TECHNISCHE UNIVERSITÄT DRESDEN

Software Engineering Ubiquitärer Systeme

Großer Beleg

# Konzeption und Entwicklung eines ubiquitären 3D-Sound-Systems

8. November 2012

Christian Schäfer
Mat.-Nr.: 3309131

**Betreuer**
Dipl.-Inf. Katja Tietze

**Verantwortl. Hochschullehrer**
Jun.-Prof. Dr.-Ing. Thomas Schlegel

## Erklärung

Hiermit erkläre ich, dass ich die vorliegende Arbeit mit dem Titel

**Konzeption und Entwicklung eines ubiquitären 3D-Sound-Systems**

unter Angabe aller Zitate und nur unter Verwendung der angegebenen Literatur und Hilfsmittel selbstständig angefertigt habe.

Dresden, den 8. November 2012

*Christian Schäfer*

# Inhaltsverzeichnis

# 1 Einleitung

Mit dem Aufkommen von 3D-Ego-Shootern wie „Doom“ [sof] in den 1990ern gab es erste Bestrebungen, neben der Grafik auch dem Sound in Videospielen räumliche Tiefe zu verleihen. Anfangs sollte mittels einfacher Links-Rechts-Aufteilungen des Audiosignals bei Stereolautsprecheraufstellungen ein 3D-Effekt erzielt werden. Später erfolgte eine Erweiterung dieses Setups durch hintere und seitliche Lautsprecher. Verbesserte 3D-Audio-Algorithmen ermöglichten dem Spieler dadurch eine präzisere horizontale Ortung von Schallquellen.
Menschen haben die Fähigkeit, Klänge aus allen Richtungen zu orten und sind dadurch in der Lage, Informationen über ihre Umwelt zu gewinnen (siehe Abschnitt 2.3, Seite 12). Ziel eines 3D-Sound-Systems sollte es deshalb sein, eine Klangwiedergabe nahe der Wirklichkeit zu simulieren, inklusive einer realistischen Ortung von Klängen im Raum.
Eine genaue vertikale Ortung ist mit den Systemen aus dem Heimanwender- beziehungsweise Gamingbereich jedoch bis heute nicht nach dem Vorbild der Umwelt realisierbar, da die Hersteller solcher Soundsysteme die Anordnung aller Lautsprecher lediglich in der horizontalen Ebene vorsehen.
In dieser Arbeit wird eine Lösung dieses Problems mittels der Kaskadierung mehrerer 7.1-Heimkinosysteme vorgestellt. Durch die vertikale Anordnung der Soundsysteme in Lautsprecherebenen wird der Raum in kleinere Bereiche aufgegliedert, welche einzeln angesprochen werden können. So lassen sich Klänge auch zwischen den Lautsprecherebenen positionieren und bewegen. Benutzer des im Rahmen dieser Arbeit entwickelten Systems gewinnen dadurch den Mehrwert der vertikalen und horizontalen Ortbarkeit von Klängen, was im Gegensatz zu herkömmlichen 5.1- und 7.1-Soundsystemen eine realitätsnahe Klangwiedergabe ermöglicht.

## 1.1 Problemstellung

Die Entwicklung und Verbreitung von 3D-Sound (siehe Abschnitt 2.1.1, Seite 5), welcher sowohl horizontal als auch vertikal präzise ortbar ist, wurde bislang vor

allem durch Virtual-Reality-Anwendungen vorangetrieben. Diese erfordern jedoch ein hohes Maß an Realismus, um den Benutzer in Simulationen eintauchen zu lassen. Projekte wie beispielsweise die Allosphere [HKMA07], ein kugelförmiger Raum, über dessen Innenseite sich 140 Lautsprecher und Bildprojektionen von 12 High-Definition-Beamern erstrecken, setzen maßgeblich auf 3D-Sound.
Das Problem dabei ist, dass es sich um Spezialentwicklungen handelt. Die Erforschung und Realisierung eines solchen Spezialsystems ist üblicherweise mit einem erheblichen Kostenaufwand verbunden [KST05]. Zudem sind diese Systeme oftmals unflexibel, da sie nur für einen bestimmten Einsatz konzipiert sind und sich nicht erweitern lassen [Gay02]. Die Installation der meisten Spezialsysteme wie beispielsweise IOSONO lässt keine Wahl der Hardware zu, sondern ist auf genau vorgegebene Komponenten angewiesen [Lee04].
Bei der Entwicklung eines 3D-Sound-Systems ist es von Vorteil, bestehende Audio-APIs zu verwenden. Jedoch sind viele APIs nur unzureichend dokumentiert, was deren Einsatz in der Praxis erschwert.

## 1.2 Zielsetzung

Ziel dieser Arbeit ist es, ein Modell für ein ubiquitäres 3D-Sound-System zu entwickeln und dieses prototypisch umzusetzen.
Zunächst wird der Stand der Forschung und Technik analysiert, um geeignete Ansätze für ein solches System zu finden. In die Betrachtung sollen insbesondere vergleichbare Soundsysteme einfließen und Raummodelle sowie Audio-APIs evaluiert werden. Auf Basis der gewonnenen Erkenntnisse wird daraufhin das Konzept für ein ganzheitliches System erstellt.
Weiterhin soll im Rahmen des zu entwickelnden Modells speziell darauf eingegangen werden, wie Schallquellen im Raum repräsentiert (Raummodell) und kostengünstige Komponenten aus dem Heimkino- beziehungsweise Gamingbereich zu einem funktionierenden System zusammengefügt werden können (Soundsystemmodell). Dabei soll eine flexible Architektur ermöglichen, dieses 3D-Sound-System mit beliebiger Audiohardware zu betreiben und das Konzept auf mehrere Räume zu erweitern.
Zur Vervollständigung des Systems bedarf es des Weiteren der Aufstellung eines Benutzermodells, welches insbesondere die Positionierung und Bewegung einzelner oder mehrerer Hörer betrachtet. Gleichzeitig soll aufgezeigt werden, wie sich Schallquellen erzeugen lassen, die auf einzelne Nutzer im Raum ausgerichtet sind, diese begleiten und sie mit akustischen Informationen wie persönlichen Telefongesprächen versorgen können.
Das Gesamtmodell soll einen praktikablen Ansatz bieten, mit dem sich auch ohne

großen Einarbeitungsaufwand ein 3D-Sound-System umsetzen lässt.
Abschließend soll der Prototyp des entwickelten Konzeptes mittels zweier 7.1 Soundsysteme für einen Raum implementiert werden.

## 1.3 Aufbau der Arbeit

Die theoretischen Grundlagen der vorliegenden Arbeit werden im Kapitel 2 (ab Seite 5) näher beleuchtet, was zum besseren Verständnis der weiteren Überlegungen beiträgt.
Es werden Begriffe wie „ubiquitäres 3D-Sound-System“ und „Szenengraph“ erklärt. Zudem erfolgt eine kurze Einführung in die notwendigen Grundlagen der Akustik und des menschlichen Hörens.
Am Ende dieses Kapitels wird ein Anforderungskatalog für ein ubiquitäres 3D-Sound-System aufgestellt, anhand dessen sich die untersuchten Modelle und APIs sowie der konzipierte Prototyp im weiteren Verlauf der Arbeit evaluieren lassen.

In Kapitel 3 (ab Seite 19) werden vergleichbare 3D-Sound-Systeme vorgestellt und Raummodelle sowie Audio-APIs auf ihre Eignung zur Konzeption eines solchen Systems geprüft.

Das in dieser Arbeit enwickelte Konzept für ein 3D-Sound-System wird in Kapitel 4 (ab Seite 35) vorgestellt. In diesem Kapitel wird insbesondere auf die Bestandteile Raum-, Benutzer- und Soundsystemmodell sowie die zugrundeliegende Architektur eingegangen.

Kapitel 5 (ab Seite 53) beschreibt die prototypische Umsetzung des zuvor entwickelten Konzeptes und stellt dabei relevante Implementierungsdetails näher vor.

Eine umfassende Evaluation des Prototypen erfolgt in Kapitel 6 (ab Seite 71). Darin wird die Implementierung anhand der in Kapitel 2 aufgestellten Kriterien bewertet.

Ab Seite 75 wird in Kapitel 7 die Arbeit zusammengefasst und ein Ausblick zu möglichen weitergehenden Forschungen und Entwicklungen des Systems gegeben.

# 2 Theoretische Grundlagen

Einleitend sollen einige für diese Arbeit relevanten Begriffe und Grundlagen erklärt werden. Dabei wird vor allem auf Charakteristika der Akustik und des menschlichen Hörens eingegangen. Insbesondere das Zusammenwirken von Schallquelle und Hörer in einer akustischen Umgebung soll dem Leser näher gebracht werden. Zudem wird erklärt, wie der Mensch Klang wahrnimmt und welche Rückschlüsse er daraus bezüglich seiner Umwelt ziehen kann.
Am Ende dieses Kapitels wird ein Anforderungskatalog aufgestellt, anhand diesem sich die im nächsten Kapitel vorgestellten Modelle und APIs sowie der implementierte Prototyp evaluieren lassen.

## 2.1 Begriffe

Die im Folgenden erläuterten Begriffe definieren wichtige Aspekte dieser Arbeit.

### 2.1.1 3D-Sound

In dieser Arbeit wird „3D-Sound“, basierend auf den „3D Audio Rendering and Evaluation Guidelines, Revision 1.0“ der IASIG [Int98], wie folgt definiert:
**Sound kann ohne Verzögerung, durch Nachbildung von dreidimensionalen akustischen Signalen im Raum sowohl horizontal als auch vertikal um einen oder mehrere Hörer platziert werden. Dabei sollte die Wahrnehmung eines Hörers dem Wahrnehmungsverhalten in natürlicher Umgebung möglichst nahe kommen.**
Oftmals verwenden Hersteller von Soundsystemen Begriffe wie „Surround Sound“ und „3D-Sound“ synonym um ihre Produkte zu bewerben. Jedoch handelt es sich bei „Surround Sound“ nicht um „3D-Sound“ nach der oben eingeführten Definition.
„Surround Sound“ steht für Mehrkanalinhalte, durch die ein Klangfeld erzeugt

werden kann, das wesentlich größer als das von Stereo ist. Durch zusätzliche Lautsprecher hinter und optional neben dem Hörer lässt sich Klang um ihn verteilen („surround“). Das Hinzufügen von weiteren Lautsprechern um den Hörer erzeugt jedoch nicht zwingend eine realistische Hörumgebung, da die zusätzlichen Lautsprecher nicht dazu beitragen, die Wahrnehmung von Entfernungen und Höhe zu verbessern. Diese ist jedoch für den Menschen notwendig, um die Umwelt als realistisch zu empfinden [Int98].

### 2.1.2 3D-Sound-System

3D-Sound-Systeme simulieren eine akustische Umgebung möglichst realitätsnah innerhalb eines Raumes. Dabei werden Klänge nicht nur horizontal, sondern auch vertikal um die Benutzer ausgebreitet, womit den Nutzern eine präzise Ortung der Schallquellen im Raum ermöglicht wird.

### 2.1.3 Ubiquitäres 3D-Sound-System

Marc Weiser hatte bereits Anfang der 1990er visionäre Ideen, wie das Arbeiten mit Computern aussehen wird. Er prognostizierte, dass sich die Computernutzung verbessert, indem Rechner vorhanden sind, die den Nutzer umgeben, für ihn jedoch gewissermaßen unsichtbar sind. Das Arbeiten mit diesen **allgegenwärtigen** Systemen bezeichntete er als „Ubiquitous Computing“ [Wei93].
Ubiquitäre 3D-Sound-Systeme können einen Benutzer umgeben, ohne sich ihm aufzudrängen. Sie können den Anwender mit akustischen Informationen unterstützen, wenn sie ihm zum Beispiel den Weg weisen oder oder ihn bereits auf ein anstehendes Meeting vorbereiten, indem sie ihn bis zum entsprechenden Ziel begleiten. Der Nutzer hat dadurch beide Hände frei und braucht sich nicht mehr mit der Bedienung einer grafischen Oberfläche aufzuhalten, um an Informationen zu gelangen. Ubiquitäre 3D-Sound-Systeme sind nicht auf einen Bereich limitiert. Sie können sich über komplette Etagen oder Gebäude erstrecken und diese vollständig sonifizieren.

### 2.1.4 Szenengraph

Szenengraphen wurden im Bereich der Computergrafik vor allem zur Strukturierung von dreidimensionalen Darstellungen entwickelt und haben sich in dieser

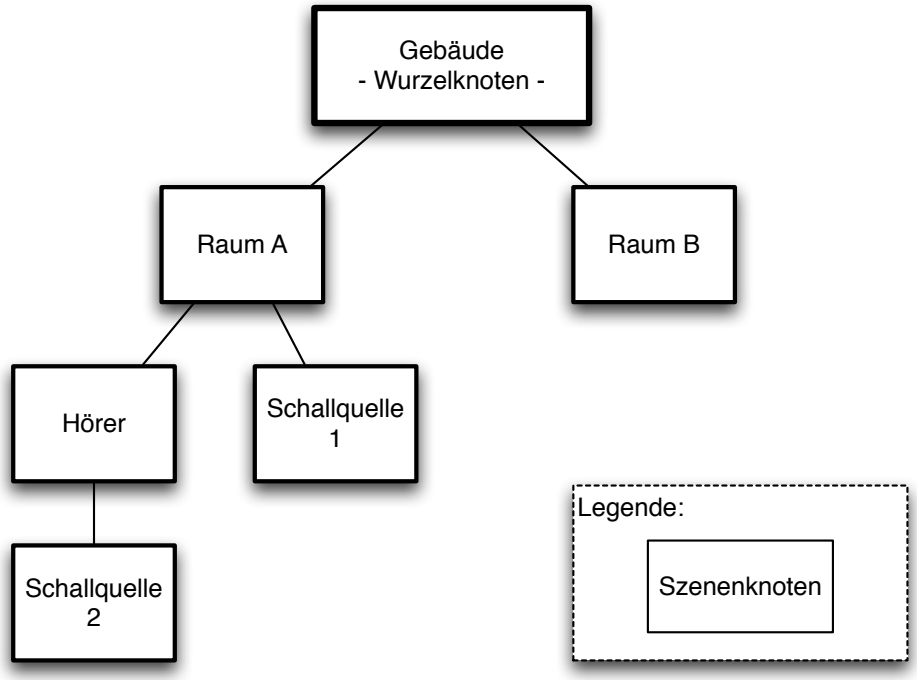


**Abbildung 2.1:** Beispiel Szenengraph mit unterschiedlich untergeordneten Schallquellen [*eigene Darstellung*]

Disziplin seit langem etabliert.
Aaron E. Walsh beschreibt in seinem Artikel „Understanding Scene Graphs“ [Aar02] Szenengraphen als Datenstrukturen zur hierarchischen Organisation und Verwaltung der Inhalte räumlich ausgerichteter Szenendaten.
Da auch in dreidimensionalen Audioszenen sehr viele im Raum positionierte Objekte auftreten, wurde das Konzept des Szenengraphen als Verwaltungsstruktur für diese Arbeit aufgegriffen und an die Anforderungen eines 3D-Sound-Systems angepasst. In dem entwickelten Raummodell (siehe Abschnitt 4.1, Seite 35) wird darauf näher eingegangen.
Szenengraphen sind meistens in Form einer Baumstruktur angelgt, welche in regelmäßigen Abständen durchlaufen wird. Dabei wird rekursiv geprüft, ob in einem Knoten Änderungen aufgetreten sind. Wenn dies der Fall ist, werden die Änderungen in Relation zum jeweiligen Überknoten berechnet.
Zum Beispiel könnte ein Szenengraph mit folgender Struktur existieren: Zwei Räume, die dem Wurzelknoten untergeordnet sind. Ein Hörer und eine Schallquelle (1) ordnen sich Raum A unter und Raum B besitzt keine Unterknoten. Zudem ist dem Hörer aus Raum A noch die Schallquelle 2 untergeordnet (siehe Abbildung 2.1, Seite 7).

Wenn sich zum Beispiel der Hörer in Raum A bewegt, so würde im nächsten Durchlauf des Szenengraphen seine Position neu berechnet, indem die erfolgte Positionsänderung zu seiner alten Position addiert wird. Danach wird diese Änderung an all seine Unterknoten (Schallquelle 2) weitergegeben und auf deren Position addiert.
Da lediglich die Änderungen in Relation zum Überknoten berechnet werden, lassen sich durch die Nutzung von Szenengraphen im Vergleich zu Strukturen, die alle Änderungen berechnen, erhebliche Performancegewinne erzielen.
Die Aufgliederung in Knoten trägt insbesondere bei größeren Projekten erheblich zur Übersichtlichkeit bei und unterstützt objektorientiertes Programmieren, da jeder Knoten ein Objekt beziehungsweise eine Objektinstanz darstellen kann.
Ein weiterer Vorteil ist die Flexibilität, da man einzelnen Knoten des Baumes unterschiedliche (Audio-) Renderer zuweisen kann. So lässt sich die Last auf mehrere Renderer verteilen, oder es lassen sich die gleichen Daten durch unterschiedliche Renderer auf verschiede Weise aufbereiten. Beispielsweise könnte man die Szene für ein nicht so leistungsfähiges Gerät mit weniger Details berechnen. Auf einem leistungsstarken Gerät könnten hingegen alle Details dargeboten werden.

### 2.1.5 Sweetspot

Der sogenannte Sweetspot wird in den „3D Audio Rendering and Evaluation Guidelines - Level 1.0“ [Int98] definiert als: „Der Ort an dem ein Hörer platziert werden muss, um den optimalen Höreindruck innerhalb eines spezifischen Lautsprechersetup zu bekommen“.
In dieser Arbeit bezieht sich der Sweetspot nicht auf einen bestimmten Ort, an dem sich eine Person befinden muss. Vielmehr wird im Rahmen der vorliegenden Arbeit der **Mittelwert der Positionen aller Hörer eines Raumes** als Sweetspot bezeichnet. Dieser Punkt wird herangezogen, um die Abstände zu allen Schallquellen im Raum zu berechnen, sofern ihr Klang für alle anwesenden Personen bestimmt ist. Anhand der Distanz zwischen dem Sweetspot und einer Schallquelle kann die notwendige Dämpfung des Schalls ermittelt werden, mit welcher sich daraufhin der 3D-Effekt erzeugen lässt.

## 2.2 Akustik

In Anlehnung an die „3D Audio Rendering and Evaluation Guidelines, Revision 1.0“ [Int98] soll kurz dargestellt werden, wie Sound in einer natürlichen Umgebung

unter realistischen Bedingungen wahrgenommen wird. In diesem Zusammenhang ist es notwendig, auf die drei Bestandteile **Schallquelle**, **akustische Umgebung** und den **Hörer** im Einzelnen einzugehen, weil sie in direktem Bezug zueinander stehen und den Klangeindruck durch ihre Positionierung, ihre physischen Eigenschaften und ihre Geometrie wesentlich beeinflussen.

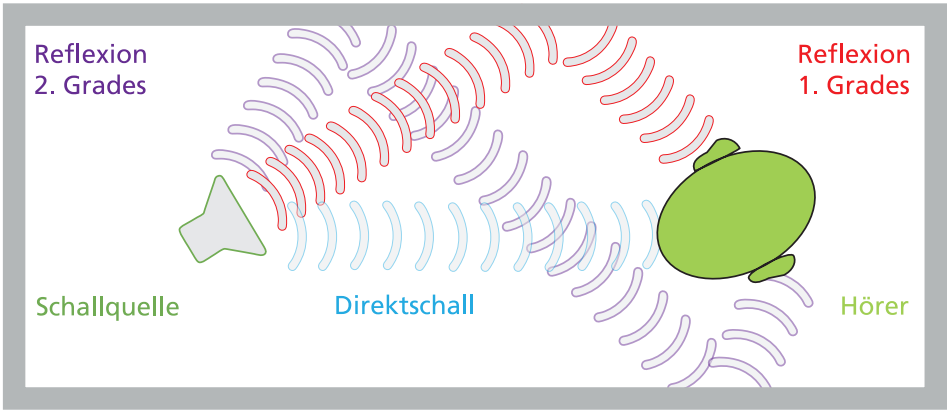


**Abbildung 2.2:** Schallfeld inkl. einer Schallquelle und einem Hörer [*eigene Darstellung*]

**Eine Schallquelle** ist ein Objekt, das Schallwellen ausgibt. Die Schallwellen werden durch mechanische Prozesse erzeugt und danach in bestimmte Richtungen verbreitet (siehe Abbildung 2.2, Seite 9).
Ein Lautsprecher ist zum Beispiel eine Schallquelle, welche Membranen zum Schwingen bringt, um Schallwellen zu erzeugen. Diese breiten sich jedoch nicht nur orthogonal zur Membran in den Raum aus, sondern verteilen sich je nach Frequenz in unterschiedlichem Maße kugelförmig um den Lautsprecher. Lautsprecher, die auf Grund ihrer Charakteristik in der Lage sind, Schall relativ stark gerichtet beziehungsweise gebündelt auf den Hörer abzustrahlen, erzeugen weniger Reflexionen als diffuse Schallquellen. Dies ermöglicht es, Klang präzise auf einen kleinen Bereich auszurichten, welcher die optimale Hörposition darstellt [Fai]. Diese wird auch oftmals als „Sweetspot“ (siehe Abschnitt 2.1.5, Seite 8) beizeichnet.

**Akustische Umgebung** Nachdem Schall erzeugt wurde, wird er innerhalb der akustischen Umgebung ausgebreitet und von dieser beeinflusst. Von besonderer Relevanz sind dabei **Absorption** und **Reflexion**, auf welche im Folgenden eingegangen wird.

**Absorption** Jeder Gegenstand besteht aus einem bestimmten Material. Wenn Schall auf diesen Gegenstand trifft, wird ein Teil seiner Energie in Wärme umgewandelt und steht dem Hörer nicht mehr zur Verfügung. Dieser umgewandelte Teil wurde absorbiert (siehe Abbildung 2.3, Seite 10). Absorption führt dazu, dass der Hörer den Schall leiser wahrnimmt und ist im Wesentlichen von der Beschaffenheit des Materials sowie der Frequenz des Schalls abhängig. Dabei werden hochfrequente Schallwellen mehr absorbiert als tieffrequente. Weiches, poröses Material absorbiert stärker als hartes und glattes, da Schallwellen besser in weiche Materialien eindringen können, ohne größtenteils reflektiert zu werden [Fai].
Nicht nur Gegenstände beeinflussen den Grad der Absorption. Eigenschaften des Ausbreitungsmediums, wie die Luftfeuchtigkeit bei einer Schallausbreitung in Luft, sind ebenso an der Absorption von Schall beteiligt [Int98].

