithmus, lässt sich leider keine Transmission darstellen. Allerdings bietet die grobe Approximation der Diffraktion ein zufriedenstellendes Ergebnis. Frequenzen im tiefen Bereich des Spektrums werden wie erwartet bei der Bewegung um eine Ecke beibehalten. Dieses Feature dürfte mit dem geringen Aufwand der Parametrisierung

der größte Erfolg dieser Methode sein.

Allgemein stellt die Ray Casting Implementation eine solide Basis für die dynamische Berechnung von Schall dar.

6.1.2 Hardware Ray Tracing Implementation

Da die beiden einzigen Features, welche bei der Ray Casting Version vermisst werden, per Design in dieser Umsetzung vorhanden sind, stellt diese Implementation die umfangreichste dar.

Wie auch bei der Ray Casting Version, spielt es für die Berechnung der Parameter keine Rolle, ob die Geometrie oder die Quelle statisch oder dynamisch sind.

Der zeitlich benötigte Aufwand für die Parametrisierung ist mit dieser Version vergleichbar zur Ray Casting Implementation. Die Einstellungen des Ray Tracers können jedoch nicht mehr alle frei gewählt werden. Nur die Anzahl der Primärstrahlen ist noch frei konfigurierbar. Beim Betrachten der anderen Parameter benötigt die Ray Tracing Version einen zusätzlichen Mixer, eine Schallgeschwindigkeit für jedes Material und gegebenenfalls Texturen für die Absorption und Diffusion eines Objektes. Dabei stellt diese Möglichkeit, Texturen für die genannten Parameter zu verwenden, ein zusätzliches Feature dar. Ein weiterer Vorteil von Hardware Ray Tracing ist die verwendete Architektur der Shader. Dafür müssen für die Umsetzung von Hardware Ray Tracing multiple Shader implementiert werden. Diese Arbeit benutzt für die Umsetzung drei verschiedene Shader. Um spezielle Verhaltensweisen von Schall zu berechnen, lassen sich multiple Shader Varianten erstellen.

Mit der Verwendung der GPU konnte die Berechnung von Transmission realisiert werden. Die Ergebnisse der Analyse sind dabei, genau wie die Diffraktion, zufriedenstellend. Die berechneten Werte würden allerdings durchaus von einer höheren Stabilität profitieren. Die Addition von Transmission zur bereits vorhandenen Diffraktion hilft dabei, das Ergebnis der kontinuierlichen Schallausbreitung zu verbessern. Beispielsweise sind geschlossene Türen und dünne Wände keine festen Grenzen mehr, sondern nun durchdringbar für virtuelle Schallwellen. Somit lassen sich Geräusche in komplett getrennten Räumen zumindest annähernd realistisch berechnen.

6.2 Aussicht

6.2.1 Allgemeine Aussicht

Vorberechnung

Um die Laufzeit beider Implementationen zu optimieren, könnte ein Bake-Prozess für statische Quellen implementiert werden. In 2.4 wurde zwar erwähnt, dass der Prozess des Bakings gerade versucht wird zu minimieren, allerdings würde es den Rechenaufwand in der Runtime[1] auf beinahe null reduzieren. Leider würde dies die Vorteile von dynamischer Geometrie wieder eliminieren. Eine hybride Möglichkeit könnte sich durch einen Vorberechnungsschritt definieren lassen, welcher relevante Pfade findet und speichert. In der Laufzeit werden dann

[1] Die Runtime beschreibt die Zeit während der Ausführung eines Programms.

die letzten Rays der vorberechneten Pfade getestet, anstelle des ganzen Pfades. Darüber hinaus kann die Implementation von Baking dazu verwendet werden, um den Rechenaufwand für statische Quellen drastisch zu reduzieren. Die Vorberechnung von statische Quellen führt dazu, dass in der Laufzeit mehr Ressourcen für die Berechnung von dynamischen Quellen zur Verfügung stehen.

Metropolis Sampling Methode

Da die Pfade von Schall durch die potenzielle Diffraktion von einer äußerst komplexen Natur sein können, würde eine Umsetzung des Ray Tracers mit der Metropolis Sampling Methode wahrscheinlich die Laufzeit und die Ergebnisse verbessern. Diese Methode basiert dabei auf der Mutation von bereits validen Pfaden. Wurden bei dieser Methode valide Pfade gefunden, so werden diese mit der Addition, oder Änderung von Rays mutiert. Diese neuen Pfade werden daraufhin auf ihre Validität getestet und gegebenenfalls gespeichert. [vgl. Med] Diese Methode könnte die Laufzeit stark reduzieren, da bereits valide Pfade mit diesem Ansatz einfach nur optimiert und vermehrt werden.