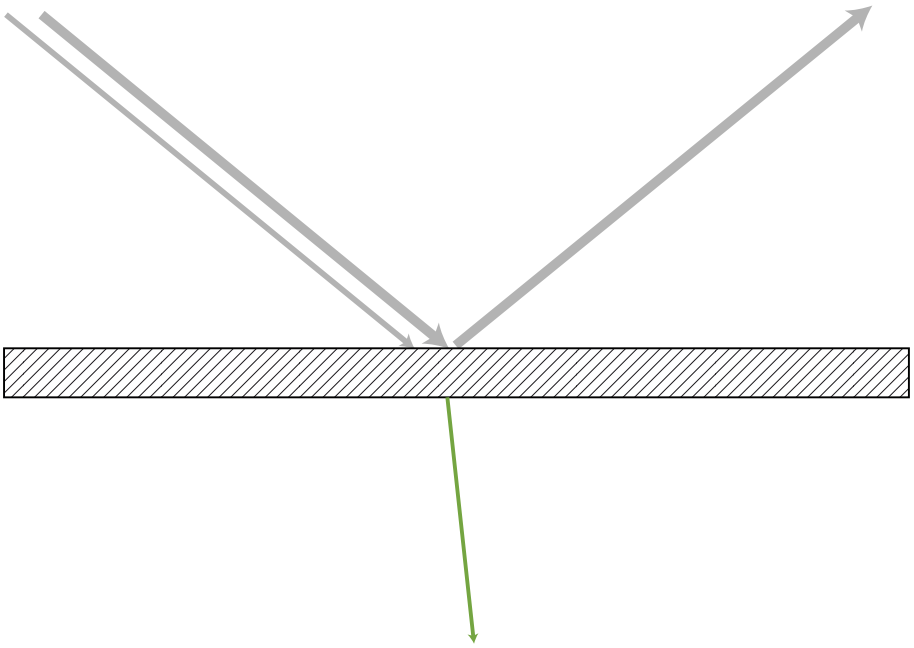

**Abbildung 2.3:** Absorption [*eigene Darstellung*]

**Reflexion** Trifft Schall auf eine Begrenzungsfläche, so wird er teilweise von ihr reflektiert. Besteht diese Fläche beispielsweise aus einem glatten und harten Material, so ist die Reflexion am stärksten, der Absorptionsgrad hingegen tendiert gegen Null. Gegenteilig verhält sich offenporiges, weiches Material mit hohem Absorptionsgrad, das viel weniger reflek-

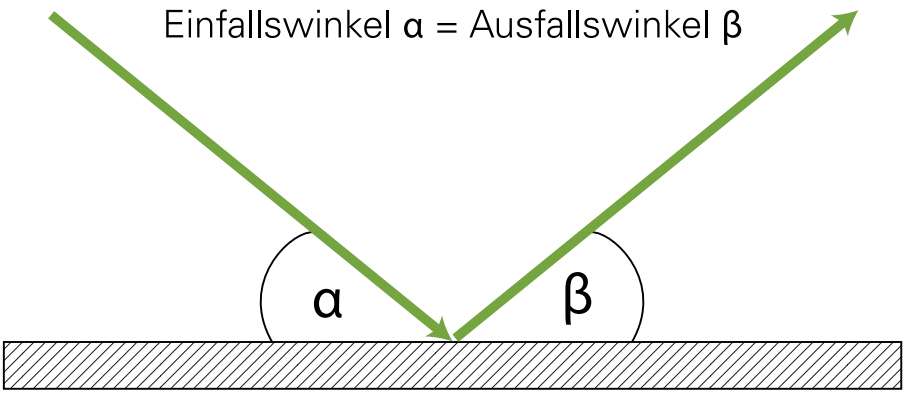


**Abbildung 2.4:** Reflexion [*eigene Darstellung*]

tiert.
Schall kann auf direktem Weg zum Hörer gelangen oder vor der Ankunft einfach, zweifach oder beliebig oft von Flächen reflektiert werden. Für jede Reflexion gilt dabei: Der Einfallswinkel ist gleich dem Ausfallswinkel (siehe Abbildung 2.4, Seite 11).
Bei der Reflexion von Schall kann es passieren, dass sich Schallwellen addieren, wenn diese am gleichen Ort zusammentreffen. Es entsteht eine sogenannte Interferenz. Diese kann einerseits zur Verstärkung (konstruktive Interferenz) führen, wenn beispielsweise zwei identische Wellen aufeinandertreffen (siehe Abbildung 2.5, Seite 12). In diesem Fall wird der Ton lauter.
Andererseits kann Schall ausgelöscht werden (destruktive Interferenz), wenn zwei entgegengesetzte Wellen zusammentreffen (siehe Abbildung 2.6, Seite 12). Das führt im Extremfall dazu, dass man nichts mehr hört [SPS+09].

**Der Hörer** Er ist das zentrale Element des Systems, das aus ihm, der akustischen Umgebung und den Schallquellen besteht. Erst durch seine Existenz wird das Zusammenwirken der einzelnen Systemelemente wahrnehmbar, da er den Schall empfängt und die daraus extrahierten Informationen verarbeitet, um sich ein „Bild“ von seiner Umgebung zu erstellen. Dies wird im nächsten Abschnitt „Menschliches Hören“ näher erläutert (siehe Abschnitt 2.3, Seite 12).
Die Bewegung des Hörers relativ zur Bewegung der Schallquellen nimmt direkten Einfluss auf den Klang. Die bekannteste Auswirkung dieses gegensätzlichen Verlaufs ist der sogenannte Dopplereffekt. Er äußert sich darin, dass sich die wahrgenommene Frequenz erhöht, sobald sich Hörer

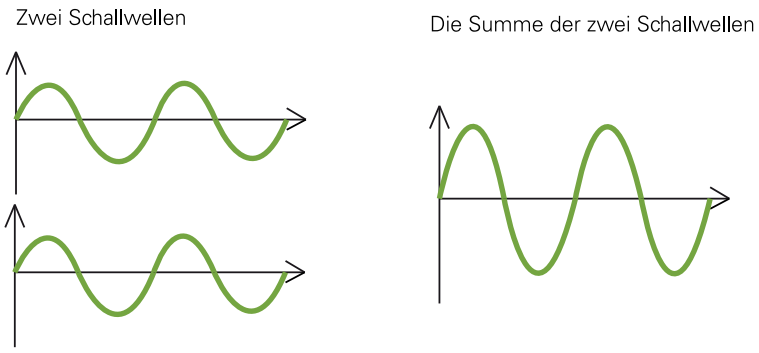


**Abbildung 2.5:** Konstruktive Interferenz [*eigene Darstellung*]

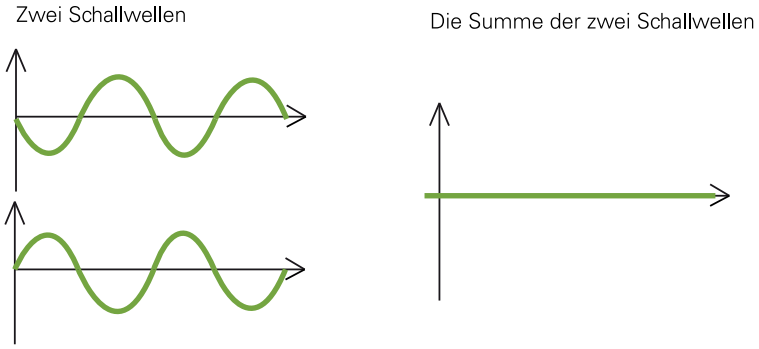


**Abbildung 2.6:** Destruktive Interferenz [*eigene Darstellung*]

und Schallquelle aufeinander zu bewegen. Erfolgt jedoch eine Bewegung in entgegengesetzte Richtungen, führt dies zur Verringerung der wahrgenommenen Frequenz. Zum Beispiel ändert sich in dieser Weise die Tonhöhe des Martinshorns (Schallquelle) eines Krankenwagens, wenn dieser einen Hörer passiert [Int98].

## 2.3 Menschliches Hören

Die Grundlagen des menschlichen Hörens sind ebenfalls in den „3D Audio Rendering and Evaluation Guidelines, Revision 1.0“ [Int98] verständlich erklärt. Darauf basierend werden sie in diesem Kapitel kurz aufbereitet.
Menschen verfügen über ein auditives Abtastsystem zur Wahrnehmung von Schall,

bestehend aus den beiden Ohren zur Aufnahme und dem Gehirn zur Interpretation der gewonnenen Informationen.
Durch die Auswertung der „**I**nteraural **I**ntensity **D**ifference (IID)“ und der „**I**nteraural **T**ime **D**ifference (ITD)“ kann das Gehirn eine erste grundlegende Ortung vornehmen und erkennen, ob sich eine Schallquelle links oder rechts vom Kopf des Hörers befindet (siehe Abbildung 2.7, Seite 13). Die Ohrmuschel liefert Informationen, welche es ermöglichen diese Ortung zu verbessern und die genaue Position einer Schallquelle relativ zum Hörer zu bestimmen. Die Auswertung weiterer Faktoren, wie zum Beispiel wahrgenommenen Reflexionen, verfeinern das entstehende „Bild“ der Umgebung eines Hörers schrittweise. Im Folgenden findet eine kurze Erläuterung der einzelnen Elemente statt.

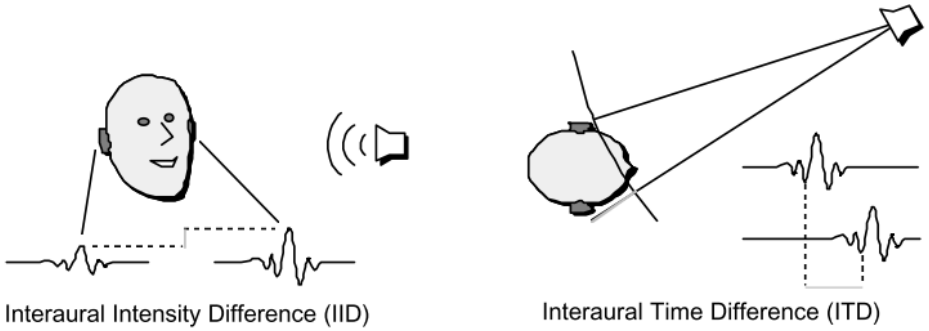


**Abbildung 2.7:** Illustrationen zu IID und ITD, aus [Int98]

**Interaural Intensity Difference (IID)** Steht dafür, dass eine Schallquelle auf dem ihr näher liegenden Ohr lauter wahrgenommen wird als auf dem anderen Ohr, das nicht nur weiter entfernt ist, sondern auch vom Kopf des Hörers verdeckt wird.

**Interaural Time Difference (ITD)** Da sich die beiden Ohren in einem bestimmten Abstand zueinander befinden, ist auch die Distanz zwischen ihnen und der Schallquelle verschieden. Schallwellen erreichen das Ohr mit der kürzeren Distanz als Erstes. Wenn eine Schallwelle zum Beispiel zuerst vom linken Ohr wahrgenommen wird, so kann das Gehirn daraus ableiten, dass sich die Schallquelle irgendwo auf der linken Seite des Kopfes befinden muss. Wenn beide Ohren die Schallwelle gleichzeitig wahrnehmen, deutet dies darauf hin, dass sich die Schallquelle vor oder hinter dem Kopf des Hörers befinden muss, da beide Ohren die gleiche Distanz zu dieser haben. ITD definiert genau den Zeitunterschied des Eintreffens einer Schallwelle auf beiden Ohren und lässt entsprechend Rückschlüsse, die zur Ortung der Schallquelle beitragen, zu.

Eine Kombination der aus IID und ITD gewonnenen Informationen erlaubt es dem Gehirn, die Position der Schallquelle auf einen kegelförmigen Bereich, dessen Mittelachse durch die beiden Ohren verläuft, einzugrenzen. Eine genauere Ortung der Schallquelle kann erst durch die Auswertung der gefilterten Informationen der Pinna erfolgen, auf welche als nächstes eingegangen wird.

**Die Pinna** Die Ohrmuschel, die sogenannte Pinna, verstärkt oder dämpft mittel- und hochfrequente Schallwellen. Dies erfolgt nachdem sie deren Öffnung passiert haben, jedoch noch bevor sie das Trommelfell erreichen. Der Grad dieser Beeinflussung ist abhängig davon, in welchem Winkel eine Schallwelle in der Pinna eintrifft. Die resultierenden Hinweise der richtungsabhängigen Filterung des Frequenzspektrums dienen dem Gehirn zur Bestimmung von Höhe und Ausrichtung einer Schallquelle. Eine zuverlässige und genaue Lokalisierung basierend auf diesen Hinweisen ist nur dann möglich, wenn das Frequenzspektrum des Klanges über eine Bandbreite verfügt, die nach der Filterung noch genug Informationen bezüglich der Form des Spektrums enthält [HVV98].

Dies ist der Schlüssel zur genauen Ortung und erst dadurch wird es dem Gehirn ermöglicht, das Ergebnis der Auswertung von IID und ITD zu verfeinern und somit die genaue Position der Schallquelle, relativ zum Hörer, zu bestimmen.

Da die Pinna und ihre Falten nur einige Zentimeter groß sind, lässt sich lediglich Schall mit einer Wellenlänge im Zentimeterbereich oder kleiner präzise verarbeiten.

Allgemein gilt: Je höher die Frequenz, desto kürzer die Wellenlänge und umso besser lässt sich der Schall orten. Aus diesem Grund ist es möglich, einen Tieftöner im Raum sehr variabel zu positionieren. Aufgrund seiner niedrigen Frequenz und dementsprechend langen Welle kann man seine Position keinem spezifischen Punkt im Raum zuordnen [Int98].

**Ausbreitungseffekte, Reichweite und Reflexionen** Bevor Schall beim Hörer ankommt wird er von vielen Faktoren, wie dem Absorptionsgrad des Ausbreitungsmediums und der Oberflächenbeschaffenheit von Umgebungsflächen, beinflusst. Aus allen Einflüssen können Rückschlüsse gezogen werden, in welchem Umfeld man sich gerade bewegt.

Zum Beispiel lässt sich aus einem leicht dumpfen, leisen Sound ableiten, dass sich die Schallquelle in der Ferne befindet. Ein sehr dumpfes Geräusch deutet jedoch daraufhin, dass man sich in einem geschlossenen Raum mit Wänden aus Glas oder ähnlichen Materialien aufhält. Der Schall wird dabei außerhalb des Raums erzeugt und durchdringt die Wände zu einem geringen Teil.

Anhand von Schallreflexionen kann der Hörer ableiten welche Größe, Form und Beschaffenheit der Raum hat, in dem er sich befindet. So kann eine Reflexion zwischen parallelen Wänden eines Raumes zu einem sogenannten „Flatterecho“ führen. Dies erzeugt entweder ein klares Echo, was auf einen großen Raum hinweist oder eine Art Nachhall des Klanges, was wiederum auf einen kleinen Raum schließen lässt. Wenn sich Schall in einem Raum sehr diffus ausbreitet und somit die Energie des Schalls gleichzeitig in alle Richtungen verteilt wird, deutet dies darauf hin, dass die Begrenzung des Raumes aus vielen unterschiedlich großen Elementen besteht, da diese ein unterschiedliches Reflexionsverhalten vorweisen.
Ungedämpfte, harte Reflexionen sind ein Zeichen dafür, dass die Raumbegrenzungen aus stabilen Materialien mit glatten Oberflächen bestehen, die wenig Schall absorbieren [Fai].
Für die Entwicklung eines Soundsystems, wie in dieser Arbeit, soll erreicht werden, dass Reflexionen im Raum so gut wie möglich unterdrückt werden, um einen neutralen Klang zu erzielen. Dadurch wird es ermöglicht, eine Vielzahl an Reflexionen, Echos und Nachhallzeiten synthetisch zu erzeugen und beispielsweise beliebige Raumgrößen zu simulieren.

Um mittels 3D-Sound-Systemen ein realistisches Schallfeld auf digitale Art nachzubilden, sollen die aufgeführten Bestandteile des menschlichen Hörens bei der Entwicklung eines solchen Systems berücksichtigt werden.
Da jeder Mensch im Laufe seines Lebens akustische Phänomene wie beispielsweise Hall, Reflexionen und Frequenzfilterungen zu interpretieren lernt, lassen sich verschiedene Räume beziehungsweise Umgebungen mit einem 3D-Sound-System realistisch simulieren und gezielt manipulieren.
Von einem kleinen Bad mit gefliesten Wänden, welche viel Hall erzeugen, über eine Unterwasserwelt mit sehr gedämpften Klängen bis hin zu einer offenen Steppe nahezu ohne Reflexionen, kann man einem Hörer nahezu jede akustische Umgebung vorgeben.

## 2.4 Anforderungskatalog

Auf Basis der Zielsetzung dieser Arbeit (siehe Abschnitt 1.2, Seite 2), den theoretischen Grundlagen und der Definition eines ubiquitären 3D-Sound-Systems (siehe 2.1.3, Seite 6) wurde ein Anforderungskatalog aufgestellt (siehe Tabelle 2.1, Seite 16). Dieser gibt vor, welche Kriterien ein (ubiquitäres) 3D-Sound-System unter idealen Umständen erfüllen sollte.

| **Index** | **Anforderung** | **Kriterium** | **M** | **A** | **P** |
|---|---|---|---|---|---|
| RS1 | Positionierung von Schallquellen ohne Verzögerung im Raum | Verzögerungsfreiheit | | | P |
| RS2 | Positionierung von Schallquellen horizontal und vertikal im Raum | 3D-Positionierbarkeit | M | A | P |
| RS3 | Schallquellen unabhängig voneinander erzeug- und bewegbar | Schallquellen-unabhängigkeit | M | A | P |
| RS4 | Systemausdehnung über mehrere Räume | Ubiquität | M | A | P |
| RS5 | Klang kann Richtungen anzeigen | Wegweisungsfähigkeit | | | P |
| H1 | Mehrere Hörer können System unabhängig voneinander nutzen | Mehrbenutzerfähigkeit | M | A | P |
| H2 | diskrete Schallquellen auf einzelne Hörer im Raum ausgerichtet | Koppelbarkeit | M | | P |
| H3 | Klang kann den Hörer über verschiedene Positionen begleiten | Begleitungsfähigkeit | | | P |
| H4 | Klang kann sich vom Hörer entfernen | Distanzierungsfähigkeit | | | P |
| H5 | Individuelle Einstellungen für Hörer | Anpassbarkeit | | A | P |
| SY1 | Hardwareunabhängigkeit durch flexible Architektur | Flexibilität | | | P |
| SY2 | leichte Einarbeitung / gute Dokumentation | Praktikabilität | | A | P |
| SY3 | Audioeingabe (mittels Mikrofon) | Audioeingabefähigkeit | | | P |

**Tabelle 2.1:** Anforderungskatalog

Legende zu Tabelle 2.1 auf Seite 16:

**Index:** ordnet die Kriterien den folgenden Bereichen zu:

**RS:** Raum und Schallquelle

**H:** Hörer

**SY:** System

---

**Anforderung:** Kurzbeschreibung der Anforderung an das System

---

**Kriterium:** aus der Anforderung abgeleitet

---

**M-A-P:** Herangezogen für Bewertung von:

**M:** Modell

**A:** API

**P:** Prototyp

Von besonderer Wichtigkeit bezüglich des Klangeindrucks sind die folgenden Kriterien des Anforderungskatalogs:

**Verzögerungsfreiheit** - damit auf jede Aktion eine direkte Reaktion folgt und das System einen natürlichen Eindruck beim Nutzer hinterlässt

**3D-Positionierbarkeit** - nur wenn Schallquellen an allen Positionen im Raum platziert werden können, ist eine realistische Ortung möglich

**Schallquellenunabhängigkeit** - Erfolgt eine unabhängige Bewegung von Schallquellen (in verschiedene Richtungen) um den Hörer herum, wird der räumliche Eindruck des Hörers nochmals verstärkt

## 2.5 Zusammenfassung

In diesem Kapitel wurden Begriffe wie „Ubiquitäres 3D-Sound-System“ oder „Szenengraph“ erklärt. Dadurch soll dem Leser das Verständis der folgenden Kapitel erleichtert und eine Möglichkeit gegeben werden, wichtige Begriffe nochmals nachzulesen.
Zudem erfolgte ein kurzer Einblick in akustische Grundlagen und Phänomene des menschlichen Hörens.
Im nächsten Kapitel werden verwandte Arbeiten vorgestellt, in denen 3D-Sound-Systeme entwickelt wurden. Dabei verfolgt jede der erwähnten Arbeiten einen anderen Ansatz, um einen möglichst realistischen Klangeindruck zu vermitteln.
Außerdem werden zwei Modelle erläutert, auf deren Basis sich ein Raummodell für ein ubiquitäres 3D-Sound-System entwickeln lässt.
Am Ende des Kapitels werden einige APIs in Betracht gezogen, die für eine prototypische Umsetzung von 3D-Sound-Systemen in Frage kommen.
Anhand des in diesem Kapitel eingeführten Anforderungskataloges werden die gezeigten Modelle und APIs hinsichtlich ihrer Eignung für ein ubiquitäres 3D-Sound-System bewertet.

# 3 Stand der Forschung und Technik

Innerhalb dieses Kapitels sollen Arbeiten vorgestellt werden, welche sich mit verwandten Themen beschäftigen. Einleitend werden drei unterschiedliche Ansätze, ein möglichst realistisch klingendes 3D-Sound-System umzusetzen, vorgestellt. „Dirt Cheap 3D Spatial Audio“, „Sound Element Spatializier“ und „3deSoundBox“ verfolgen dabei unterschiedliche Wege, um dieses Ziel zu erreichen.
Anschließend wird mit „Interactive 3D Sound Hyperstories for Blind Children“ und „Audio3D“ auf Arbeiten eingegangen, die eine geeignete Grundlage für die Entwicklung des Raummodells dieser Arbeit bilden.
Im letzten Teilabschnitt dieses Kapitels werden die Audio-APIs „OpenAL“, „CSL“, „jReality“, „BASS“ und „XNA“ evaluiert, welche für eine prototypische Umsetzung eines 3D-Sound-Systems in Frage kommen.