6.2.2 Weitere Aussicht für Ray Casting

Für die Ray Casting Implementation wurde die Physik Engine verwendet. Dieser Ansatz wurde gewählt, sodass eine schnelle Programmierung des Algorithmus möglich ist. Um eine auf der CPU laufende Implementation zu erstellen, sollte unbedingt eine eigene Ray Tracing Engine programmiert werden, welche auf einem Hintergrund-Thread[2] arbeiten kann. Dieser Ansatz würde es erlauben, den größten Teil der Laufzeit (97%) auf dem Haupt-Thread zu eliminieren. Dazu kommt noch, dass eine eigene Implementation mit den benötigten Beschleunigungsstrukturen implementiert werden kann, um die Leistung weiter zu erhöhen. Auch wird somit die Physik Engine des Spiels nicht mit Schallberechnungen allokiert. Darüber hinaus kann mit einer eigenen Implementation die Leistung vermutlich noch stark gesteigert werden. Auch kann es skalierbar für multiple Threads programmiert werden. Den Ansatz eines eigenen Ray Tracing Engines auf der CPU hat Audiokinetic für Wwise gewählt.

6.2.3 Nutzung mit anderen Audio Engines

Die in dieser Arbeit erstellten Implementationen berechnen eigenständige Parameter. Diese Abgrenzung ermöglicht es, die berechneten Werte an andere Engines weiterzugeben. Beispielsweise lassen sich somit die Berechnung der virtuellen Position an Wwise weitergeben. Dieser Ansatz ermöglicht es andere umfangreiche Engines selbständig zu erweitern.

[2]Ein Thread ist ein separierter Ablauf von Code innerhalb des gleichen Programms. Ein Spiel hat dabei meistens einen Haupt-Thread, Render-Thread, Audio-Thread und ein Job System, welches einzelne Aufgaben auf multiple Threads verteilt.

6.2.4 Weitere Aussicht für Hardware Ray Tracing

Multiple Shader Varianten

Da die Berechnungen des Schalls pro Objekt von dessen Shader abhängen, lassen sich verschiedene Shader implementieren, welche beispielsweise Transmission nicht berechnen, da sie diese nicht erlauben. Dieser Ansatz ermöglicht es für jedes Objekt ein anderes Verhalten darzustellen. Es könnten beispielsweise auch voluminöse Berechnungen getätigt werden, um gegebenenfalls Bäume und Büsche darzustellen. Auch lässt sich der Übergang von Schall in Wasser dadurch separat darstellen.

Alternative Grafik APIs

Anhand der durchgeführten Leistungsanalyse lässt sich sehen, dass der größte Teil (95%) der Laufzeit für die Konvertierung der Daten für den CPU benötigt wird. Dieser Prozess wurde dabei genauer analysiert. Die Ergebnisse dieser Betrachtungen haben ergeben, dass der größte Aufwand innerhalb dieser Funktion auf das Abrufen der berechneten Daten von der Grafikkarte zurückzuführen ist. Da das Abrufen der Daten in jeder Grafik API auf eine unterschiedliche Weise funktioniert, ist dabei nicht auszuschließen, dass die Verwendung von Vulkan dieses Bottleneck[3] minimieren kann. Leider konnte in dieser Arbeit nur DirectX 12 getestet werden. Dazu kommt noch, dass die Entwickler von Unity gegebenenfalls keinen Fokus auf die Laufzeit dieser Funktion gelegt haben. Würde sich diese Funktion vom Haupt-Thread auf einen Hintergrund-Thread legen lassen, so wäre die Laufzeit der Hardware Ray Tracing Implementation um 95% reduziert.