## 3.1 3D-Sound-Systeme

3D-Sound spielt in immersiven Virtual-Reality-Umgebungen schon sehr lange eine zentrale Rolle. Jedoch kommen dafür häufig Spezialanfertigungen beziehungsweise kommerzielle Systeme wie das vom Fraunhofer IDMT entwickelte IOSONO [Lee04] zum Einsatz.
In diesem Abschnitt wird auf drei verschiedene Projekte eingegangen, die unterschiedliche Ansätze darstellen, ein 3D-Sound-System zu entwickeln.

### 3.1.1 Dirt Cheap 3D Spatial Audio

Dirt Cheap 3D Spatial Audio [KST05] stellt einen der ersten Versuche dar, ein 3D-Sound-System mittels kostengünstiger, handelsüblicher Hardware zu realisieren. Die Lautsprecher werden dabei hexaedrisch angeordnet, wobei jede Box in einer Ecke des Raums befestigt ist. Klein et al. setzen zur Ansteuerung dieser nichtplanaren Lautsprecherkonfiguration auf die Mustajuuri-API [Ilm02]. Mittels dieser wird jedes Eingangssignal auf drei passende Lautsprecherboxen verteilt.

Diese Lösung stellt einen guten und günstigen Einstieg in 3D-Sound (siehe Abschnitt 2.1.1, Seite 5) dar, ist jedoch auf acht Lautsprecher begrenzt, da nur eine 7.1 Soundkarte für dieses Projekt vorgesehen wurde. Desweiteren sind fundierte Linux-Kenntnisse erforderlich, um die Treiber der verwendeten Soundkarte anzupassen beziehungsweise selbst zu programmieren. Mustajuuri setzt zudem voraus, dass die zu verwendenden Sounddateien bereits vor dem Laden des Programms feststehen, was diese Lösung sehr statisch macht. Das Einbinden einer dynamischen Soundquelle zur Laufzeit, wie zum Beispiel eines Telefongespräches, ist somit undenkbar.

### 3.1.2 SES - Sound Element Spatializer

Ryan Michael McGee hat in seiner Masterarbeit [McG10] ein flexibles System entwickelt, welches eine beliebige Anzahl an Lautsprechern und beweglichen Schallquellen unterstützt. Es können sowohl Ausgaben aus Anwendungen, wie Digital Audio Workstations (DAWs), als auch Live-Eingaben, wie zum Beispiel durch Mikrofone, verarbeitet werden.
Die Verteilung dieser eingehenden Signale an die entsprechenden Positionen im Raum übernimmt eine eigene Komponente, der Sound Element Spatializer (SES). Bewegungsbahnen, Lautstärke, Geschwindigkeit, Verzögerung und weitere Parameter können zur Laufzeit mittels entsprechender Steuersignale im SpatDIF-Format [BS11] durch eine eigenständige dritte Komponente vorgenommen werden. Dieser dreigliedrige Aufbau ermöglicht es, die Rechenlast auf mehrere Geräte zu verteilen. Der Vorteil ist nicht nur die Entlastung einzelner Workstations, sondern auch die erhöhte Flexibilität. Durch Verwendung von Standardprotokollen und APIs lassen sich theoretisch DAWs betriebssystemunabhängig in das Audionetzwerk einbinden. Die separate Steuereinheit begünstigt zudem die Implementierung verschiedener Arten der Manipulation, denkbar sind zum Beispiel Eingaben per Touchscreen oder Kinect.
SES wurde in C++ programmiert und setzt auf die CREATE Signal Library (siehe Abschnitt 3.3.2, Seite 28), welche vorwiegend unter MacOSX und Linux verwendet wird. Deshalb wurde SES auch primär für diese Plattformen entworfen.
Der Sound Element Spatializer ist durch seine Distributivität und der damit einhergehenden Flexibilität vor allem für sehr große Projekte mit mehreren DAWs geeignet.

### 3.1.3 3deSoundBox

Die 3deSoundBox [Sta03] stellt einen weiteren flexiblen und zugleich kostengünstigen Ansatz dar, ein 3D-Sound-Systems umzusetzen. Eine solche Box kann maximal ein 7.1 System ansprechen. Es ist jedoch möglich, bis zu 21 Soundboxen zu kaskadieren, was 168 diskrete Ausgabekanäle gestattet.
Anwendungen werden in C++ mittels einer eigenen API für die 3deSoundBoxen geschrieben. Dabei ist der Programmcode immer gleich, unabhängig von der Anzahl der gekoppelten Boxen. Dies hat den Vorteil, dass der Programmierer im kleinen Rahmen, zum Beispiel lediglich mit einem 5.1 System ausgestattet, seine Arbeit direkt testen kann. Das fertige Programm wird trotzdem in einem Konzertsaal mit mehreren 3deSoundBoxen lauffähig sein. Je mehr Lautsprecher angesteuert werden, desto realistischer wird der Gesamteindruck.
Das von Stampfl [Sta03] beschriebene 3D-Sound-System ist auf sehr viele Anwendungen anpassbar und ohne großen Aufwand zu erweitern. Es können Stereo, 4.1, 5.1, 7.1 Systeme oder Kopfhörer zur Wiedergabe verwendet und beliebig kombiniert werden. Zudem erleichtert es Programmierern die Arbeit, da sie gegen die mitgelieferte API programmieren können, und die kaskadierten Soundboxen autonom eine Verteilung auf angeschlossene Lautsprecher realisieren.

## 3.2 Raummodelle

Durch Modelle werden alle Elemente eines 3D-Sound-Systems in abstrakter Form beschrieben, die sich von der Sounderzeugung über die Verarbeitung bis hin zur Ausgabe erstrecken und dazu beitragen, die „wahrgenommene Welt“ formal abzubilden.
Ein Raummodell definiert wie sich virtuelle Schallquellen in einem virtuellen Raum verhalten, wie sie von ihrer Umgebung beeinflusst werden und wie sie ihre Umgebung beeinflussen.
Das Raummodell muss um ein Benutzermodell ergänzt werden, um den Einfluss von Hörern auf die Klänge in einem Raum zu erweitern.
Durch ein Soundsystemmodell kann die zu verwendende Hardware in das Modell integriert und die Beschreibung der Zusammenhänge aller Komponenten eines 3D-Sound-Systems weiter verbessert werden.
In diesem Abschnitt werden zwei Arbeiten vorgestellt, die als Grundlage für das Raummodell eines ubiquitären 3D-Sound-Systems dienen können.
„Interactive 3D Sound Hyperstories for Blind Children“ verfolgt dabei ein eventbasiertes, zustandsabhängiges Konzept, um einzelne Komponenten des Systems

miteinander zu koppeln.
Das Modell aus „Audio3D“ setzt auf einen hierarchischen Szenengraphen zur Modellierung der virtuellen Umgebung.

### 3.2.1 Interactive 3D Sound Hyperstories for Blind Children

Da die meisten Bildrepräsentationen, beispielsweise Videospiele, auf visuelle Wahrnehmung ausgerichtet sind, werden Menschen mit Blindheit von diesen ausgeschlossen. Mauricio Lumbreras und Jaime Sánchez nahmen sich dieses Problem in Interactive 3D Sound Hyperstories for Blind Children [LS99] an und erforschten wie blinde Kinder mittels Soundsystemen Erkenntnisse über die räumlichen Strukturen gewinnen können.
Sie haben in ihrer Arbeit ein Modell entwickelt (siehe Abbildung 3.1, Seite 23), um eine akustisch navigierbare Umgebung zu beschreiben und so narrative Geschichten räumlich erschließbar zu machen.
Basierend auf den Grundklassen „Kontext“, „Link“, „Entität“ und „Kanal“ lässt sich eine virtuelle Welt konstruieren und von einem Spieler durchschreiten.
Ein **Kontext** ist dabei ein Container bestehend aus Objekten (Entitäten).
**Links** geben im Modell von Lumbreras und Sánchez die Verbindung zwischen Kontexten vor und werden beispielsweise in Form von Türen, Fenstern oder Portalen zwischen Kontexten eingesetzt.
**Kanäle** dienen als Transportweg für Eventnachrichten zwischen allen Klassen.
Eine **Entität** stellt ein statisches oder dynamisches Objekt in einem Kontext dar. Statische Objekte sind auf den Kontext limitert, dem sie zugeordnet wurden. Dynamische Objekte hingegen können zwischen verschiedenen Kontexten bewegt werden beziehungsweise sich autonom zwischen diesen bewegen.
Das Hyperstories-Modell ist eventbasiert und nutzt intensiv das State-Machine-Entwurfsmuster, um spezifisches zustandsgesteuertes Verhalten für jede Klasse zu implementieren.
Wenn sich der Spieler beispielsweise mehrfach durch eine virtuelle Welt mit den gleichen Objekten bewegt, so sollte er jedes Mal eine andere Geschichte erleben, da die Objekte abhängig von ihrem Zustand unterschiedliches Verhalten implementieren und so auf Events ihrer Umwelt reagieren. Passive Objekte geben als Reaktion nur einfache Werte zurück, aktive hingegen können komplexe Algorithmen ausführen, um ihr Verhalten abhängig von ihrem Zustand zu berechnen.

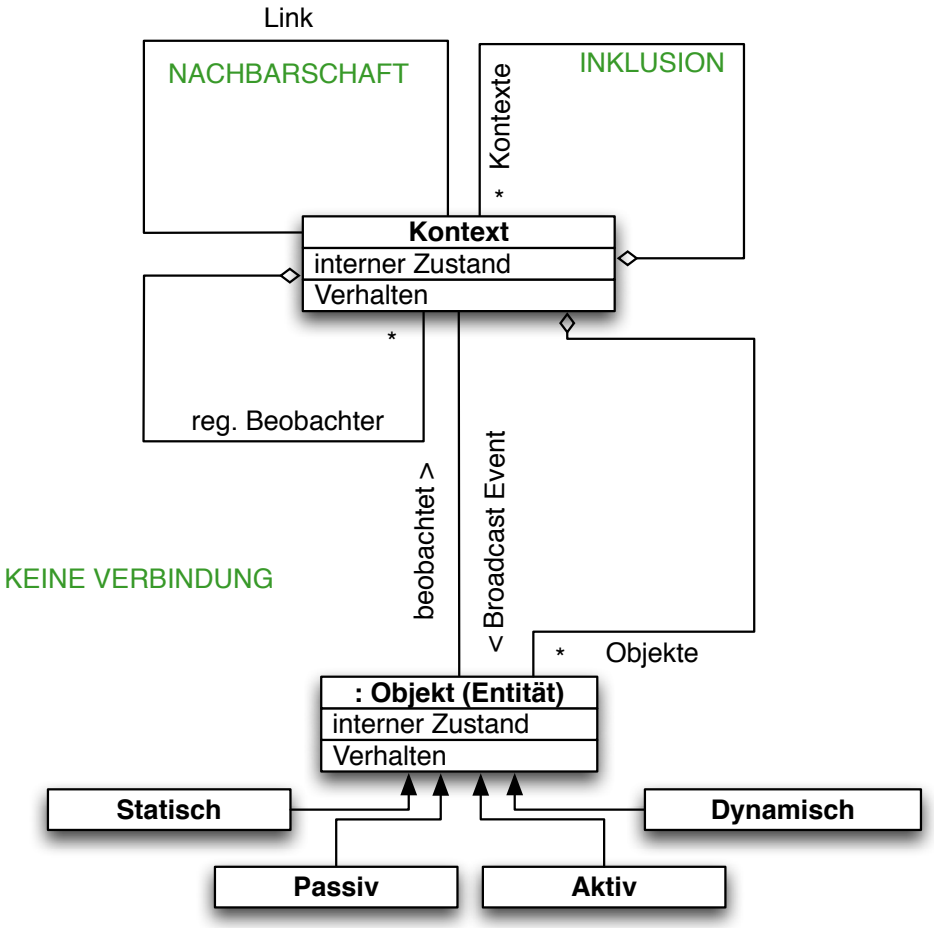


**Abbildung 3.1:** Entwurfsmodell Hyperstories [*eigene Darstellung*]

### 3.2.2 Audio3D

Audio3D ist ein XML basiertes Format zur deklarativen Beschreibung akustischer Umgebungen für 3D Auditory Displays. Audiodesigner können mit dem von Hoffmann, Dachselt und Meissner [HDM03] entwickelten Modell 3D-Audio-Szenen unabhängig von Programmierern, Plattformen und APIs erstellen.
Der hierarchische, azyklische Szenengraph ihres Modells gewährt eine High-Level-Sicht auf akustische Szenen und ermöglicht es, diese sowohl in Echtzeit zu generieren, als auch vorzurendern. Die Knoten des Graphen, welcher an der Wurzel „AudioScene“ beginnt (siehe Abbildung 3.2, Seite 24) sind in Gruppen, den sogenannten „AudioGroups“ und Untergruppen organisiert. Sofern diese einen Transformationsknoten enthalten, wirken sich alle Änderungen auf die

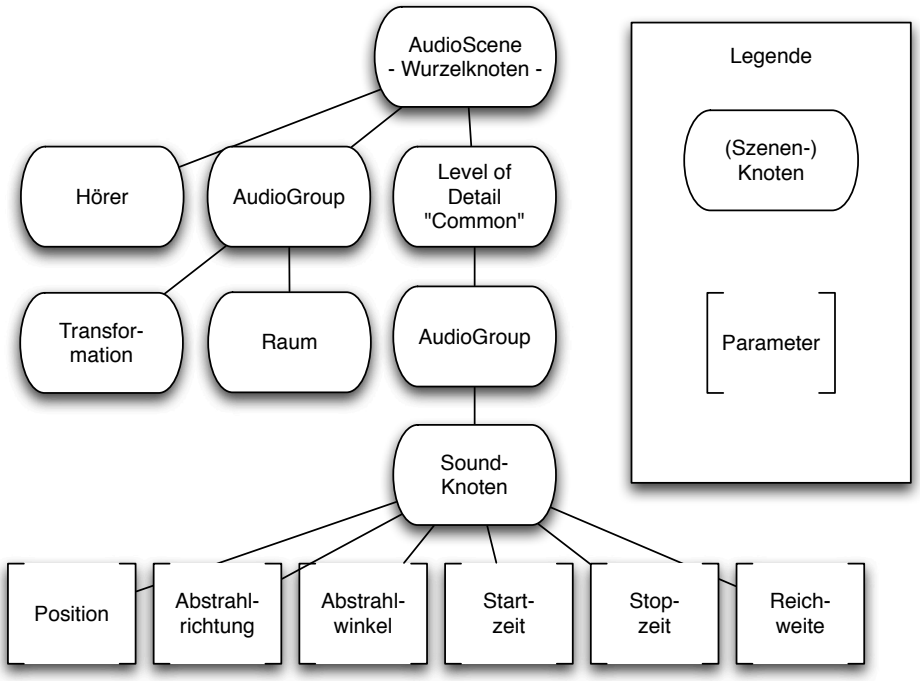


**Abbildung 3.2:** Ausschnitt Szenengraph des Audio3D-Modells [*eigene Darstellung*]

untergeordneten Knoten aus. Beispielsweise führt das Festlegen einer neuen Position innerhalb einer AudioGroup dazu, dass rekursiv für alle Unterknoten die Positionsänderung berechnet wird (siehe Abschnitt 2.1.4, Seite 6).
Im Modell von Hoffmann, Dachselt und Meissner lassen sich sogenannte „Level-of-Detail (LOD)“ Knoten anlegen, denen man AudioGroups unterordnen kann. Dadurch lässt sich festlegen wie detailliert die Soundberechnung durchgeführt werden soll. Weil Klänge mit zunehmender Distanz immer stärker gedämpft werden, gilt allgemein: Je höher die Enfernung einer Schallquelle zum Hörer ist, desto geringer kann das zu berechnende Detaillevel sein.
Audio3D unterscheidet zwischen den drei Level-of-Details „Core“, „Common“ und „Full“.
LOD Core bietet lediglich ein Grundgerüst, um die notwendigsten Klangparameter festzulegen. Dazu zählen Transformation, Distanzdämpfung, Dopplereffekt, Hörerposition und Animation.
Den Detaillevel Core kann man durch den LOD Common erweitern, welcher es ermöglicht, raumakustische Eigenschaften vorzugeben. Damit lassen sich beispielsweise Absorption, Hall und Öffnungen zwischen Räumen simulieren.
Um akustische Umgebungen zu beschreiben, definiert Audio3D im Common-Level einige Hallbereiche mittels geometrischer Primitiven wie Würfeln, Zylindern oder

Kugeln. Soll einer dieser Bereiche als geschlossener Raum fungieren, so muss der Unterknoten „Walls“ (Wände) dem Raumknoten zugeordnet werden. „Walls“ enthält Parameter, welche die Dämpfungsrate für niedrige und hohe Frequenzen festlegen. Dadurch können einem Hörer, der sich innerhalb des Raumes befindet, Schallquellen von außerhalb realistisch simuliert werden. In diesem Fall wird sowohl der Direktschall der außen befindlichen Quelle, als auch der Hall innerhalb des Raumes gedämpft.
Der LOD Full kann durch die parametrische Beschreibung von Einzelreflexionen und Eigenschaften des Ausbreitungsmediums den Grad an Realismus der Simulation nochmals steigern. Dies ist jedoch mit einem erheblichen Rechenaufwand verbunden, weshalb dieser Detaillevel nur sparsam eingesetzt wird.

### 3.2.3 Anforderungsevaluation der Modelle

| | Hyperstories | Audio3D |
|---|---|---|
| 3D-Positionierbarkeit | *** | *** |
| Schallquellenunabhängigkeit | *** | *** |
| Ubiquität | *** | *** |
| Mehrbenutzerfähigkeit | * | *** |
| Koppelbarkeit | - | *** |
| **Gesamtpunkte** | **10** | **15** |

**Tabelle 3.1:** Vergleich zwischen Hyperstories und Audio3D

In Tabelle 3.1, auf Seite 25 wurden das Hyperstories- und Audio3D-Modell verglichen.
„***“ bedeutet dabei, dass die Anforderung vollständig erfüllt ist.
„**“ steht für zum Teil erfüllt und „*“ kennzeichnet ein nicht erfülltes Kriterium.
Wenn in den Quellen zu einem Modell keinerlei Informationen bezüglich der Anforderung zu finden waren, ist dies mit „-“ in der Tabelle markiert.

Am „Hyperstories“-Modell [LS99] ist vor allem positiv hervorzuheben, dass sich die wenigen vorhandenen Basisklassen zu komplexen Umgebungen kombinieren lassen (*Ubiquität*).
Ein ubiquitäres 3D-Sound-System, welches sich über ein gesamtes Gebäude erstreckt, ist mit der Metapher „Inklusion“ (siehe Abbildung 3.1, Seite 23) realisierbar.
Ein „Kontext“ würde einen dreidimensionalen Raum darstellen, in welchem sich Objekte wie Schallquellen und ein einzelner Hörer (Spielfigur) bewegen (*3D-Positionierbarkeit* und *Schallquellenunabhänigkeit*). Aneinander angrenzende Räume sind mit einem Link verknüpft. Der Link ist beispielsweise eine Tür, welche die Räume miteinander verbindet. Bei Lumbreras und Sánchez wird diese Beziehung der Kontexte (Räume) mit der Metapher „Nachbarschaft“ bezeichnet und modelliert zum Beispiel einen Flur mit angrenzenden Zimmern. Ein Kontext inkludiert wiederum diese zusammenhängenden Räume, um das Gebäude zu vervollständigen.
Negativ ist am „Hyperstories“-Modell hervorzuheben, dass es nur für einen Nutzer ausgelegt ist (*Mehrbenutzerfähigkeit*).
Bezüglich der Ausrichtung von diskreten Schallquellen auf den einzelnen Nutzer werden in der Arbeit von Lumbreras und Sánchez [LS99] keine Aussagen getroffen (*Koppelbarkeit*).

Mit dem „Audio3D“-Modell [HDM03] ist realisierbar, dass der „AudioScene“-Knoten (des Szenengraphen) ein Gebäude darstellt und alle Etagen davon mittels „AudioGroups“, bestehend aus mehreren Räumen umgesetzt werden (*Ubiquität*).
Jedem Raum lässt sich pro Nutzer ein „Hörer“-Knoten (des Szenengraphen) unterordnen, der diesen repräsentiert (*Mehrbenutzerfähigkeit*).
Soll ein Hörer über eine bestimmte Anzahl an speziell auf ihn ausgerichteten Schallquellen verfügen, so wird ihm diese Anzahl an „Sound“-Knoten (des Szenegraphen) zugeordnet (*Koppelbarkeit*).
Jedem Raum lassen sich beliebig viele Schallquellen hinzufügen, die alle über einen eigenen Vektor verfügen, der ihre Position im dreidimensionalen Raum beschreibt (*3D-Positionierbarkeit* und *Schallquellenunabhängigkeit*).

Die Evaluation der Modelle hat ergeben, dass Hoffmann et al. den besser geeigneten der beiden Ansätze bietet, da „Audio3D“ alle gestellten Anforderungen vollständig erfüllt und sich somit uneingeschränkt für die Konzeption eines ubiquitäres 3D-Sound-Systems eignet.