6.3 Vergleich zu Wwise

Um ein Fazit fällen zu können, müssen die erstellten Implementationen mit Wwise verglichen werden. Vereinfacht ausgedrückt, bietet Wwise so ziemlich jedes Feature, welches in dieser Arbeit implementiert wurde und mehr. Eine Besonderheit in dieser Arbeit ist jedoch das Auslagern der Berechnungen auf die Grafikkarte und die äußerst schnelle Parametrisierung. Auch können in dieser Arbeit Texturen (Bilder) für die lokale Beschreibung von Diffusion und Absorption verwendet werden. Das Konzept von Diffusion und Isometrie werden dabei ebenfalls nur in dieser Arbeit beschrieben. Auch besitzen die Acoustic Textures in Wwise keine Schallgeschwindigkeit. [vgl. Auda] Dadurch kann davon ausgegangen werden, dass Schall nicht richtig gebrochen wird beim Übergang zwischen zwei Medien. Die Laufzeiten der Implementationen in dieser Arbeit sind die Aspekte, welche am kritischsten zu betrachten sind. Da die Umsetzungen direkt im Code des Spieles implementiert sind, haben sie einen direkten Einfluss auf die Leistung der Software. Wwise hat dabei den großen Vorteil, komplett unabhängig von der Ausführung des Spiels zu sein. Auch ist die Berechnung des Halls im Rahmen dieser Arbeit weniger gut gelungen. In diesem Aspekt ist Wwise definitiv zu empfehlen. Zusätzlich besitzt Wwise die Fähigkeit alle einzelnen Effekte eines Geräusches einzeln im Spatial Audio Engine zu berechnen. Dies erlaubt es Wwise Echos und alle anderen Effekte einzeln räumlich zu berechnen. Das eingebaute System von Unity erlaubt solch eine Struktur unter keinem Umstand.

[3]Ein Bottleneck ist der Engpass eines Systems, welches die allgemeine Leistung limitiert.

6.4 Abschließendes Fazit

Werden die Implementationen mit ihren aufgezeigten Potenzialen gewertet, so lässt sich als abschließendes Fazit sagen, dass beide Implementationen für die richtigen Projekte durchaus sinnvoll sind. Wwise ist zwar eine weit überlegene Audio Engine, allerdings ist sie kostenpflichtig. Die allgemeine Implementation eines Audio Ray Tracers ist dabei eine überschaubare Aufgabe für ein kleines Team an Programmierern. Dies ist nicht nur eine Möglichkeit für kleine Spielstudios, welche kein Budget für Wwise haben, sondern auch eine Lösung für Engines, welche keine Wwise Integration vorweisen können. Zusätzlich ergibt die in dieser Arbeit verwendete Methode zur Approximation der Diffraktion überzeugende Ergebnisse. Sie ist zwar nicht physikalisch korrekt, allerdings ist sie um einiges performanter. Auch lässt die Implementation von Hardware Ray Tracing eine voluminöse Berechnung der Rays zu.

Abschließend lässt sich sagen, dass sich die eigenständige Implementation nur lohnt, wenn das Budget oder die verwendete Game Engine die Nutzung von Wwise nicht zulassen.

Abbildungsverzeichnis

Tabellenverzeichnis

Abkürzungsverzeichnis

GPU Graphics Processing Unit

SoC System on a Chip

CPU Central Processing Unit

EQ Equalizer

SIL Sound Intensity Level

SPL Sound Pressure Level

API Application Programming Interface

BSP Tree Binary Space Partitioning Tree

TOH Threshhold of Hearing

Literaturverzeichnis

[aco] *Absorption Coefficient Chart.* https://www.acoustic-supplies.com/absorption-coefficient-chart/

[And09] Andrew, Turner John, Anderson Phil, Lachlan-Cope Tom, Colwell Steve, Phillips Tony, Kirchgaessner AméLie, Marshall Gareth J., King John C., Bracegirdle Tom, Vaughan David G., Lagun Victor Orr: Record low surface air temperature at Vostok station, Antarctica. (2009). http://nora.nerc.ac.uk/id/eprint/9656/1/jgrd15635.pdf

[App68] Appel, Arthur: Some Techniques for Shading Machine Renderings of Solids. (1968), 1–9. https://graphics.stanford.edu/courses/Appel.pdf

[Auda] *Using the Geometry API for Simulating Early Reflections Introduction.* https://www.audiokinetic.com/library/edge/?source=SDK&id=spatial_audio_apigeometry_er.html

[Audb] *Wwise Reflect.* https://www.audiokinetic.com/products/plug-ins/reflect/

[Audc] *Wwise Spatial Audio.* https://www.audiokinetic.com/products/wwise-spatial-audio/

[DA19] Digital Archaeology, The I.: The Science of 3D Rendering. (2019). http://digitalarchaeology.org.uk/the-science-of-3d-rendering/

[Fia18] Fianhua, Huang: A Simple Accurate Formula for Calculating Saturation Vapor Pressure of Water and Ice. (2018). https://journals.ametsoc.org/view/journals/apme/57/6/jamc-d-17-0334.1.xml

[Fli] *Flight Simulator.* https://en.wikipedia.org/wiki/Microsoft_Flight_Simulator_(2020_video_game)