## 3.3 APIs und Frameworks

Um einen Prototypen für ein 3D-Sound-System zu implementieren, ist es sinnvoll auf eine bereits bestehende API zu setzen, welche dem Programmierer einen Teil der Arbeit abnimmt beziehungsweise diese vereinfacht.
Da in dieser Arbeit ein Prototyp eines solchen Systems für einen Raum mit zwei Soundsystemen umgesetzt werden soll, erfolgt in diesem Abschnitt eine Beschreibung und anschließend eine Evaluation bestehender Audio-Frameworks.
Um einen realistischen Klangeindruck zu erzielen, sollte man möglichst viele Lautsprecher verwenden, um den zu interpolierenden Bereich zwischen diesen zu verringern. In dieser Arbeit kommen zwei 7.1 Soundsysteme mit insgesamt vierzehn Lautsprechern und zwei Subwoofern zum Einsatz. Die Ansteuerung von acht diskreten Audiokanälen erfordert eine Bandbreite, die sich mittels optisch-(S/PDIF) und koaxial-digitaler Audioübertragung nicht erzielen lässt. Deshalb wurde das in dieser Arbeit entwickelte 3D-Sound-System mit PC-Soundkarten aus dem Gamingbereich geplant, die über einen HDMI-Ausgang zur Audioübertragung verfügen. Dieser liefert ausreichend Bandbreite, um die Audiosignale von acht diskreten Kanälen unkomprimiert zum AV-Receiver zu übertragen. Die uneingeschränkte Nutzung der HDMI-Soundkarten erfordert momentan Windows, da nur auf dieser Plattform die notwendigen Treiber bereitgestellt werden. Aus diesem Grund sollte die Audio-API vollständig kompatibel zu Windows 7 sein.
Zudem wäre es angebracht, wenn die Unterstützung von mehr als einer Soundkarte von vornherein innerhalb der API vorgesehen ist, um Synchronisierungsprobleme zu minimieren.
Ein vorbereiteter Netzwerksupport des Audio-Frameworks wäre eine notwendige Voraussetzung für eine spätere Erweiterung des Prototypen auf mehr als einen Raum.
Da eine gut dokumentierte API die Einarbeitungszeit verkürzt und eine schnelle Hilfestellung bei auftretenden Problemen bietet, ist auch die Dokumentation ein wichtiges Kriterium bei der Auswahl eines passenden Frameworks.

### 3.3.1 OpenAL

OpenAL [Cre] wurde von Creative Labs vorangetrieben, um Spieleentwicklern ein einheitliches Framework zur Verfügung zu stellen, gegen dass sie plattformübergreifend 3D-Sound-Umgebungen programmieren können.
Dafür stellt es die Basisobjekte, wie Schallquellen, Audiopuffer und einen Hörer zur Verfügung. Audiopuffer werden genutzt, um Sounds zwischenzuspeichern. OpenAL stellt Methoden zur Verfügung, um beispielsweise diese Audiopuffer an

Schallquellen zu binden oder den Hörer im Raum zu platzieren.
Ein Kontext definiert einen Bereich, in dem die Schallquellen in Relation zum Hörer gerendert werden. Er ist fest an die Ausgabe einer Soundkarte gebunden, wobei maximal acht Kanäle (7.1) unterstützt werden.

**Relevanz für diese Arbeit**
OpenAL stellt lediglich ein minimales Grundgerüst zur Verfügung und setzt auf Vertrautheit in der Low-Level-Programmierung. Es existieren zwar Bindings für Hochsprachen, wie beispielsweise Tao [Tao] für Mono/.NET oder JOAL [JOA] für Java, welche die Benutzung etwas verbessern, jedoch die Arbeit mit OpenAL nicht wesentlich angenehmer machen. Die unzureichende Dokumentation bietet zudem keinen leichten Einstieg in die API.
Das größte Problem ist jedoch, dass standardmäßig nur ein aktiver Kontext vorhanden sein kann, welcher auf eine bestimmte Soundkarte festgelegt ist. Da dieser maximal acht Ausgabekanäle unterstützt und die Kombination mehrerer Kontexte nicht vorgesehen ist, wird von der Verwendung von OpenAL zur Implementierung des Prototypen dieser Arbeit abgesehen.

### 3.3.2 CREATE Signal Library (CSL)

Die CREATE Signal Library [Pop] ist ein flexibles C++ OpenSource-Framework, welches gleichermaßen sowohl zur Soundsynthese, als auch zur Verarbeitung von Sounds genuzt werden kann. Es wird vorwiegend unter MacOSX verwendet, lässt sich jedoch theoretisch auch für Linux und Windows kompilieren.
Durch seine dreigeteile Architektur, bestehend aus jeweils einer Softwarekomponente zur Soundeingabe, zur räumlichen Verteilung und zur Steuerung, wurde der Einsatz in verteilten Systemen schon bei der Entwicklung von CSL vorgesehen und lässt sich dementsprechend gut in der Praxis realisieren.
Die wichtigsten 3D-Sound-Panning-Methoden, wie Ambisonics, Vektor Basiertes Amplituden Panning (VBAP) und Wellenfeldsynthese (WFS) sind bereits standardmäßig im Framework implementiert.
Für die Anzahl der unterstützten Ausgabekanäle existiert keine Limitierung innerhalb des Frameworks. Somit lassen sich in der Theorie beliebig viele Lautsprecher ansteuern.
CSL ist sehr modular aufgebaut und leicht erweiterbar, da alle Softwarekomponenten durch einen DSP („Digital Sound Processing“)-Graphen miteinander verknüpft sind. Eine Kette aufeinanderfolgender Operationen wird durch einen „Pull“ am Wurzelknoten realisiert. Dieser ist im einfachsten Fall direkt mit der Ausgabeschnittstelle der Soundkarte verknüpft. Der Takt, in dem die „Pulls“ durchgeführt werden, ist dann mittels *Samplingrate dividiert durch Größe des Ausgabepuffers* direkt

vom Treiber vorgegeben.

**Relevanz für diese Arbeit**
Der in CSL verfolgte Ansatz mittels DSP-Graphen stellt einen sehr flexiblen und durchdachten Weg dar, mit dem es durchaus denkbar ist, ein ubiquitäres 3D-Soundsystem umzusetzen.
Die Windowskompatibilität von CSL ist nur unzureichend vorhanden. Eine Kompilierung des Frameworks ist nach vielen Anpassungen zwar möglich, jedoch sind einige wichtige Bestandteile nicht lauffähig. Nach mehreren Tests wurde klar, dass CSL ohne zeitäufwändige Änderungen nicht vollständig auf Windows 7 portierbar ist. Es vermittelt im Bezug auf diese Arbeit zwar einige inspirierende Ansätze, wie z.B den DSP-Graphen, ist aber praktisch nicht verwendbar.

### 3.3.3 jReality

jReality [TU ] ist ein, von der TU Berlin entwickeltes, OpenSource-Java-Framework zur Programmierung von Visualisierungen und Sonifizierungen. Das Rendering von Grafik und Sound wird durch die Abarbeitung eines Szenengraphen (siehe Abschnitt 2.1.4, Seite 6) erreicht und bietet somit vielfältige Modularisierungsmöglichkeiten.
Der Audioteil unterstützt sowohl die Verarbeitung von Eingangssignalen über die Audioeingänge von Soundkarten, als auch Signale, die aus anderen Programmen umgeleitet wurden. Zudem können Klänge direkt mittels softwareseitiger Soundsysthese generiert werden.
3D-Sound ist in jReality bereits vorgesehen, indem alle Klanginformationen der Audioausgabe in 3D-Audiostreams abgemischt werden. Das dieser Kodierung zugrundeliegende Verfahren nennt sich „Ambisonics“.
Das Dekodieren kann mit einem eigenständigen Hardwaredekoder oder über das „ambdec“-Plugin [ZA] erfolgen. Um diese softwarebasierte Dekodierung zu nutzen, muss der im Ambisonics-Format kodierte Ausgabestrom von jReality an den „JACK-Audio-Router“ [JAC], einem Programm zur Umleitung von Audioein- und ausgaben, weitergereicht werden. Dort wird der kodierte Audiostrom von „ambdec“ dekodiert und an die Soundkarte oder ins Netzwerk zur Ausgabe weitergeleitet. Durch die Möglichkeit der Verteilung von Audioströmen mittels „JACK-Audio-Router“ über ein Netzwerk wird auch die Realisierung eines ubiquitären 3D-Sound-Systems ermöglicht.

**Relevanz für diese Arbeit**
Die Verwendung eines Szenengraphen, die Unterstützung einer Vielzahl an 3D-Formaten und der bereits im Framework enthaltenen realistischen 3D-Sound-

kodierung durch Ambisonics ermöglicht einen guten Einstieg in die Programmierung virtueller Realitäten beziehungsweise 3D-Soundumgebungen. Die Dekodierung des Ambisonicstreams kann mit Hardwaredecodern oder zum Beispiel mit den kostenfreien JACK-Plugin ambdec [ZA] erfolgen. Dieses wurde jedoch nur für Linux und MacOSX umgesetzt, und eine Portierung auf ein Windows System gestaltet sich schwierig. Das Gleiche gilt für jjack [Gul], ein Framework, welches Java-Anwendungen erlaubt, den JACK-Audio-Router zu benutzen. Aufgrund der Windowsinkompatibilität kostenfreier Plugins und der ohne teure Ambisonics-Hardwaredecoder unmöglichen Umsetzung des 3D-Sound-Systems wird auf die Nutzung von jReality in dieser Arbeit verzichtet.

### 3.3.4 BASS Audio Library

Die BASS Audio Library [Un4] wurde vom Entwicklerteam „un4seen“ zusammengestellt und ist für nichtkommerzielle Projekte frei verfügbar. Sie bietet Programmieren ein breites Spektrum an Funktionen zur Audioverarbeitung. Die dreidimensionale Ausrichtung von Sounds im Raum ist dabei ebenso vorbereitet wie die Unterstützung möglichst kleiner Puffer für Echtzeitanwendungen.
Desweiteren ist die Nutzung mehrerer Soundkarten zur gleichen Zeit sowie der Austausch von Sounddateien zwischen diesen explizit vorgesehen.
Die BASS Audio Library ist in 32- und 64-Bit-Versionen für Programmiersprachen wie C++, C#, Java, Delphi und viele weitere verfügbar. Es gibt fertige Bibliotheken für Linux, MacOSX und Windows. Zudem werden die etabliertesten mobilen Plattformen wie iOS, Android und WinCE unterstützt.

**Relevanz für diese Arbeit**
Aufgrund der vorgesehenen, gleichzeitigen Unterstützung meherer Soundkarten und der guten Windowskompatibilität ist ein 3D-Soundsystem mit der BASS Audio Library umsetzbar.
Es stehen zudem viele Plugins zur Verfügung, welche die API mit zusätzlichen Dateiformaten, wie z.B. FLAC, umgehen lassen oder das Ansprechverhalten der Soundhardware durch WASAPI beziehungsweise ASIO verbessern.

### 3.3.5 XNA

Microsoft XNA [Mic] bildet ein Framework, das die Schnittstellen zur Grafik-, Audio- und Eingabeprogrammierung für Spieleentwickler als kostenfreies Gesamtpaket zur Verfügung stellt.

Mit XNA Game Studio Express wird eine komplette Entwicklungsumgebung für Visual C# bereitgestellt, um Anwendungen für Windows, XBOX360 und Windows Phone zu implementieren.
Die Audioentwicklung in XNA weist eine Besonderheit gegenüber den meisten Audio APIs auf. Durch eine Trennung von Design und Programmierung ist es möglich, Sounddesigner und Programmierer unabhängig voneinander, parallel an einem Projekt arbeiten zu lassen. Der Designer stellt Klangsammlungen, sogenannte „Soundbanks“ inklusive ausgewählter Audiosamples („SoundCues“) für zum Beispiel eine *Explosion* zusammen, legt Parameter wie Position im Raum, Pitch oder Gain fest und kompiliert das Ergebnis. Der Programmierer bindet an entsprechender Stelle die Soundcue *Explosion* ein und zur Laufzeit wird bei jedem Aufruf von *Explosion* ein randomisierter Sound dieser Cue abgespielt.

**Relevanz für diese Arbeit**
Die Trennung von Design und Programmierung gestattet es, Soundbibliotheken anzupassen, ohne den eigentlichen Programmcode zu verändern. Dies ermöglicht es auch Laien, komplexere 3D-Soundlandschaften in ein vorhandenes Projekt einzubinden. Zudem ist der Einstieg in XNA durch die Verwendung eines Komplettpakets mit allen benötigten Werkzeugen und einer guten Onlinedokumentation einfach gestaltet und sorgt für eine flache Lernkurve. Nachteil für die Audioprogrammierung ist der begrenzte Funktionsumfang zur Verarbeitung von Sounds.

### 3.3.6 Anforderungsevaluation der Audio-APIs

| | OpenAL | CSL | jReality | BASS | XNA |
|---|---|---|---|---|---|
| 3D-Positionierbarkeit | *** | *** | *** | *** | *** |
| Schallquellenunabhängigkeit | *** | *** | *** | *** | *** |
| Ubiquität | * | *** | ** | *** | *** |
| Mehrbenutzerfähigkeit | * | *** | *** | ** | ** |
| Anpassbarkeit | * | *** | ** | *** | ** |
| Praktikabilität | * | ** | * | ** | *** |
| **Gesamtpunkte** | **10** | **17** | **14** | **16** | **16** |

**Tabelle 3.2:** Vergleich der vorgestellten Audio-APIs

In Tabelle 3.2, auf Seite 32 ist der Vergleich der Audio-APIs im Überblick dargestellt.
„***“ bedeutet dabei, dass die Anforderung vollständig erfüllt ist.
„**“ steht für zum Teil erfüllt und „*“ kennzeichnet ein nicht erfülltes Kriterium.

Die vorgestellten APIs wurden alle zur Programmierung von Audioanwendungen entwickelt. Jedoch ist lediglich die Verwendung von „CSL“, „BASS“ und „XNA“ zur Implementierung von 3D-Sound-Systemen geeignet, da sie mit mehreren Soundkarten umgehen können, für nicht-kommerzielle Projekte kostenlos sind und dem Entwickler keine Folgekosten für Dekoderplugins auferlegen.
CSL kann nahezu alle evaluierten Anforderungen an ein ubiquitäres 3D-Sound-System in vollem Umfang erfüllen, ist jedoch nicht ohne zeitaufwendige Änderungen auf die Windows-Plattform anpassbar. Da, wie eingangs erwähnt, eine vollständige Windows 7 Kompatibilität aufgrund der Soundkartentreiber erforderlich ist, wird von der Verwendung von CSL zur Implementierung des Prototypen abgesehen.
Somit verbleiben BASS und XNA in der engeren Auswahl.
Da XNA die beste Windowsintegration aller verglichenen APIs aufweist sowie aufgrund der guten Dokumentation eine hohe Praktikabilität erreicht, bekommt es den Vorzug gegenüber BASS und wird für die Implementierung des Prototypen in dieser Arbeit verwendet.

## 3.4 Zusammenfassung

In diesem Kapitel wurde ein Überblick verwandter Arbeiten vorgestellt, denen unterschiedliche Ansätze zur Entwicklung eines 3D-Sound-Systems zugrunde liegen. Besonders die Arbeit „Sound Element Spatializer“ von Ryan Michael McGee ist aufschlussreich, da durch sie erstmals eine Graphenstruktur für diese Arbeit in Betracht gezogen wurde.
Zudem wurden Modelle aufgezeigt, auf deren Basis sich ein Raummodell für ubiquitäre 3D-Sound-Systeme entwickeln lässt. „Audio3D“ ist hier besonders hervorzuheben. Der Szenengraph des von Hoffmann et al. entwickelten Modells liefert aufgrund seiner klaren und übersichtlichen Struktur die Grundlage für das Raummodell dieser Arbeit.
Im folgenden Kapitel wird die Konzeption des ubiquitären 3D-Sound-Systems vorgestellt, bestehend aus dem entwickelten Raummodell, Benutzermodell, Soundsystemmodell und der zugrundeliegenden Architektur.

# 4 Konzeption des 3D-Sound-Systems

Im Rahmen dieser Arbeit wird ein Modell für ein ubiquitäres 3D-Sound-System entwickelt. Die folgenden vier Teilaspekte werden dabei gesondert behandelt:

(1) Durch ein **Raummodell** erfolgt die Spezifikation einer geeigneten Repräsentation für die Beschreibung der Schallquellen im Raum.
(2) Mittels einem **Benutzermodell** wird sowohl die Positionierung und Bewegung von Benutzern im Raum, als auch individuelle Lautstärkeeinstellungen und die Berechnung eines gewichteten Sweetspots dargestellt.
(3) Ein **Soundsystemmodell** gibt den Rahmen vor in dem Einzelkomponenten, wie Soundkarten und Sound-Systeme, inklusive der entsprechenden Verbindung zu einem 3D-Sound-System, verwendet werden.
(4) Die **Architektur** vereint die aufgestellten Modelle und veranschaulicht deren Zusammenhänge.

## 4.1 Raummodell

Das in dieser Arbeit entwickelte ubiquitäre 3D-Sound-System ermöglicht es, virtuelle Schallquellen in einem physischen Raum zu platzieren und zu bewegen. Klang kann dabei Hörern folgen oder ihnen Wege zeigen. So entsteht eine Abhängigkeit zwischen Schallquelle und Hörer bezogen auf ihre Position innerhalb des Raumes. Da sich ein ubiquitäres 3D-Sound-System (siehe Abschnitt 2.1.3, Seite 6) aus mehreren Räumen zusammensetzen kann, sollte es möglichst modular aufgebaut sein und sich einfach erweitern lassen. So kann das auf einer Etage installierte 3D-Sound-System später ohne größere Umstände für den Einsatz in einem kompletten Gebäude angepasst werden.
Ausgangsbasis für die Entwicklung eines Raummodells waren die Arbeiten „Interactive 3D Sound Hyperstories for Blind Children“ von Lumbreras und Sánchez (siehe Abschnitt 3.2.1, Seite 22) sowie „Audio3D“ von Hoffmann, Dachselt und Meissner (siehe Abschnitt 3.2.2, Seite 23). Das von Lumbreras und Sánchez aufgestellte Modell kommt in dieser Arbeit jedoch nicht zum Einsatz, da es sich eher für Umgebungen mit vielen autonomen, unabhängigen Objekten eignet, die

eventbasiert miteinander kommunizieren können.
Der in „Audio3D“ verfolgte Ansatz eines Szenengraphen bietet eine übersichtliche hierarchische Struktur. Zudem ist er effizient, da nur die erfolgten Änderungen rekursiv ausgewertet werden. Wenn sich beispielsweise die Position eines Objekts im Raum ändert, so wird diese Änderung auf alle ihm untergeordneten Objekte übernommen. Dies ist in dieser Arbeit vor allem dann der Fall, wenn „gekoppelte Schallquellen“ einem Hörer untergeordnet sind, der seinen Aufenthaltsort ändert.
Die Erkenntnisse aus Audio3D wurden aufgegriffen und bilden die Grundlage des im Rahmen dieser Arbeit entwickelten Raummodells (siehe Abbildung 4.1, Seite 37).
Im Gegensatz zu bisherigen 3D-Sound-Systemen, welche sich meist auf einen Raum beschränken, ist das in dieser Arbeit entwickelte Raummodell so flexibel strukturiert, dass es sich auch über komplette Gebäude erstrecken kann, um den Nutzer an jedem Ort darin mit akustischen Informationen zu versorgen.

### 4.1.1 Szenengraph

Ein Szenengraph bildet die Grundordnung des Raummodells. Die hierarchische Struktur des Modells macht es übersichtlich und leicht verständlich. Zudem gewährleistet diese Hierarchie eine einfache Erweiterbarkeit.
Jeder Knoten im Graphen erbt von dem Objekt „Szenenknoten“ um zu gewährleisten, dass wichtige Attribute wie Position und eine Liste der Unterknoten sowie eine Update-Methode implementiert sind, da der Szenengraph diese rekursiv abfragt. Durch die Vererbung wird sichergestellt, dass der Graph von der Wurzel bis zum tiefsten Blatt durchlaufen werden kann.
Wird ein neues Objekt erstellt, erbt diese von „Szenenknoten“ oder einer Unterklasse davon und wird danach an der zu expandierenden Stelle in den Graphen eingegliedert sowie durch weitere Unterknoten näher definiert. In den automatisch ablaufenden Updatezyklen des Graphen werden die neuen Objekte unmittelbar mit einbezogen, da sie die Updatemethoden eines Szenenknoten geerbt haben.
Das Modell bietet dadurch nicht nur einfache Erweiterbarkeit, es lassen sich auch bestimmte Probleme von Anfang an in Teilaspekte gliedern, die dann beispielsweise von unterschiedlichen Entwicklern umgesetzt werden können. In dieser Arbeit erfolgt zum Beispiel eine Ausgliederung der Hörerverwaltung in ein eigenes Benutzermodell (siehe Abschnitt 4.2, Seite 42) basierend auf Szenenknoten, wodurch sich das Benutzermodell nahtlos in den Szenengraphen integrieren lässt.
„Gebäude“, „Verband“ und „Raum“ stellen eine Basisgliederung des Modells in Form einer Baumstruktur dar (siehe Abbildung 4.1, Seite 37). Die Ergänzung der ebenso wichtigen Knoten „Hörer“ und „Schallquelle“ im Graphen und die

Verfeinerung durch weitere Knoten, die zur genauen Beschreibung des Raumes beitragen, bilden die Grundlage zur effizienten Simulation komplexer akustischer Umgebungen. „Hörer“ steht dabei symbolisch für das im nächten Kapitel erläuterte Benutzermodell.
Alle vordefinierten Knotentypen werden im Folgenden beschrieben.

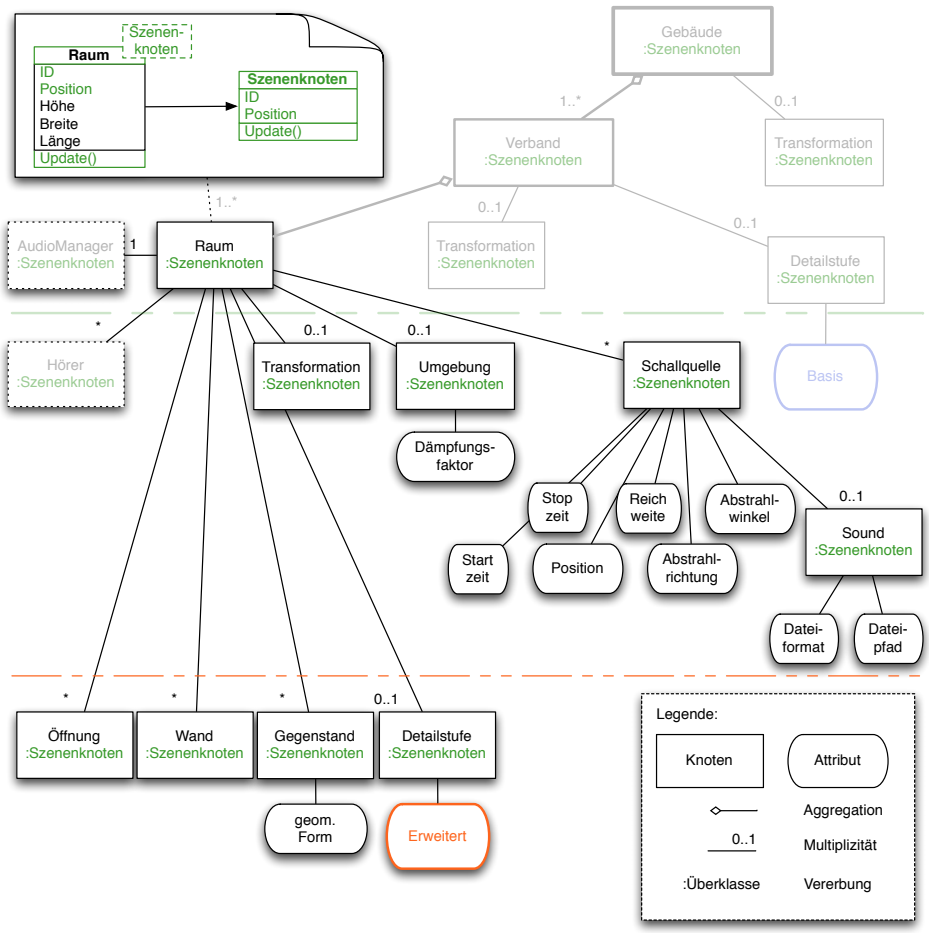


**Abbildung 4.1:** Das hierarchische Raummodell inklusive seiner Grundelemente [*eigene Darstellung*]

**Gebäude** Der Gebäudeknoten bildet die Wurzel des Szenengraphen. Er aggregiert mindestens einen Verbandsknoten. Diese Aggregation unterliegt keinerlei Begrenzung hinsichtlich der Anzahl der maximalen Knoten, womit sich beliebig komplexe Raumkombinationen erstellen lassen. Ein ubiquitäres 3D-Sound-System, welches sich über ein komplettes Gebäude erstreckt, ist nach diesem Modell somit problemlos entwickelbar.

**Hörer** Dieser Punkt steht symbolisch für das Benutzermodell. Es lässt sich nahtlos als Unterknoten von Verband oder Raum in den Graphen integrieren.
Im Abschnitt 4.2 auf Seite 42 wird das Benutzermodell näher erläutert.

**AudioManager** Für jeden Raum muss ein Audiomanager festgelegt werden. Dieser organisiert die Audioausgabe für diesen Raum. Auf den Audiomanager wird unter 4.4.2, Seite 49 näher eingegangen.

Die folgenden Knoten sind von dem durch Hoffmann et al. [HDM03] beschriebenen Audioszenengraphen abgeleitet. Dabei stellt die Bezeichnung in Klammern hinter dem Knotentyp, in der Form: Baum („Tree“), die entsprechende Benennung von Hoffmann et al. dar.

**Verband („AudioGroup“)** Ein Verband kann mehrere Knoten vom gleichen Typ gruppieren. Mit einem Raumverband lässt sich in dieser Arbeit beispielsweise eine Wohnung beschreiben oder ein Verband von Schallquellen gleichermaßen starten, stoppen oder umpositionieren.
Ein Verband kann an beliebiger Stelle im Baum stehen.

**Raum („Room“)** Er dient dazu, wie auch an der Bezeichnung erkennbar, einen Raum näher zu spezifizieren und ist „Verband“ immer direkt untergeordnet. Um Reflexionen zu simulieren, ist es notwendig die Unterknoten „Öffnungen“ und „Wände“ einzufügen. Wenn Hindernisse im Raum simuliert werden sollen, können diese durch den Unterknoten „Gegenstand“ definiert werden.

**Schallquelle („Source“)** Einerseits kann der „Schallquellen“-Knoten einem „Raum“-Knoten untergeordnet werden, um eine Quelle einem Raum zuzuordnen. Andererseits kann er Tochterknoten von „Verband“ sein, der mehrere Schallquellen gruppieren kann.
Der „Schallquellen“-Knoten sollte mindestens über einen Unterknoten „Sound“ verfügen, welcher die abzuspielende Datei beziehungsweise einen Stream öffnet.
Für jede Quelle können die folgenden Attribute definiert werden:

- **Position** gibt an, wo im dreidimensionalen Raum sich die Schallquelle befindet.
- **Startzeit** legt fest, wann das Abspielen beginnt.
- **Stopzeit** vereinbart einen Zeitpunkt, zu dem die Wiedergabe anhält.
- **Reichweite** definiert, wie weit ein Geräusch maximal zu hören ist.
- **Abstrahlrichtung** legt fest, in welche Richtung sich die Schallwellen ausbreiten.

- **Abstrahlwinkel** gibt an, bis zu welchem Grad Schall aus dieser Quelle abgestrahlt werden kann.

Durch das Festlegen der Reichweite kann man Rechenkosten sparen, da alle Schallquellen, deren Distanz zum Hörer größer als ihre maximale Reichweite ist, nicht berechnet werden müssen.
Die Attribute Abstrahlrichtung und - winkel sind vor allem von Bedeutung, um Reflexionen ersten bis n-ten Grades zu errechnen.

**Umgebung („Environment“)** Dieser Knoten kann „Gebäude“, „Verband“ und „Raum“ untergeordnet werden, um bereichsweit Umgebungsattribute wie beispielsweise den Dämpfungsgrad des Schalls festzulegen. Sobald ein tiefer, im Graphen gelegener Knoten, eine eigene „Umgebung“ definiert, werden Vorgaben höherer Knoten überschrieben.

**Transformation („Transformation“)** Positions- beziehungsweise Größenänderungen können anhand dieses Knotens festgelegt werden und wirken sich auf alle Unterknoten des jeweiligen Einhängepunktes, relativ zu diesem, aus. So ist es zum Beispiel möglich, gezielt eine Schallquelle zu bewegen oder zu drehen, wenn sie einen Transformationsknoten als Unterknoten hat. Es lassen sich auch mehrere Schallquellen gleichzeitig transformieren, sofern diese in einem „Verband“ zusammengefasst werden und dieser einen Transformationsknoten enthält.
„Gebäude“, „Verband“, „Raum“, „Schallquelle“, „Gegenstand“ und „Hörer“ können Erlternknoten von einer Transformation sein.

**Gegenstand („Object“)** Hindernisse im Raum können durch Einhängen von Gegenstandsknoten in den jeweiligen Raum simuliert werden. Dabei lassen sich Attribute wie die Form als geometrische Primitive (Box, Zylinder, Kugel) definieren, welche dann Einfluss auf die Reflexionsrichtung nehmen. Durch Materialdefinitionen kann der Reflexions- und Absorptionsgrad zusätzlich variiert werden.

**Wände („Walls“)** Diese Knoten stecken den Raum ab. Attribute können den Dämpfungsgrad sowohl für hohe als auch niedrige Frequenzen festlegen.
Wenn sich ein Hörer innerhalb und die Schallquelle außerhalb eines geschlossenen Raumes befindet, dann nehmen die Wände entsprechend Einfluss auf die Wahrnehmung des Klanges.

**Öffnung („Opening“)** Durch das Festlegen von Flächen, welche die Öffnungen eines Raumes beschreiben, kann die Wahrnehmung von Klängen aus anderen Räumen simuliert werden.

**Detailstufe („Feature Level“)** Die Detailstufe legt in diesem Modell fest, wie präzise ein „Gebäude“, ein „Verband“ oder ein „Raum“ simuliert wird. Dabei erfolgt eine Unterscheidung zwischen den beiden Stufen „Basis“ und „Erweitert“.

Die **Basisstufe** bietet lediglich ein Grundgerüst, um die notwendigsten Szenengraphenparameter auszuwerten. Dazu zählen Transformationen, Distanzdämpfung, Hörerposition, Schallquellenposition und Animationen.

Die Stufe **Erweitert** ermöglicht es, zusätzlich zur Basisstufe raumakustische Eigenschaften auszuwerten. Absorption und Reflexion von Schall, Verdeckung durch Hindernisse, Hall und Öffnungen zwischen den Räumen lassen sich in dieser Stufe simulieren. Die Auswertung der Geschwindigkeitsparameter von Schall, Hörer und Schallquelle erlaubt es zudem, den Dopplereffekt in dieser Stufe zu simulieren.

Wenn eine Detailstufe einem Szenenknoten „Gebäude“, „Verband“ oder „Raum“ untergeordnet wird, so ist sie global für alle in der Hierarchie tiefer angesiedelten Knoten gültig, kann jedoch von diesen überschrieben werden. Sollte gleichzeitig eine globale Stufe eingestellt sein und die Durchführung einer aufwendigen Simulation in einem Raum umgesetzt werden, so kann man für diesen Raum die Detailstufe „Basis“ festlegen. Dadurch müssen weniger Berechnungen erfolgen, was gerade bei einer großen Anzahl an Schallquellen die benötigte Rechenleistung minimiert. Somit ließe sich auch beispielsweise ein Fliegenschwarm flüssig simulieren, bei dem die Rechenlast bei Verwendung der „Erweitert“- Detailstufe zu hoch und eine flüssige Wiedergabe unmöglich gewesen wäre.
Wenn die Rechenleistung für die höhere Detailstufe nicht ausreichend ist, kann die Detailstufe der Simulation zur Laufzeit durch einen **Fallbackmechanismus** von „Erweitert“ auf „Basis“ herabgestuft werden. Dies ermöglicht eine weiterhin flüssige, jedoch weniger detaillierte Wiedergabe, da zum Beispiel die Berechnung von Absorption und Reflexion nicht mehr durchgeführt wird.
Das in dieser Arbeit entwickelte Raummodell lässt sich durch den Einsatz der Detailstufen flexibel an Vorgaben wie die zur Verfügung stehende Rechenleistung und den gewünschten Grad an Realismus anpassen. Diese Flexibilität des Einsatzes eines solchen Systems, die durch die Verwendung eines Fallbackmechanismus unterstützt wird, hebt sich von bisherigen 3D-Sound-Systemen ab und ist ein Alleinstellungsmerkmal dieser Arbeit.

### 4.1.2 Besonderheiten

Durch das Anlegen von Spezialstrukturen und Bewegungsanimationen lassen sich besondere Klangeffekte erzielen. Zum Erstellen dieser Strukturen werden mehrere Knoten kombiniert und an geeigneter Stelle im Szenengraphen eingehangen.

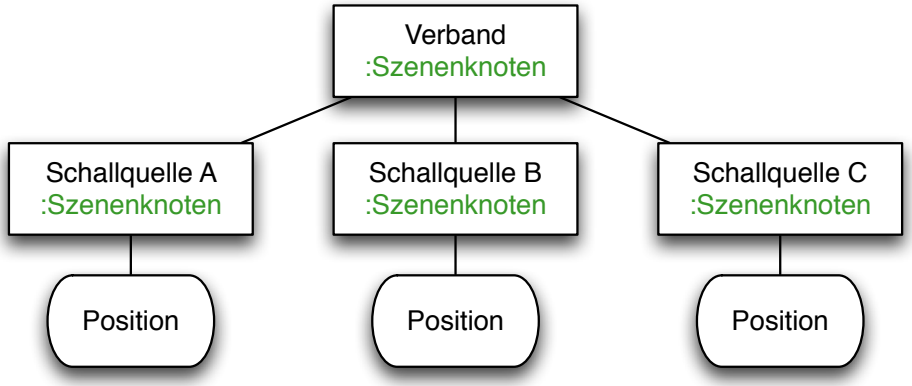


**Abbildung 4.2:** Spezialstruktur bestehend aus 3 Schallquellen [*eigene Darstellung*]

**Spezialstrukturen** Jedem „Raum"- beziehungsweise „Verband"- Knoten lassen sich Spezialstrukturen unterordnen.
Diese gruppieren mehrere Schallquellen mit unterschiedlichen Positionen (siehe Abbildung 4.2, Seite 41). Man könnte beispielsweise eine kleine Musik-Band vor einer Wand eines Raumes simulieren, indem man einen Verband aus drei Schallquellen erstellt. Schallquelle A, der Gitarrist würde links vor der Wand positioniert, Schallquelle B, der Schlagzeuger mittig vor der Wand und Schallquelle C, der Bassist rechts vor der Wand. In kurzer Zeit und mit wenig Aufwand kann auf diese Weise eine virtuelle Konzertbühne erzeugt werden. Konzertsäle, Marktplätze, Theaterbühnen und viele weitere Szenarien lassen sich durch Spezialstrukturen definieren und mit dem 3D-Sound-System simulieren.

**Bewegungsanimationen** Für jede Schallquelle kann genau eine Bewegungsanimation angelegt werden, indem man einen Knoten „Bewegungspfad" erstellt und diesen der gewünschten Schallquelle unterordnet.
Ein solcher „Bewegungspfad" setzt sich aus einer beliebigen Anzahl an „Bewegungspunkten" zusammen, welche durch einen Positionsvektor und einen Zeitpunkt, zu dem die Position erreicht werden soll, definiert werden (siehe Abbildung 4.3, Seite 42).

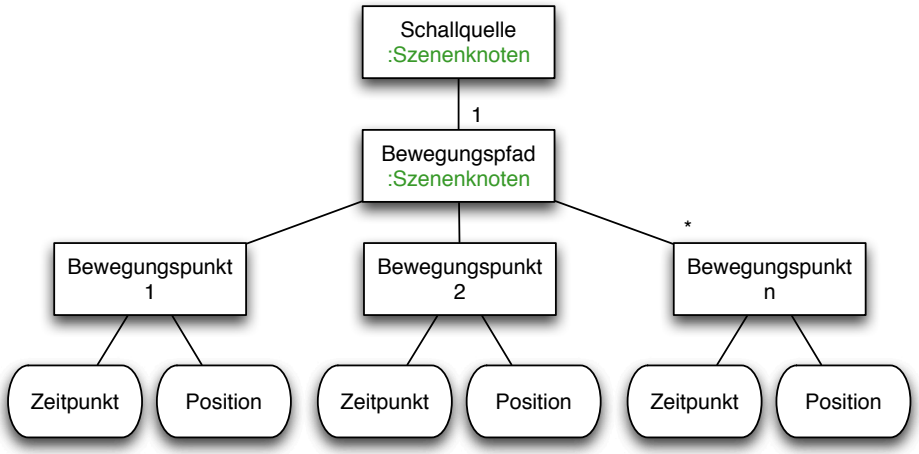


**Abbildung 4.3:** Bewegungsanimation über 3 Positionen [*eigene Darstellung*]

Ein durch den Raum fliegender Vogel oder eine Abfolge von Hinweistönen, die einen Weg anzeigen, können durch Bewegungsanimationen von Schallquellen realisiert werden.
Einmal erstellte Animationen können persistent abgespeichert und für beliebige Schallquellen wieder geladen werden, sofern diese sich auf diesem Pfad bewegen sollen.

Spezialstrukturen und Bewegungsanimationen können nicht nur getrennt benutzt werden. Sie lassen sich auch kombinieren.
Das zuvor genannte Beispiel der Spezialstruktur „Musik-Band“ kann durch Bewegungsanimationen für jedes Band-Mitglied des Verbands erweitert werden, indem man entsprechende Pfade erstellt, diese den Schallquellen unterordnet und somit die Band animiert.
Dieses Baukastensystem führt zu einem hohen Maß an Erweiterbarkeit und Wiederverwendbarkeit von selbst definierten Strukturen in dem in dieser Arbeit vorgestellten Modell.

## 4.2 Benutzermodell

Die Berücksichtigung mehrerer Nutzer in einem Raum steht im Mittelpunkt des entwickelten Benutzermodells. Da in der Literatur keine Modelle gefunden wurden, die eine Mehrzahl an Benutzern betrachten, wird für diese Arbeit ein entsprechendes Modell aufgestellt.

Es wird in diesem Abschnitt näher beschrieben.

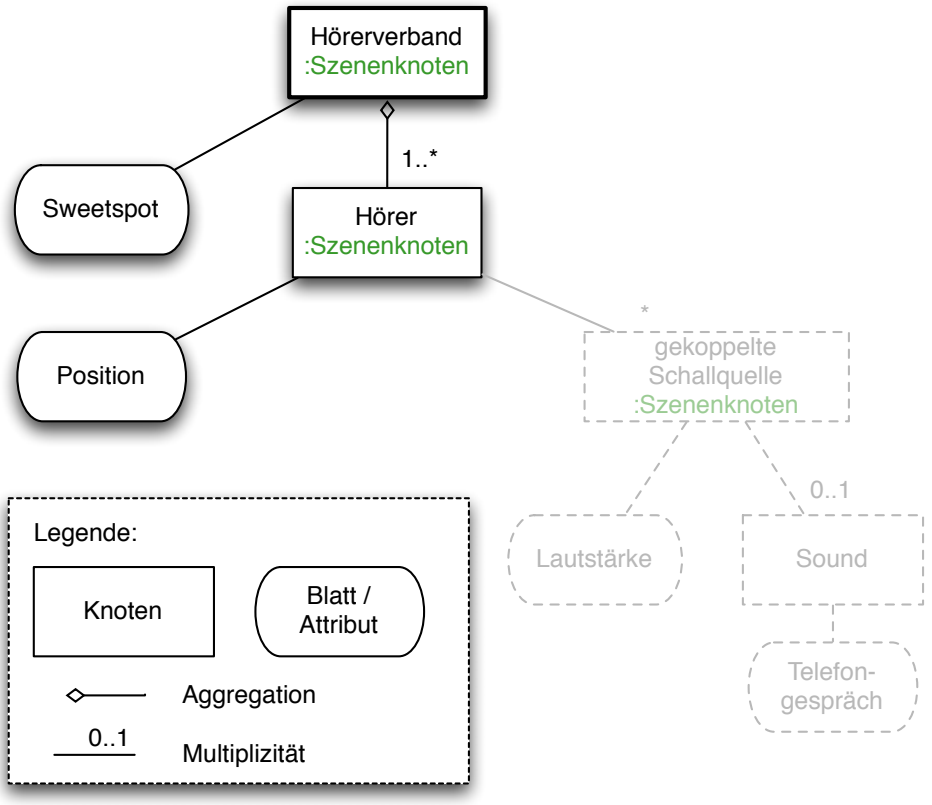


**Abbildung 4.4:** Benutzermodell [*eigene Darstellung*]

Das Benutzermodell basiert ebenfalls auf einer hierarchischen Baumstruktur und lässt sich dadurch nahtlos als Unterknoten von „Verband“ oder „Raum“ in das Raummodell integrieren.
Der Wurzelknoten ist ein spezieller Verband, hier mit „Hörerverband“ bezeichnet. Unter ihm werden alle Benutzer gruppiert, die sich in einem bestimmten Bereich (Raum) aufhalten oder bewegen (siehe Abbildung 4.4, Seite 43).

**Sweetspot** Aus den Positionen aller Hörer wird dynamisch ein gewichteter Mittelwert gebildet, in diesem Modell mit „Sweetspot“ (siehe Abschnitt 2.1.5, Seite 8) bezeichnet. Mit „gewichtet“ ist hier gemeint, dass Experten Faktoren in Bezug auf Schwellwerte definieren können, die festlegen wie stark die Position eines Höres in die Sweetspotberechnung einfließt.
Zur Laufzeit der Simulation muss überprüft werden, ob sich eine Gruppe von Hörern in einem voreingestellten Radius zum Mittelwert ihrer Positionen be-

findet. Sofern sich mehrere Personen in diesem Radius befinden, werden die Abstände von dem Mittelpunkt dieser Gruppe zu den restlichen Benutzern beziehungsweise Mittelpunkten von anderen Nutzergruppen ermittelt (siehe Abbildung 4.5, Seite 4.5). Überschreitet diese Distanz die vordefinierten Schwellwerte, so werden die Positionen der Benutzergruppen (absteigend der Anzahl ihrer Nutzer) mit vordefinierten Faktoren (zwischen 0.0 und 1.0) multipliziert. Die Positionen aller Nutzer werden somit in Abhängigkeit ihrer Gruppierungen im Raum gewichtet. Auf Basis dieser Gewichtung wird der Sweetspot berechnet. Dadurch wird vermieden, dass eine Person, die sich sehr von einer Gruppe entfernt, den Sweetspot zu stark beeinflusst und das akustische Geschehen zu weit von der Hauptgruppe ablenkt.
Dies ist ein Ansatz zur Berechnung der Gewichtung des Sweetspots. Sie kann jedoch beliebig umgesetzt werden oder der Sweetspot ohne Gewichtung berechnet werden. Wenn man darauf verzichtet Positionen der Hörer zu gewichten, kann dies gerade bei vielen Hörern zu einem spürbaren Performancegewinn führen.