[FTC+04] Funkhouser, Thomas ; Tsingos, Nicolas ; Carlbom, Ingrid ; Elko, Gary ; Sondhi, Mohan ; West, James E. ; Pingali, Gopal ; Min, Patrick ; Ngan, Addy: A beam tracing method for interactive architectural acoustics. In: *The Journal of the Acoustical Society of America* 115 (2004), Nr. 2, 739-756. http://dx.doi.org/10.1121/1.1641020. – DOI 10.1121/1.1641020

[Gar20] Garrett, Steven L.: *Understanding Acoustics: An Experimentalist's View of Sound and Vibration.* 2. Springer, 2020 https://www.amazon.de/-/en/Steven-L-Garrett-ebook/dp/B08MKXFKBB/ref=tmm_kin_swatch_0?_encoding=UTF8&qid=1628171839&sr=8-1. – ISBN 978–3–030–44786–1

[gee20] *Binary Space Partitioning.* https://www.geeksforgeeks.org/binary-space-partitioning/. Version: September 2020

[Hyp] *Wave Speeds*. http://hyperphysics.phy-astr.gsu.edu/hbase/Sound/souspe2.html

[Lab] Labschütz, Matthias: Realistic real-time audio rendering in virtual environments. https://citeseerx.ist.psu.edu/viewdoc/download?doi=10.1.1.248.4610&rep=rep1&type=pdf

[Lib] *17.8: The Doppler Effect*. https://phys.libretexts.org/Bookshelves/University_Physics/Book%3A_University_Physics_(OpenStax)/Book%3A_University_Physics_I_-_Mechanics_Sound_Oscillations_and_Waves_(OpenStax)/17%3A_Sound/17.08%3A_The_Doppler_Effect

[Med] Metropolis Light Transport for Participating Media. https://www.uni-kl.de/AG-Heinrich/MediaMLT.pdf

[Nav] Nave, R.: *Bulk Elastic Properties*. http://hyperphysics.phy-astr.gsu.edu/hbase/permot3.html

[Ref13] *Reflection of Sound*. http://aven.amritalearning.com/index.php?sub=102&brch=303&sim=1549&cnt=3641. Version: 2013

[Sch97] Schultz, Charity Lu, Alex Roetter, Amy: Sophomore College Ray Tracing Site. (1997). https://cs.stanford.edu/people/eroberts/courses/soco/projects/1997-98/ray-tracing/intro.html

[Scr] *Introduction to Ray Tracing*. https://www.scratchapixel.com/lessons/3d-basic-rendering/introduction-to-ray-tracing/raytracing-algorithm-in-a-nutshell

[Sena] *Berechnen der Schallgeschwindigkeit in Luft und die wirksame Temperatur*. http://www.sengpielaudio.com/Rechner-schallgeschw.htm

[Senb] *Conversion of sound units*. http://www.sengpielaudio.com/calculator-soundlevel.htm

[Senc] *How does the sound or the noise depend on distance from the source*. http://www.sengpielaudio.com/calculator-SoundAndDistance.htm

[Send] Sengpiel, Alexander: *Formula for calculating the damping of air*. http://www.sengpielaudio.com/AirdampingFormula.htm

[Ska] Skalsky, Michal: *DXR PathTracer in Unity*. https://github.com/SlightlyMad/SimpleDxrPathTracer

[Sve02] Svensson, U.P.: Modelling acoustic spaces for audio virtual reality. (2002), 01

[tec] *GTX 1070 vs RTX 2080*. https://technical.city/en/video/GeForce-GTX-1070-vs-GeForce-RTX-2080

[The15] *Sabine's Formula*. https://www.thermaxxjackets.com/sabine-modern-architectural/. Version: Januar 2015

[Uni12] University, Arizona S.: *World: Highest Temperature*. https://web.archive.org/web/20170714144146/https://wmo.asu.edu/content/world-highest-temperature. Version: 2012

[Unr] *A first look at Unreal Engine 5*. https://www.unrealengine.com/en-US/blog/a-first-look-at-unreal-engine-5

Erklärung

Hiermit versichere ich, dass ich die vorliegende Arbeit selbstständig angefertigt, nicht anderweitig für Prüfungszwecke vorgelegt, alle benutzten Quellen und Hilfsmittel angegeben, sowie wörtliche und sinngemäße Zitate gekennzeichnet habe.

87435 Kempten, den 27. September 2021 .....

Unterschrift des Verfassers

Ermächtigung

Hiermit ermächtige ich die Hochschule Kempten zur Veröffentlichung der Kurzzusammenfassung (Abstract) meiner Arbeit, z.B. auf gedruckten Medien oder auf einer Internetseite.

87435 Kempten, den 27. September 2021 .....

Unterschrift des Verfassers