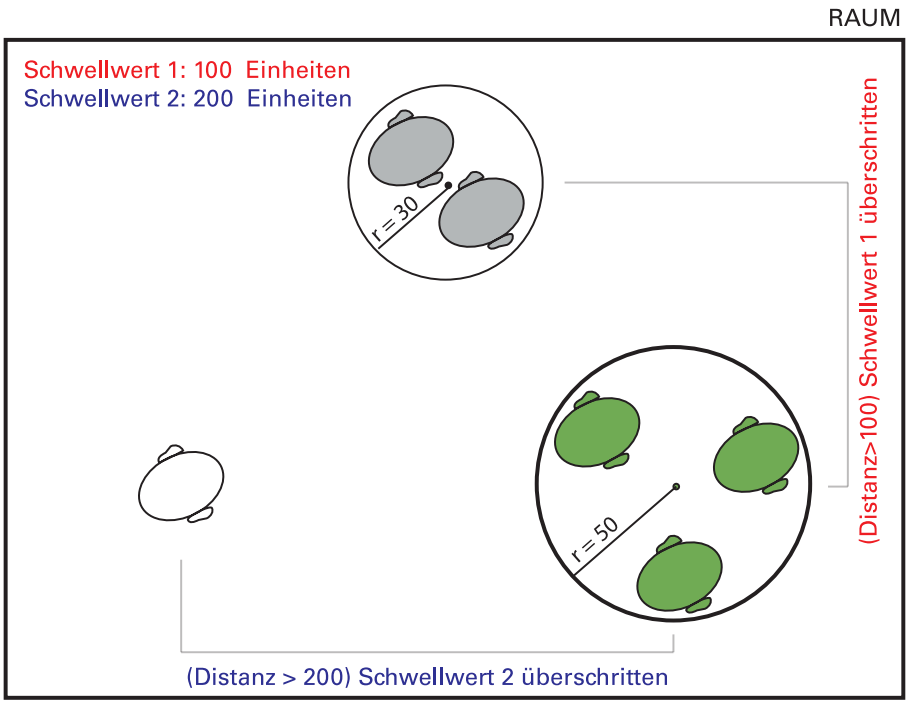


**Abbildung 4.5:** Sweetspotgewichtung - Distanzermittlung zwischen größter Benutzergruppe, einer Gruppe aus 2 Personen und einer Einzelperson [*eigene Darstellung*]

Der Sweetspot stellt einen Positionsvektor des dreidimensionalen Raumes dar, der zum Beispiel zur vereinfachten Abstandsberechnung zwischen der (Haupt-) Benutzergruppe und einer Schallquelle genutzt werden kann. Durch diese Vereinfachung sind nur noch Abstandsberechnungen in Höhe der im Raum beziehungweise Verband befindlichen Schallquellen erforderlich. Dies sollte in Situationen mit vielen Hörern und Schallquellen einen messbaren Performancevorteil darstellen.
Bei einer Gewichtung der Benutzergruppen wäre es auch denkbar mehrere Sweetspots zu berechnen, sofern an verschiedenen Orten im Raum mehrere Personen gruppiert auftreten.
Um Benutzer gezielt mit akustischen Informationen wie eMail- Eingangsbenachrichtigungen oder persönlichen Gesprächen zu versorgen, kann man dem Hörerknoten des gewünschten Nutzers eine „gekoppelte Schallquelle“ unterordnen. Sie ist identisch mit normalen Schallquellen, hält jedoch immer die gleich Distanz zum Hörer. Wenn sich der Hörer bewegt, folgen ihm all seine gekoppelten Schallquellen.
Sofern ein Hörer genug Abstand zu anderen Benutzern im Raum hat, können seine gekoppelten Schallquellen im Pegel gesenkt und so eine Individuallautstärke für diesen Hörer realisiert werden.

## 4.3 Soundsystemmodell

Das Soundsystemmodell beschreibt die Komponenten, welche in dieser Arbeit für das 3D-Sound-System zum Einsatz kommen. Dabei erfolgt eine Unterteilung in folgende sechs Bestandteile: „Raumsimulation“, „Audiomanager“, „System-Audio-API“, „Audio-Renderer“, „Audio-Ausgabegerät“ und „Lautsprecherebene“ (siehe Abbildung 4.6, Seite 46). Diese werden nun im Folgenden erklärt.

**Raumsimulation** Die Raumsimulation ist ein Programm, welches die Nutzereingaben verarbeitet und das Abspielen von Sound organisiert. Dazu zählen vor allem Mediaplayer-Funktionalitäten wie Sounds „laden“, „abspielen“, „pausieren“ oder „verwerfen“. Weiterhin werden zum Beispiel Positionseingaben des Benutzers von der Raumsimulation verarbeitet.

**Audiomanager** Der Audiomanger verwaltet die einzelnen Lautsprecherebenen. Er kann aufgrund der Positionsvektoren und Sounds, die er von der Raumsimulation bekommt, regulieren in welcher Weise jeder Sound auf der jeweiligen Lautsprecherebene ausgegeben werden soll (siehe Abschnitt 4.4.2, Seite 49).
Er hat zudem Zugriff auf die System-Audio-API und kann diese zur An-

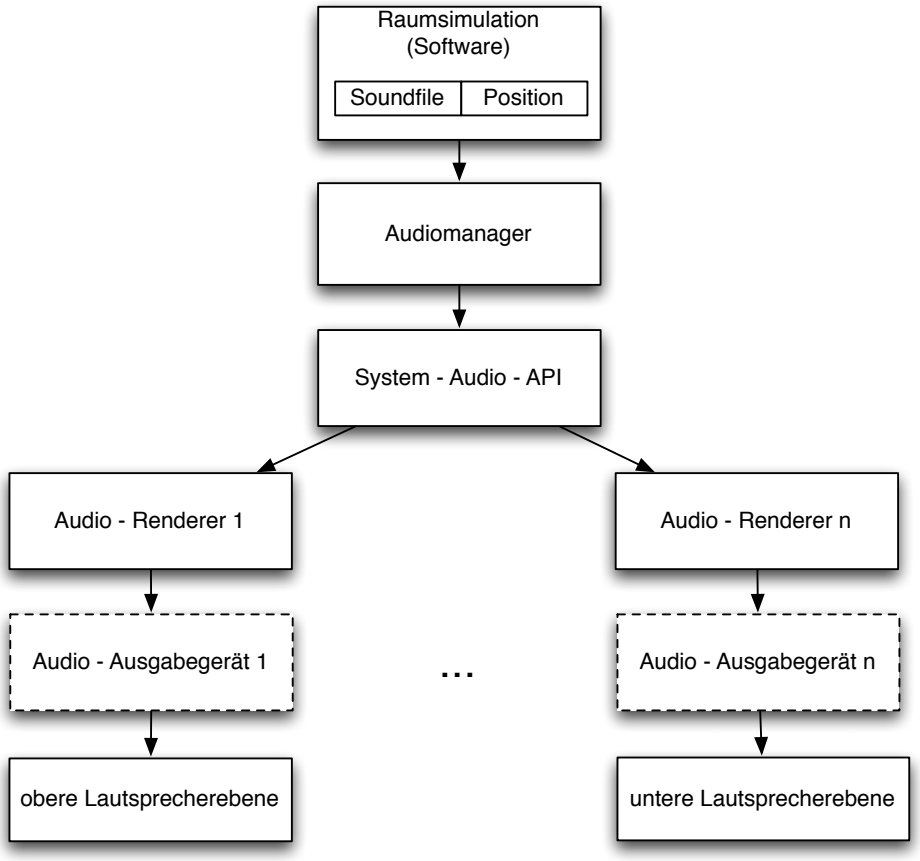


**Abbildung 4.6:** Soundsystemmodell [*eigene Darstellung*]

steuerung der Soundkarten benutzen.

**System-Audio-API** Die System-Audio-API stellt eine Verbindungsschicht zwischen dem Audiomanager und den im System verbauten Soundkarten dar. Durch diese API werden Methoden bereitgestellt die eine Kommunikation zwischen Software (Audiomanager) und Hardware (Soundkarten) ermöglichen.

**Audio-Renderer** Als Audio-Renderer werden in diesem Modell die im System verbauten Soundkarten bezeichnet. Sie geben Audiosignale entweder an Audioausgabegeräte oder Aktivlautsprecher weiter.

**Audio-Ausgabegerät** Audio-Ausgabegerät bezeichnet zum Beispiel einen AV-Receiver, der die von der Soundkarte kommenden Signale auswertet und auf die angeschlossenen Lautsprecher verteilt.

Wenn aktive Lautsprecher verwendet werden, können diese direkt an die Soundkarten angeschlossen werden und man kann sich den Zwischenschritt dieser Ebene einsparen.

**Lautsprecherebene** Jede Lautsprecherebene besteht aus mehreren Lautsprechern, die vom Audio-Ausgabegerät kommende Signale verarbeiten und in Schallwellen umwandeln. Dadurch werden die Klänge für Hörer wahrnehmbar.

Im Gegensatz zur meist monolithischen Struktur anderer betrachteter Systeme ermöglicht der modulare Aufbau des hier entwickelten Soundsystemmodells, alle Komponenten beliebig zu ersetzen.
Beispielsweise lässt sich der Audiomanager austauschen. Der Rest des Graphen kann dabei unverändert bleiben, da er nur Methoden eines Audiomanagers benutzt, deren Vorhandensein durch die Implementierung des AudioManagerInterface sichergestellt wird (siehe Abschnitt 4.4.2, Seite 49).
Ebenso lassen sich als „Audio-Renderer“ beliebige Soundkarten verwenden, sofern diese die gewünschte Anzahl an Ausgabekanälen bereitstellen.
„Ausgabegeräte“ können AV-Receiver sein, die die Lautsprecher ansteuern oder es werden Aktivboxen direkt mit dem jeweiligen „Audio-Renderer“ verbunden. Für ein einheitliches Klangbild sollten auf allen Lautsprecherebenen die gleichen Lautsprecher verbaut werden.
Zusammenfassend lässt sich ableiten, dass der Kombinierbarkeit von Komponenten mit diesem Soundsystemmodell fast keine Grenzen gesetzt sind.

## 4.4 Architektur

In diesem Abschnitt wird beschrieben, wie sich die einzelnen Modelle zusammenfügen und somit die Umsetzung eines ubiquitären 3D-Sound-Systems ermöglichen.

### 4.4.1 Integration Benutzermodell in Raummodell

Der Szenengraph des Raummodells ist die Basis der Architektur. In ihn kann das Benutzermodell als Unterknoten von Verband oder Raum eingehangen werden. Damit ist es möglich, Personengruppen bestimmten Bereichen zuzuordnen, wobei bei jedem Betreten oder Verlassen des Bereichs die Anzahl der Hörer in diesem berechnet wird. Denkbar sind dabei die folgenden Konstellationen.

**Ein Benutzer, ein Raum - statisch** In diesem einfachsten Fall besteht die Simulation lediglich aus einem Raum inklusive einem Benutzer, der den Raum nicht verlässt. Da keine Bewegung zwischen verschiedenen Räumen erfolgt, wurde diese Variante als „statisch“ klassifiziert - innerhalb des Raumes kann sich der Nutzer weiterhin bewegen.
Weil die Anzahl der Nutzer im Raum in dieser Situation genau bei eins liegt und sich somit das gesamte Geschehen auf sie bezieht, wird die Postion des Benutzers auf den Sweetspot abgebildet.
Entfernungen zwischen Schallquellen im Raum und dem Benutzer werden auf diese Weise „direkt“ aus der Position des Hörers und der Position der jeweiligen Schallquelle berechnet. Dies ist schneller als vorher einen Sweetspot zu berechnen, da die Anzahl der Nutzer im Raum sowieso ermittelt wird und somit nur ein einfacher Vergleich stattfinden muss.
Anhand der ermittelten Distanzen lässt sich je nach Umsetzung der 3D-Sound-Implementierung beispielsweise eine Distanzdämpfung der Schallquelle ableiten.
Gekoppelte Schallquellen (siehe Abschnitt 4.2, Seite 42) sind dem Hörer untergeordnet, werden in diesem Fall jedoch wie die restlichen Schallquellen im Raum behandelt.

**Mehrere Benutzer, ein Raum - statisch** Ein weiterer statischer Fall ist gegeben, wenn sich eine Gruppe von Benutzern bis zum Ende der Simulation in einem Raum aufhält, ohne dass Anwender diesen verlassen oder neue Nutzer dazukommen.
Da die Anzahl der Nutzer im Raum größer als eins ist, wird in diesem Fall für den Sweetspot ein Mittelwert aller Positionen der Benutzer im Raum gebildet. Dieser wird herangezogen, wenn Entfernungen zwischen Schallquellen im Raum und der Nutzergruppe berechnet werden sollen. Schallquellen beziehen sich dabei immer auf den Sweetspot, es sei denn es handelt sich um „gekoppelte Schallquellen“, welche im direkten Bezug zu der Position des Nutzers stehen, dem sie untergeordnet sind.

**Mehrere Benutzer, mehrere Räume - dynamisch** In diesem Fall gilt es Benutzer, die den Raum betreten, in den „Hörerverband“ beziehungsweise die Sweetspot-Berechnung mit einzubeziehen. Oder, im umgekehrten Fall, aus der Berechnung zu entfernen, wenn sie den Raum verlassen. Zudem muss anschließend geprüft werden, ob sie einen anderen Raum betreten haben. Ist dies der Fall, so müssen die Benutzer in den „Hörerverband“ des neuen Raumes aufgenommen werden.

**Ein Benutzer, mehrere Räume - dynamisch** Der Fall, dass sich nur ein Benutzer durch die Räume bewegt, ist „ein Benutzer, ein Raum - statisch“ sehr ähnlich. Der Unterschied besteht lediglich darin, dass er beim Verlassen des einen Raumes aus diesem abgemeldet und beim Betreten des nächsten Raumes in diesem registriert wird. Alle „gekoppelten Schallquellen“ des Nutzers können ihm von Raum zu Raum folgen, da sie Unterknoten dieses Benutzers sind.
In jedem Raum werden zusätzlich zu den mitgeführten gekoppelten Schallquellen alle Entfernungen zwischen dem Nutzer und den bereits im Raum vorhandenen Schallquellen berechnet.

Das Tracking des Nutzers ist nicht Teil dieser Arbeit. Deshalb werden hier auch keine Lösungen beschrieben, wie festgestellt werden kann, ob ein Benutzer einen Raum verlässt oder betritt. Denkbar wäre in diesem Hinblick sicherlich eine Ortung mittels Microsofts Kinect oder mit Hilfe von GPS, sofern dies genau genug ist. Für ein Tracking ist ist es in jedem Fall wichtig, dass genau bestimmt werden kann, in welchem Raum sich ein Benutzer befindet.
Darauf wird an dieser Stelle jedoch nicht näher eingegangen.

### 4.4.2 Audiomanager und Soundsystemmodell

Für jeden Raum in der Realität beziehungsweise Raumknoten im Modell dieser Arbeit muss ein Audiomanager existieren. Dieser stellt sicher dass die räumliche Aufteilung der Audiosignale auf mehrere Soundkarten, respektive Soundsysteme ordnungsgemäß erfolgt. Dabei ist es denkbar, unterschiedliche AudioManager für verschiedene Installationen der Lautsprechersysteme, zu entwickeln. Da die Schnittstelle „AudioManagerInterface“ von jedem Audiomanager implementiert werden muss, wird sichergestellt, dass sich beliebige Manager in das Raummodell einbinden lassen (siehe Abbildung 4.7, Seite 50).
Da jeder Audiomanager einem Raum zugeordnet ist hat er nur Zugriff auf die Daten dieses Raumes. Ihm werden beispielsweise Hörer- und Schallquellenpositionen von der Anwendung („Raumsimulation“) zur Verfügung gestellt.
Im Soundsystemmodell dieser Arbeit befindet sich der AudioManager zwischen der Raumsimulation und der System-Audio-API. Er wird von der Anwendung mit Daten versorgt, die er dann mittels der System-Audio-API an die Hardware weitergibt.

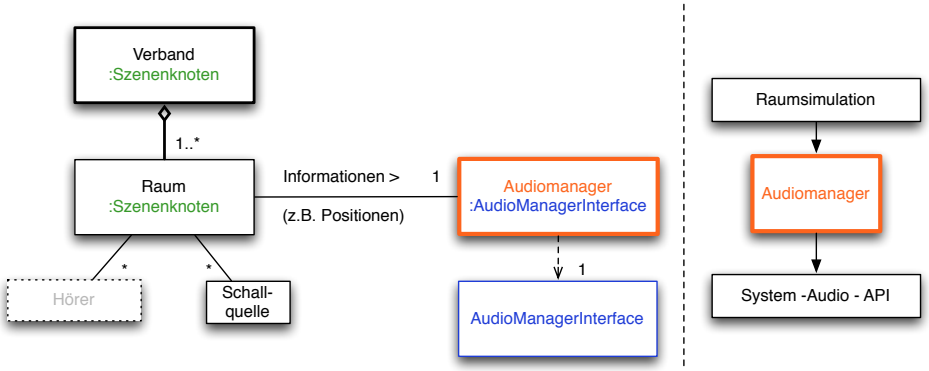


**Abbildung 4.7:** Ausschnitt Audiomanager in Raummodell (links) und in Soundsystemmodell (rechts) [*eigene Darstellung*]

### 4.4.3 Lautsprecherebenen

Es ist möglich einen Raum mit beliebig vielen Lautsprecherebenen auszustatten, wobei diese in gleichmäßigen Abständen vertikal zwischen dem Boden und der Decke des Raumes verteilt angebracht sein sollten (siehe Abbildung 4.8, Seite 51). Der Audiomanager ist dafür verantwortlich, die Laustärke der Schallquellen abhängig von ihrer vertikalen und horizontalen Position im Raum zu dämpfen und ihre Position akustisch wiederzuspiegeln. Er nimmt dabei die Aufteilung auf die Lautsprecherebenen vor und sorgt dafür, dass alle Klänge auf den Ebenen zum Abspielen vorbereitet werden und wenn es erforderlich ist, auch abgespielt werden.

## 4.5 Zusammenfassung

In diesem Kapitel wurde ein modular aufgebautes Raummodell, basierend auf einem Szenengraphen erläutert. Durch Detailstufen lässt sich der Grad an Realismus abhängig von der Anzahl der Schallquellen und der zur Verfügung stehenden Rechenleistung an die jeweilie Situation anpassen.
Schallquellen lassen sich einem „Verbandsknoten“ unterordnen und so gruppieren. Zudem können Bewegungsanimationen definiert werden, um „Klänge durch den Raum fliegen“ zu lassen.
Im für diese Arbeit aufgestellten Benutzermodell wurde erklärt, wie sich Benutzer in einen Raum einfügen und die Berechnung eines (gewichteten) Sweetspots für mehrere Nutzer aussehen kann.

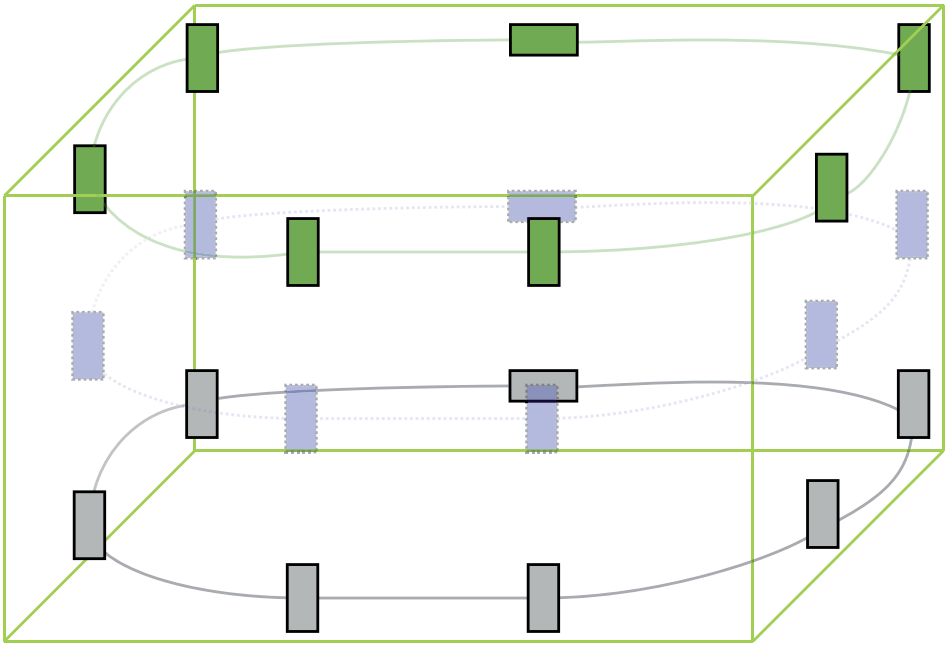

**Abbildung 4.8:** Skizze Lautsprecherebenen vertikal verteilt [*eigene Darstellung*]

Das vorgestellte Soundsystemmodell zeigt wie die verschiedenen Bestandteile des Soundsystems zusammenspielen. Es verdeutlicht außerdem, dass dieses Modell im Vergleich zu bisherigen „kostengünstigen“ 3D-Sound-Systemen nicht auf bestimmte Komponenten beschränkt ist, sondern sich nahezu in jedem Punkt flexibel anpassen lässt.
Desweiteren wurde erläutert, wie sich Nutzer innerhalb und zwischen Räumen bewegen und warum der Audiomanager eine wichtige Komponente des Systems darstellt.
Aufbauend auf diesen Erkenntnissen wird im nächsten Kapitel die Implementierung eines ersten Prototypen „Ubiqu3DA“ für einen Raum vorgestellt.

# 5 Prototypische Implementierung

In diesem Kapitel wird auf die prototypische Umsetzung eines 3D-Sound-Systems, basierend auf den zuvor entwickelten Modellen und der zugrundeliegenden Architektur eingegangen.
Ziel dieser Arbeit ist es, einen funktionsfähigen 3D-Sound-System-Prototypen unter Verwendung zweier 7.1 Soundsysteme für einen Raum zu implementieren.
In diesem Kapitel wird die Implementierung anhand des Soundsystemmodells dargestellt. Beginnend mit der Hardware (den Lautsprecherebenen), über die Schnittstelle zwischen Hard- und Software (der System-Audio-API) und abschließend mit dem Prototypen der Raumsimulation wird die Umsetzung des 3D-Sound-Systems erläutert.
Anschließend erfolgt eine Erklärung, wie ein Projekt mit dem „XACT 3.0 - Audio Creation Tool“ erstellt werden kann, das sich nach einer Kompilierung mit dem entwickelten Prototypen laden und verarbeiten lässt.

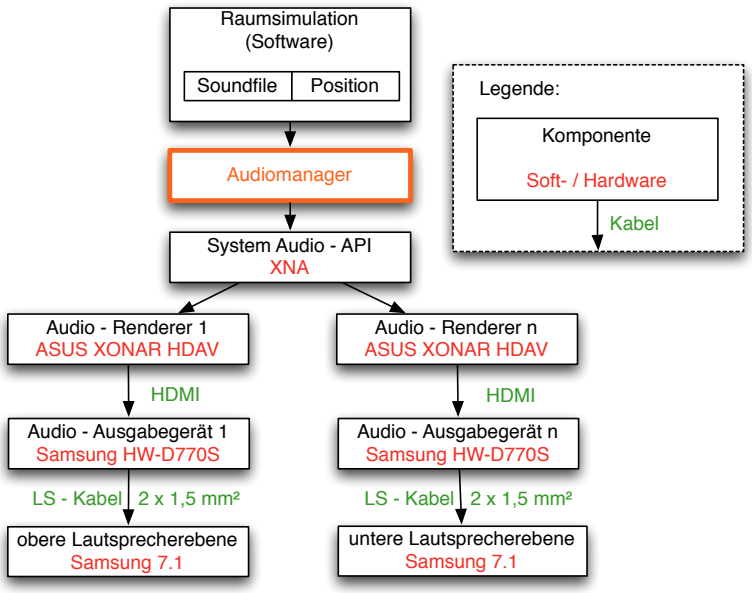


**Abbildung 5.1:** Implementierung des Soundsystemmodells [*eigene Darstellung*]

## 5.1 Lautsprecherebenen

Um einen möglichst realistischen 3D-Sound zu erzielen und nicht nur horizontale, sondern auch vertikale Positionsänderungen von Schallquellen wahrnehmbar zu reproduzieren, wurden in einem Raum (Breite: 5,89 m, Höhe: 3,10 m, Tiefe: 6,68 m) zwei Lautsprecherebenen installiert. Eine Ebene ist knapp unter der Decke in 2,75 m Höhe angebracht. Die zweite Ebene befindet sich am Boden des Raumes. Die Entscheidung für lediglich zwei Ebenen ist damit zu begründen, dass sich schon mit den beiden Lautschprecherebenen eine Schallquelle in verschiedenen simulierten Höhen unterscheidbar wiedergeben und sich diese Installation kostengünstig realisieren lässt.
Insgesamt kommen vierzehn Lautsprecher und zwei Tieftöner eines Samsung HW-D770S Heimkinosystems zum Einsatz. Die Lautsprecher sind zu gleicher Anzahl auf beide Ebenen verteilt. Dabei ist die Front des Raumes mit drei Boxen, die Seitenwände mit jeweils einer Box und die Rückwand des Raumes mit zwei Lautsprecherboxen auf jeder Ebene ausgestattet (siehe Abbildung 5.2, Seite 54). Die Tieftöner wurden am Boden vor der Rückwand positioniert. Da der Mensch ihre ausgestrahlten tiefen Frequenzen nicht orten kann (siehe Abschnitt 2.3, Seite 12), war die Aufstellung im Raum frei wählbar.
Alle Lautsprecher im Raum werden von zwei Samsung HW-D770S AV-Receivern angesteuert, welche die Audio-Ausgabegeräte des Soundsystemmodells darstellen. Darauf wird im nächsten Abschnitt eingegangen.

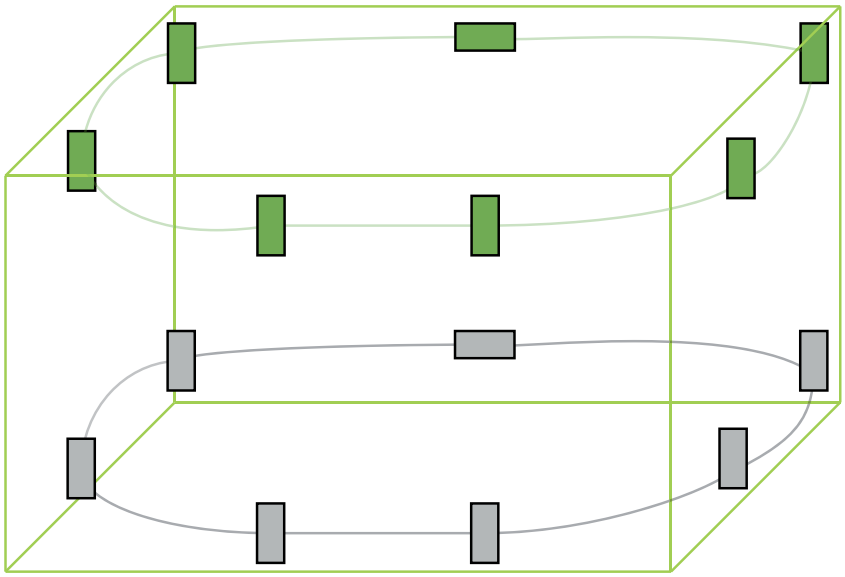

**Abbildung 5.2:** Skizze Lautsprecherebenen an Raumdecke und -boden [*eigene Darstellung*]

## 5.2 Audio-Ausgabegeräte

Ein Audio-Ausgabegerät dekodiert die von der Soundkarte kommenden Audiostreams, verstärkt die Audiosignale und verteilt diese an die angeschlossenen Lautsprecher.
In der Installation des in dieser Arbeit umgesetzten 3D-Sound-Systems kommen als Audio-Ausgabegeräte zwei AV-Receiver vom Typ Samsung HW-D770S zum Einsatz. Diese stammen aus dem Heimkinobereich und verfügen über mehrere HDMI-Eingänge. Sie unterstützen jeweils den Anschluss von sieben Lautsprechern und zwei Tieftönern. Pro AV-Receiver wird hier jedoch nur ein Tieftöner angeschlossen, da sich gezeigt hat, dass die Konfiguration aus ingesamt zwei Tieftönern (beider Ebenen) für den 42 $m^2$ großen Raum des Aufbaus mehr als ausreichend ist.

## 5.3 Audio-Renderer

Die Soundkarten des 3D-Sound-Systems sind die Audio-Renderer nach dem in dieser Arbeit aufgestellten Soundsystemmodell. Sie verarbeiten die vom Prototypen (Raumsimulation) kommenden Audiosignale, führen die 8-Kanal-Abmischung durch und erstellen einen PCM(„Pulse Code modulierten“)- Stream zur Weitergabe an die Audio-Ausgabegeräte.
Im Gegensatz zu anderen betrachteten 3D-Sound-Systemen kommt in dieser Arbeit eine digitale Signalübertragung mittels HDMI zur Anwendung.
Dies hat den Vorteil, dass man pro Soundkarte nur ein Kabel zum zugehöringen AV-Receiver verlegen muss. Durch den digitalen Übertragungsweg und die geringere Anzahl an Kabeln ist das System wesentlich unanfälliger gegenüber Störungen als bei Nutzung analoger Übertragungswege. Zudem reicht die Bandbreite von HDMI aus, um mindestens acht Audiokanäle unkomprimiert und verlustfrei zu übertragen.
Hier werden zwei ASUS XONAR HDAV verwendet, welche jeweils ein 8-Kanal-PCM-Signal per HDMI-Schnittstelle zur Verfügung stellen können, das dann digital an den entsprechenden AV-Receiver übertragen wird.
Zu beachten gilt, den Soundkarten auch ein Videosignal einzuspeißen, da sie sonst stumm bleiben. Man kann dazu zum Beispiel das Monitorbild auf einen HDMI-Splitter klonen, welcher die Videosignale wiederum auf alle Soundkarten verteilt.
Alternativ lassen sich anstatt der Soundkarten auch aktuelle Grafikkarten verwenden, die über einen dedizierten Audioprozessor verfügen und auch ein 8-Kanal-

PCM-Signal auf dem HDMI-Ausgang ausgeben können. Getestet wurden dafür in dieser Arbeit zwei AMD Radeon HD 6450 PCIe-Grafikkarten (unter Windows 7), mit denen die Audioausgabe über HDMI problemlos funktioniert.

## 5.4 System-Audio-API: XNA

Die System-Audio-API verbindet den Audiomanager mit den Audio-Renderern. Sie stellt eine Klassenbibliothek zur Verfügung, die eine Kommunikation mit der Soundhardware ermöglicht.
Im entwickelten Prototypen wird Microsofts XNA als Audio-API verwendet. Diese stellt ein Entwicklungswerkzeug dar, mit dem man sowohl 2D- als auch 3D-Spiele umsetzen kann. Die Verwendung von C# als Programmiersprache erleichtert den Einstieg und vereinfacht dem Programmierer die Entwicklung, da er sich nicht um das Speichermanagement kümmern muss.
Neben Klassenbibliotheken zur Grafik- und Eingabeverarbeitung werden auch Klassen zur Audiomanipulation bereitgestellt.
XNA bietet mit „SoundEffect“ und „XACT“ zwei verschiedene Ansätze, um Sound in ein Spiel beziehungsweise Programm zu integrieren.
Diese sollen im Folgenden kurz erklärt werden.

### 5.4.1 SoundEffect-Klasse

Die Klasse „SoundEffect“ ist die einfachste Möglichkeit, Sound mit XNA in eine Anwendung einzubauen. Es sind lediglich die folgenden drei Schritte erforderlich bis eine Datei abspielbar ist.

(1) Eine Instanz von „SoundEffect“ erstellen.
(2) Den relativen Pfad (beginnend im Anwendungsverzeichnis) zur Datei ohne Endung an die XNA-Hilfsklasse „ContentManager“ zum Laden dieser Datei übergeben und den Rückgabewert an die zuvor erstellte Instanz von SoundEffect weiterleiten.
(3) Zum Abspielen muss nur noch die Play()-Methode der Instanz von SoundEffect aufgerufen werden.

Um beispielsweise die Datei „music.wav“ aus dem Programmhauptverzeichnis abzuspielen, würde der Code wie in Listing 5.1, Seite 57 aussehen.

```
1 SoundEffect testSound;
2 testSound = Content.Load("music");
3 testSound.Play();
```

**Listing 5.1:** „music.wav“ mittels SoundEffect-Klasse abspielen

Der Ansatz mittels der SoundEffect-Klasse Audiodateien in ein Programm zu integrieren , ist sehr einfach gehalten, wie das obige Beispiel verdeutlicht. Dies stellt jedoch ein Problem dar, da sich dem Programmierer wenig Einflussmöglichkeiten bieten.
SoundEffect kann nur auf die im System als „Standard“ festgelegte Soundkarte zugreifen und bietet keine Optionen dies zu ändern.
Wenn Sound wie in dieser Arbeit auf mehreren Lautsprecherebenen ausgegeben werden soll, ist die Verwendung von SoundEffect nicht möglich, da eine Ansteuerung von mehr als einer Soundkarte zur gleichen Zeit damit nicht realisierbar ist.
Für den entwickelten Prototypen wurde daher der zweite Ansatz - Einbindung mittels XACT - gewählt, der als nächstes vorgestellt wird.

### 5.4.2 XACT

Eine weitere Möglichkeit, Audio in eine Anwendung mittels XNA einzubinden, stellt „XACT“ dar. Es handelt sich dabei um eine Trennung von Sounddesign und Programmierung.
Mit dem XACT Audio Creation Tool werden die Audioprojekte erstellt und vorkompiliert, die im Prototypen dieser Arbeit geladen und verarbeitet werden. Eine detaillierte Erklärung, wie man in XACT ein Projekt für die Raumsimulation erstellt folgt in Abschnitt 5.7 auf Seite 67.

Das Einbinden eines XACT-Projekts in die Raumsimulation ist etwas umständlicher als die Verwendung der SoundEffect-Klasse, um einen Sound abzuspielen. Es lassen sich jedoch mehrere Audio-Renderer initialisieren und ansteuern sowie Parameter wie die Vorbereitungszeit eines Sounds („lookAheadTime“) anpassen.
Wenn man davon ausgeht, dass sich alle benötigten Projektdateien in einem Ordner mit der Bezeichnung „AudioSim1“ befinden und die ID des Audio-Renderers bereits ermittelt wurde, muss bei der Programmierung wie in Listing 5.2 , Seite 58 vorgegangen werden, um den Sound „music“ abzuspielen.
In dieser Arbeit ist der Audiomanager dafür zuständig die komplette Initialisierung aller Audio-Renderer zu übernehmen und das Abspielen von Sounds einzulei-

```
1  // Pfadangaben
2  String projectSettingsPath = "AudioSim1\\settings.xgs";
3  String soundBankPath = "AudioSim1\\sB.xsb";
4  String waveBankPath = "AudioSim1\\wB.xwb";
5  // Audio-Renderer initialisieren
6  TimeSpan lookAheadTime = new System.TimeSpan(0,0,0,0,250);
7  AudioEngine audioRenderer1 = new
       AudioEngine(projectSettingsPath, lookAheadTime, id);
8  // Sound- und WaveBank initialisieren
9  SoundBank sb1 = new SoundBank(audioRenderer1, soundBankPath);
10 WaveBank wb1 = new WaveBank(audioRenderer1, waveBankPath);
11 // SoundCue aus SoundBank holen und abspielen
12 Cue musicCue = sb1.GetCue("music");
13 musicCue.Play();
```

**Listing 5.2:** SoundCue „music“ aus XACT-Projekt abspielen

ten. Darauf wird in Abschnitt 5.5, Seite 60 näher eingegangen.

### 5.4.3 Limitierungen durch XNA

Bei allen Berechnungen von 3D-Effekten, wie beispielsweise der Distanzdämpfung, geht XNA davon aus, dass sich ein Hörer in der Mitte aller installierten Lautsprecher befindet. Dies bedeutet für die Implementierung des Prototypen dieser Arbeit, dass es nur einen Sweetspot gibt und sich dieser in der Mitte des Raumes befindet.
Wird versucht die im Benutzermodell (siehe Abschnitt 4.2, Seite 42) entwickelte Sweetspotberechnung in XNA umzusetzen, entstehen Nebeneffekte, die den 3D-Eindruck der Simulation negativ beeinflussen. Ein Nebeneffekt ist beispielsweise, dass Klänge nicht mehr im Raum ortbar sind, sobald Schallquellen an einen Hörer gekoppelt werden und sich dieser bewegt. Die Klänge werden dann ohne Veränderung ihrer Position abgespielt und es ist keine Bewegung im Raum wahrnehmbar. Grund dafür ist, dass der Hörer sich entgegen seiner realen Position, API-intern in der Mitte des Raumes befindet und der Abstand zu den gekoppelten Schallquellen trotz der Bewegung immer gleich bleibt. Die Lautstärke einer Schallquelle lässt sich über ihre Distanz zum Hörer beeinflussen. Es gibt jedoch keine Möglichkeit manuell zu bestimmen, aus welchen Boxen ein Klang abgespielt werden soll, um die Schallquelle korrekt im Raum zu positionieren.
Um dennoch einen 3D-Effekt zu erzielen, wenn Schallquellen dem Hörer untergeordnet sind, wird hier ein anderer Ansatz verfolgt. Die 3D-Berechnungen von XNA werden auf Benutzer bezogen, die mittig im Raum auf Höhe der Lautsprecherebene platziert sind. Wenn Hörer über gekoppelte Schallquellen verfügen, so

wird jede dieser Quellen um den Wert der Änderung der Nutzerposition im Raum verschoben (siehe Abbildung 5.3, Seite 59). Die anschließende 3D-Berechnung mittels XNA bezieht sich wiederum auf die Raummitte. So bleibt der Abstand zwischen Schallquelle und Hörer immer konstant, der Abstand zwischen der Quelle und der Raummitte verändert sich jedoch. Dadurch lässt sich die Position der Quelle orten, auch wenn sie an einen Hörer gekoppelt ist. Somit bleibt der wahrgenommene Klang der gekoppelten Schallquellen eines Hörers für ihn immer gleich, auch wenn er sich bewegt. Die Schallquellen ändern trotzdem akustisch ortbar ihre Position, da sie von den passenden Lautsprechern wiedergegeben werden.

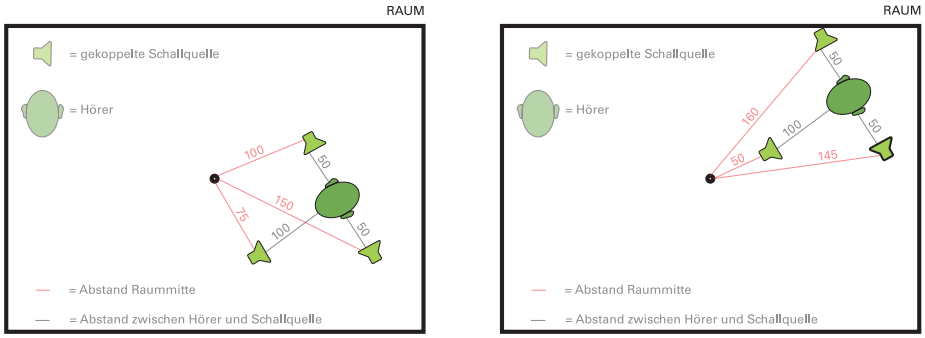


**Abbildung 5.3:** Links: Hörer inkl. gekoppelten Schallquellen **vor** Bewegung
Rechts: Hörer inkl. gekoppelten Schallquellen **nach** Bewegung [*eigene Darstellung*]

Da sich die Distanz zwischen Schallquelle und Raummitte ändert, würde standardmäßig in XNA auch eine Distanzdämpfung stattfinden und der Klang abhängig vom Abstand zur Mitte unterschiedlich laut wiedergegeben werden. Die Distanz zwischen dem Hörer und der gekoppelten Schallquelle bleibt aber immer gleich während er sich bewegt. Aus diesem Grund soll bei horizontalen Bewegungen innerhalb des Raumes keine Dämpfung stattfinden.
In der Höhe soll jedoch auf den einzelnen Ebenenen eine Dämpfung erfolgen, um die vertikale Position einer Schallquelle akustisch im Raum abzubilden.
Dieser Konflikt wird durch Abbildung des Raumes auf einen Würfel, einer individuellen Dämpfungskurve („RPC-Preset“ in XACT Audio Creation Tool) und einem Normalisierungsfaktor gelöst.
Es ist die Aufgabe des Audiomanagers, diese Lösung mit Hilfe der System-Audio-API auf alle verwendeten Audio-Renderer anzuwenden. Es wurde dafür ein Lautstärkemodell in XNA aufgestellt, auf welches im nächsten Abschnitt eingegangen wird.

## 5.5 Audiomanager

Der Audiomanager nutzt die von XNA zur Verfügung gestellten Klassen, um für die Raumsimulation Operationen auf den Audio-Renderern transparent durchzuführen.
Er verwaltet alle initialisierten Renderer („AudioLayer“) und eine Liste mit allen geladenen Klängen. Soll das Abspielen eines Klangs gestartet werden, führt der Audiomanager die Abspiel-Methode auf allen AudioLayern aus. Gleiches gilt für das Stoppen, Wiederaufnehmen und Verwerfen eines Klangs.
Soll eine Schallquelle im Raum positioniert werden, so normalisiert der Audiomanager ihre Position und leitet eine 3D-Berechnung auf allen AudioLayern ein. Dies wird nachfolgend (siehe Abschnitt 5.5.1, Seite 60) erklärt.

### 5.5.1 Lautstärkemodell in XNA

Weil XNA davon ausgeht, dass sich der (API-interne) Hörer in der Mitte der Lautsprecher befindet, wird bei Verwendung der Standarddämpfung von XNA der Klang zum Rande des Raumes leiser. Im Extremfall könnte es beispielsweise sein, dass sich eine Schallquelle und ein (realer) Hörer am Rande des Raumes an der gleichen Position befinden, aber kein Klang zu hören ist, da er komplett gedämpft wird.
Aus dem Grund wurde in dieser Arbeit ein Lautstärkemodell entwickelt, welches dafür sorgt, dass die Schallquellen hauptsächlich bezüglich ihrer vertikalen Position im Raum auf den Lautsprecherebenen (AudioLayers) unterschiedlich gedämpft werden.
Als erstes wird dazu der Raum auf einen Würfel mit einer Kantenlänge von 2000 Einheiten abgebildet, um Breite, Höhe und Tiefe auf einen einheitlichen Wert zu normalisieren und in jede Richtung die gleiche Dämpfungskurve anwenden zu können. Dies ist notwendig, da XNA die Dämpfung bezüglich der Distanz berechnet, jedoch in der Horizontale möglichst keine Dämpfung erfolgen soll, um zum Rand des Raumes das Leiserwerden einer Schallquelle zu vermeiden.
Es wurde eine eigene Dämpfungskurve ermittelt und im XACT Audio Creation Tool als RPC-Preset angelegt (siehe Abbildung 5.4, Seite 61).

Die Dämpfungskurve definiert folgende drei Abschnitte:

(1) In einer Umkugel vom Radius zwischen 0 und 1414,21 Einheiten, ausgehend vom Mittelpunkt der normalisierten Lautsprecherebene erfolgt keine Dämpfung, um eine Verringerung der Lautstärke am Rand der Ebene zu vermeiden.

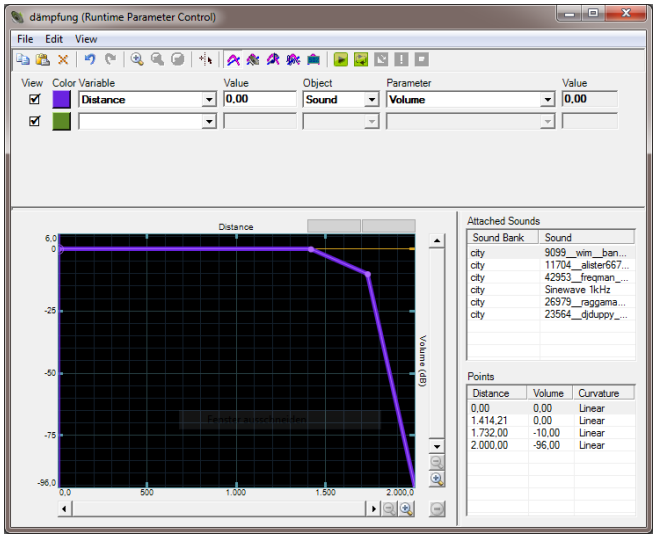


**Abbildung 5.4:** Dämpfungskurve - RPC Preset in XACT Audio Creation Tool

Dieser Radius ergibt sich nach dem Satz des Pythagoras aus der Distanz *dist* zwischen dem Mittelpunkt und einem Eckpunkt der normalisierten Ebene (siehe Abbildung 5.5, Seite 61):

$$dist = \sqrt{1000^2 + 1000^2} \approx 1414,21$$

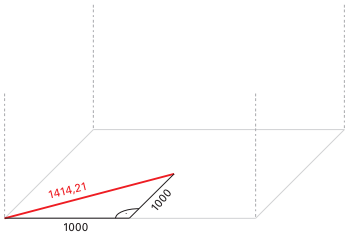


**Abbildung 5.5:** Radius vom Mittelpunkt zum Eckpunkt = 1414,21 [*eigene Darstellung*]

(2) In einem Radius zwischen 1414,21 und 1732 um den Mittelpunkt der normalisierten Lautsprecherebene erfolgt eine lineare Dämpfung von 0 auf -10 db, um auf halber Höhe des Raumes die Lautstärke jeder Ebene zu halbieren. Dies ist notwendig, um einen Anstieg der Gesamtlautstärke zu vermeiden, da im mittleren Bereich des Raumes beide Lautsprecherebenen gleichzeitig Ton ausgeben. Laut Begault entsprechen -10 db einer wahrgenommenen Halbierung der Lautstärke [Beg00]. Dabei stehen 0 db für die Normallautstärke des Sounds und -96 db für Stille.

„1732 Einheiten“ entspricht nach Pythagoras der Distanz *dist* zwischen dem Mittelpunkt und einem Eckpunkt des Raumes (siehe Abbildung 5.6, Seite 62):

$$dist = \sqrt{1414,21^2 + 1000^2} \approx 1732,0$$

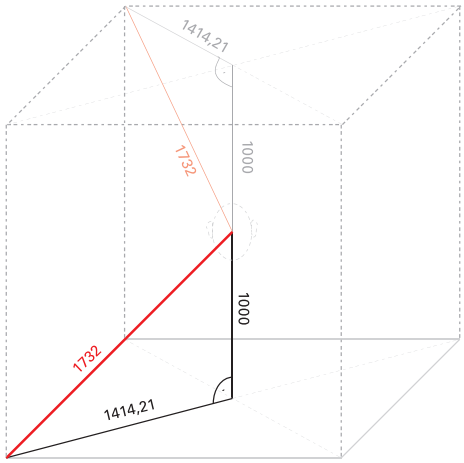


**Abbildung 5.6:** Radius Raummitte zu -ecke beträgt 1732 Einheiten [*eigene Darstellung*]

(3) In einem Radius zwischen 1732 und 2000 erfolgt eine lineare Dämpfung auf -96 db, sodass die Schallquelle verstummt. Dieses Abklingen wird benötigt, um eine Dämpfung in der Vertikalen zu erreichen, damit Klänge auf den Lautsprecherebenen unterschiedlich laut wiedergeben werden und sich somit einen Höhenunterschied herausstellt.

Um die reale Position einer Schallquelle im physischen Raum auf einen Bereich innerhalb des Würfels (Kantenlänge 2000) abzubilden, muss sie mit einem Normalisierungsfaktor multipliziert werden.
Der Faktor für die X-, Y- und Z-Achse berechnet sich wie folgt:

$$Normalisierungsfaktor_x = 2000/Raumbreite$$

$$Normalisierungsfaktor_y = 2000/Raumhoehe$$

$$Normalisierungsfaktor_z = 2000/Raumtiefe$$

In jeder vom Audiomanager verwalteten Lautsprecherebene (AudioLayer) ist ein ein „LayerListener“ hinterlegt. Dieser hat als Positionsvektor („LayerCenterPos“) genau die normalisierte Mitte der Ebene auf der normalisierten Höhe der Ebene

gespeichert (siehe Abbildung 5.7, Seite 63).

$$Vektor_{LayerCenterPos} = (1000, (Hoehe_{Lautsprecherebene} * Normalisierungsfaktor_y), 1000)$$

Der Audiomanager berechnet für jede Lautsprecherebene mittels der XNA „Apply3D()"- Methode den Euklidischen Abstand zwischen der Position des „Layer-Listeners" der Ebene und der Schallquellenposition. Anschließend löst die „Apply3D"- Methode eine Dämpfung, basierend auf dem ermittelten Abstand und der im RPC-Preset hinterlegten Kurve (siehe Abbildung 5.4, Seite 61) aus.

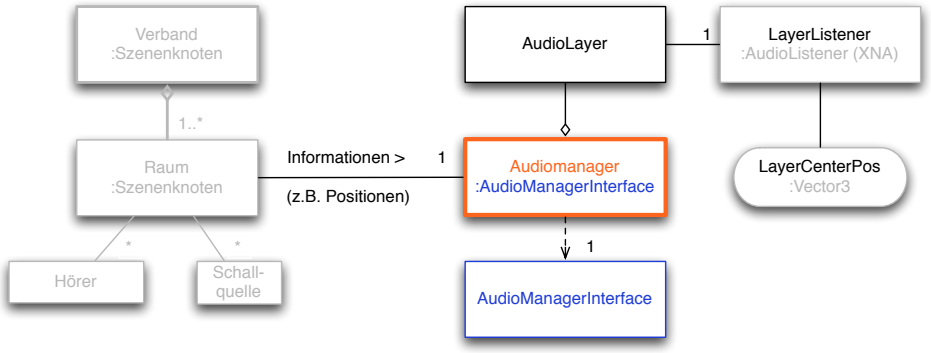


**Abbildung 5.7:** AudioManager inkl. Lautsprecherebenen (AudioLayer) [*eigene Darstellung*]

Angenommen der LayerListener $LL0$ der Ebene 0 hat die Position $(1000, 0, 1000)$, der LayerListener $LL1$ der Ebene 1 die Position $(1000, 1774, 1000)$ und eine Schallquelle $SQ0$ die Position (400, 300, 200). Dann wird der Euklidische Abstand $d$ wie folgt berechnet:

$$d_{Ebene0}(LL0, SQ0) = \sqrt{(1000-400)^2 + (0-300)^2 + (1000-200)^2} \approx 1044,03$$

$$d_{Ebene1}(LL1, SQ0) = \sqrt{(1000-400)^2 + (1774-300)^2 + (1000-200)^2} \approx 1781,20$$

Nach dem Abgleich von $d_{Ebene0}(LL0, SQ0) \approx 1044,03$ mit der Dämpfungskurve erfolgt auf Lautsprecherebene 0 keine Dämpfung, da $d_{Ebene0}(LL0, SQ0) < 1414,21$ gilt.
Nach dem Abgleich von $d_{Ebene1}(LL1, SQ0) \approx 1781,20$ erfolgt auf Lautsprecherebene 1 eine Dämpfung um -25,77 db, da folgende Herleitung gilt:

$$1732 < d_{Ebene1} \approx 1781,20 < 2000$$

$$(2000 - 1732)/((-96db) - (-10db)) = 268/-86db \approx -3,12\frac{1}{db}$$

$$1781,20 - 1732 = 49,2$$
$$49,2/ - 3,12\frac{1}{db} = -15,77db$$
$$-15,77db - 10db = -25,77db$$

Die Schallquelle wird im obigen Beispiel somit von Lautsprecherebene 0 mit ihrer Normallautstärke abgespielt und von Lautsprecherebene 1 um knapp 26 db gedämpft. Durch diesen Lautstärkeunterschied lässt sich die Schallquelle im unteren Bereich des Raumes orten.

## 5.6 Raumsimulation - Prototyp Ubiqu3DA

Als Raumsimulation wurde in dieser Arbeit ein Prototyp „Ubiqu3DA“ implementiert, der auf einer Vereinfachung des zuvor beschriebenen Raummodells basiert (siehe Abschnitt 4.1, Seite 35). Es lassen sich entgegen dem Raummodell beispielsweise keine „Öffnungen“, „Wände“, „Gegenstanstände“ oder „Detailstufen“ festlegen. Der Prototyp implementiert die Grundfunktionalität des Raummodells und lässt sich in späteren Versionen durch weitere Features ergänzen.
Er verwendet den Audiomanager, um die Soundkarten des Systems anzusprechen und verarbeitet Projekte, die mit dem XACT Audio Creation Tool erstellt wurden. Im Folgenden werden die Funktionen des Prototyps kurz vorgestellt.

**Ausgangssituation** Nach dem Start von Ubiqu3DA hat man die Möglichkeit, eine Raumkonfiguration zu laden beziehungsweise neu zu erstellen. Zudem lässt sich ein Ordner öffnen, in dem sich ein kompiliertes XACT-Projekt befindet. Es müssen folgende Dateien in dem Projektordner hinterlegt sein:

- Eine CueList (*.scue), in welcher alle SoundBanks inklusive einer Zuordnung ihrer Sounds gespeichert sind
- Eine WaveBank (*.xwb), die alle vorkompilierten Audiodateien enthält
- Beliebig viele SoundBanks (*.xsb), die Sounds thematisch gruppieren
- Ein XACT-Settingsfile (*.xgs), welches Voreinstellungen des Projekts beinhaltet

Nachdem eine Raumkonfiguration und ein Projektordner erfolgreich geladen wurden, erfolgt automatisch die Initialisierung des Systems.
Es wird dabei ein Raumknoten („RoomSceneNode“) mit den Maßen und der ID

aus der geladenen Raumkonfiguration erstellt und dem Szenengraphen untergeordnet. Desweiteren wird ein Hörerverband („ListenerManager“) zum Raum hinzugefügt. Dieser verwaltet alle Hörer, die den Raum betreten.
Außerdem erfolgt für diesen Raum die Initialisierung des Audiomanagers inklusive aller „AudioLayer“ auf Grundlage der in der Raumkonfiguration festgelegten Audio-Renderer und der Dateien aus dem Projektordner. Zudem werden die zuvor erklärten Normalisierungsfaktoren aus den Abmessungen des Raumes berechnet und dem Audiomanager übergeben.
Bei der Erstellung jedes AudioLayers durch den Audiomanager wird die Verbindung zum voreingestellten Audio-Renderer aufgebaut, indem eine XNA-„AudioEngine“ initialisiert wird. Danach erfolgt die Initialisierung der WaveBank sowie der SoundBanks für die AudioEngine des Layers und es wird ein „LayerListener“ angelegt (siehe Abschnitt 5.5, Seite 60).
Nachdem alle Vorbereitungen abgeschlossen sind, wechselt das Programm in die Hauptansicht (siehe Abbildung 5.8, Seite 65).

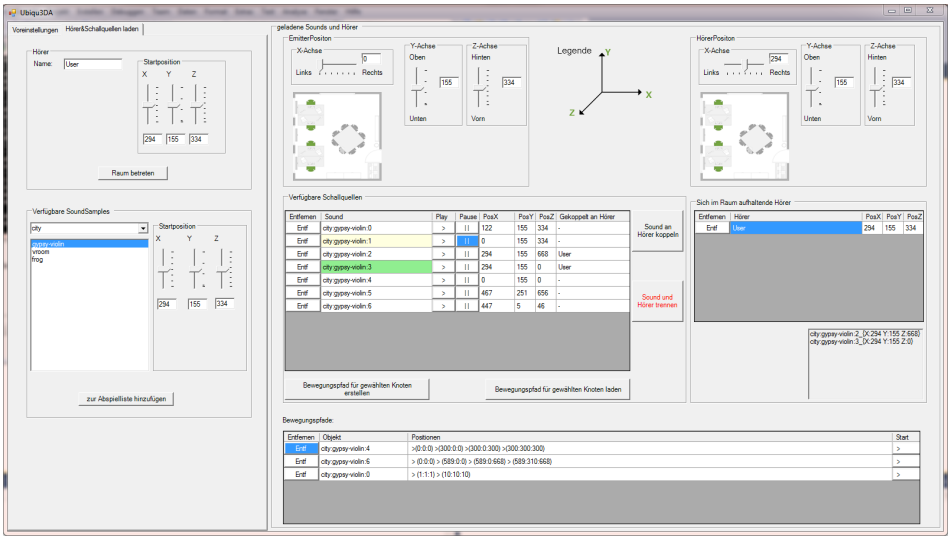


**Abbildung 5.8:** Hauptansicht des Prototypen „Ubiqu3DA“

**Initialisierung abgeschlossen, Hauptansicht geladen** In der Hauptansicht können Hörer und Schallquellen innerhalb des Raumes beeinflusst werden.
Wenn dem Raum ein neuer Hörer hinzugefügt wird, legt das Programm intern einen Hörerknoten („ListenerSceneNode“) an, passt die Position nach den Vorgaben der Trackbars beziehungsweise Eingabefelder an und fügt den Szenenknoten dem Hörerverband des Raumes hinzu.
Nachdem ein Sound aus einer der geladenen SoundBanks ausgewählt wurde, kann dieser ebenfalls dem Raum hinzugefügt werden. Dabei wird programmintern

eine Schallquelle („SoundEmitterSceneNode“) angelegt und ihr sowohl die voreingestellte Position, als auch der gewählte Sound zugewiesen.

Wird in der linken Tabellenansicht („Verfügbare Schallquellen“) ein Sound ausgewählt, so kann dieser durch Klick auf den „ > “ Button abgespielt und durch ein „ || “-Klick seine Wiedergabe pausiert werden. Bei erneutem „ > “-Klick wird die Pause beendet und der Sound spielt weiter.
Außerdem kann die Position der ausgewählten Schallquelle mit den Trackbars im Bereich „EmitterPosition“ über der linken Tabelle verändert werden. Um die horizontale Umpositionierung von Schallquellen zu vereinfachen, steht in diesem Bereich zusätzlich ein Pad mit dem Raumgrundriss zur Verfügung (siehe Abbildung 5.9, Seite 66). Nach einem Klick darauf kann durch Mausbewegung über das Pad die Schallquelle durch den Raum bewegt werden. Ein erneuter Klick stoppt die Positionsänderung. Die Hörerpositionen können auf die gleiche Weise im Bereich „HörerPosition“ manipuliert werden.
Wird eine Schallquelle nicht mehr benötigt, so kann sie mit dem „Entf“-Button aus dem Raum gelöscht werden. All ihre Einstellungen und Kopplungen verfallen dabei und der „SoundEmitterSceneNode“ wird entfernt.

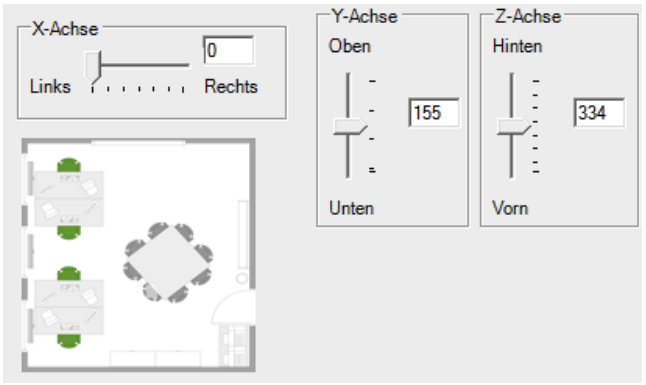


**Abbildung 5.9:** Positionierung über Trackbars oder Pad mit Grundriss des Raums

Schallquellen lassen sich an Hörer koppeln, indem man die Schallquelle und den Hörer in der jeweiligen Tabelle selektiert und den „Sound an Hörer koppeln“-Button drückt.
Programmintern wird die Schallquelle im Szenengraphen verschoben. Dazu wird der „SoundEmitterSceneNode“ dem gewählten Hörer untergeordnet und aus den Unterknoten des Raumes entfernt.

Für Schallquellen, die keinem bestimmten Hörer zugeordnet sind, lassen sich Bewegungspfade anlegen beziehungsweise zuvor erstellte Pfade laden.
Ein Bewegungspfad besteht aus KeyFrames, die einem Animationscontroller hinzugefügt werden. Jedes Keyframe besteht dabei aus einem Zeitpunkt und einem Positionsvektor. Der Controller wird der zu animierenden Schallquelle („SoundEmitterSceneNode“ untergeordnet und definiert eine neue UpdateSceneNode()-Methode, welche in jedem Updatezyklus des Szenengraphen den „SoundEmitterSceneNode“ anpasst. Im Falle des Bewegungspfads wird im Zeitraum zwischen zwei Keyframes eine lineare Interpolation über die Position der beiden Keyframes durchgeführt. Die Interpolation vermeidet, dass eine Schallquelle durch den Raum „springt“, indem sie für eine gleichmäßige Bewegung zwischen den verschiedenen Positionen sorgt.

## 5.7 Projekt mit XACT Audio Creation Tool

In diesem Abschnitt soll erklärt werden, wie man mit dem XACT Audio Creation Tool (siehe Abbildung 5.10, Seite 68) ein Projekt erstellt, welches von der Raumsimulation „Ubiqu3DA“ verarbeitet werden kann.
Dabei wird im Einzelnen auf die Besonderheiten eingegangen, damit das Projekt vom hier entwickelten Prototypen korrekt verarbeitet werden kann.

(1) Zuerst legt man ein neues Projekt an. In Abbildung 5.10 unter „1“ mit „attenuation“ bezeichnet.
Man sollte einen **neuen Ordner für das Projekt** anlegen und zumindest den **„Project Path“ sowie die Pfade von „Cue List“ und „Windows Global Setting“ auf diesen Ordner** verweisen lassen. Zudem muss die **Cue List auf „*.scue“ enden**. Diese Einstellungen sind erforderlich, damit der Prototyp den Projektordner einlesen kann.

(2) Danach erstellt man eine Wave Bank, in welche man alle Soundsamples kopiert, die man später verwenden möchte. Der **„Windows Build Path“** der Wave Bank sollte ebenfalls auf den zuvor angelegten **Projektordner** verweisen. Zudem gilt zu beachten, dass die Sounds als **WAV-Datei in 44100Hz und 16Bit** vorliegen müssen. Die für dieses Beispiel erstellte Wave Bank wurde „general“ benannt und ist in Abbildung 5.10 mit der Ziffer 2 gekennzeichnet.

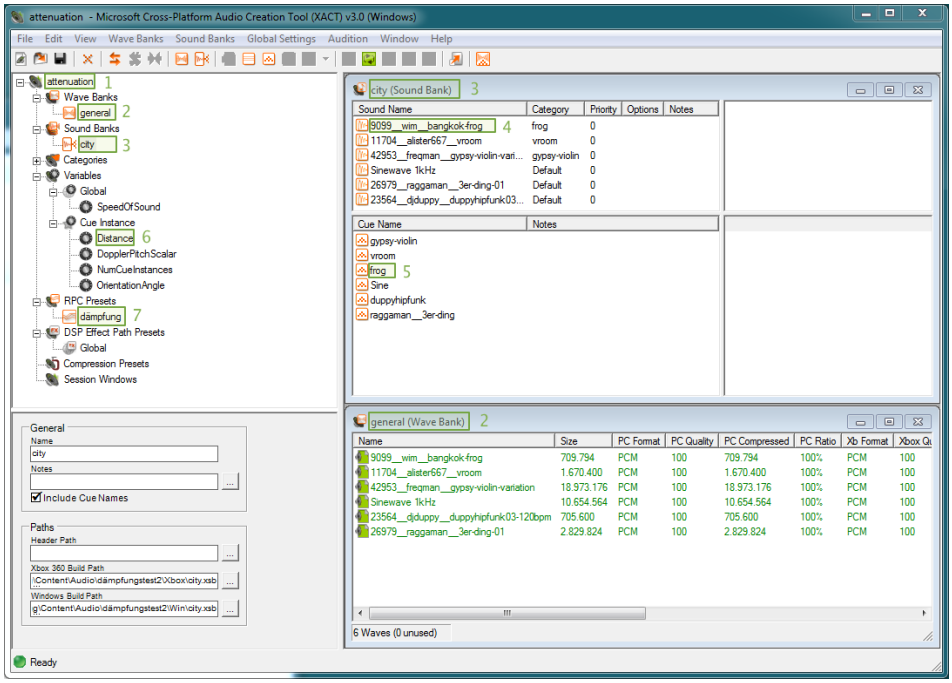


**Abbildung 5.10:** Programmoberfläche XACT Audio Creation Tool

Um die Sounds später im Prototypen verwenden zu können, müssen sie kompiliert werden („File > Build ...“). Alle unkompilierten Dateien sind im Wave Bank Fenster in roter Schrift gehalten. Alle kompilierten Dateien in grün.

(3) Man kann beliebig viele Sound Banks anlegen und sollte dies bei großen Projekten in dieser Form handhaben, um die Übersichtlichkeit zu wahren. Auch hier sollte der **Build Path wieder auf den Projektordner** festgelegt werden und **„Include Cue Names“ ausgewählt** sein (siehe Menü links unten in Abbildung 5.10).
In diesem Beispiel wurde symbolisch nur eine Sound Bank „city“ angelegt (siehe „3“ in Abbildung 5.10).

(4) Möchte man einen Sound in eine Sound Bank einsortieren, kann man diesen aus der Wave Bank in das Sound Bank Fenster ziehen. Befindet sich der Sound noch nicht in einer Cue, ist seine Schrift in rot gehalten.
Wenn man einen Sound in der Sound Bank auswählt, kann man auf links unten im XACT Tool noch verschiedene Parameter manuell einstellen. Dies sollte bei der Projekteinstellung geschehen, da sich die Paramter zur Laufzeit des Prototypen nicht mehr beeinflussen lassen. Nützlich ist es **„Looping“**

**auf „Infinite“** zu setzen. Dadurch beginnt der Sound neu zu spielen, wenn er abgelaufen ist.

(5) SoundCues sind Stapel, die normalerweise aus mehreren Sounds bestehen, von denen zufällig einer abgespielt wird, wenn die Cue angesprochen wird. Genau dieser Fall ist hier jedoch nicht erwünscht, damit auf allen Lautsprecherebenen der gleiche Sound gespielt wird.
Es sollte **für jedes Sample eine eigene Cue** mit eindeutiger Bezeichnung gewählt werden, um diesen Sound in der Raumsimulation mit einer Wahrscheinlichkeit von 100% auf jeder Ebene abzuspielen.
Um eine Cue für einen Sound zu erstellen, kann man diesen aus einer Sound Bank in den Cue-Bereich ziehen (siehe „4“ und „5“ in Abbildung 5.10).

(6) Unter **„Variables > Distance“** muss die „Variable Range“ auf einen Bereich von **„0 bis 2000“** gesetzt werden (siehe „6“ in Abbildung 5.10).
Dies ist erforderlich, damit Schallquellen erst in einer Distanz von 2000 Einheiten zum Hörer verstummen, da der normalisierte Raum des in dieser Arbeit entwickelten Lautstärkemodells in XNA (siehe Abschnitt 5.5.1, Seite 60) eine Kantenlänge von 2000 Einheiten vorweist und die Dämpfungskurve darauf ausgerichtet ist.

(7) Diese Dämpfungskurve sollte als RPC-Preset für das Projekt eingestellt werden. Im hier erläuterten Beispiel wurde das RPC-Preset mit „dämpfung“ bezeichnet und implementiert die in Abbildung 5.4 auf Seite 61 zu sehenden Werte, welche in Abschnitt 5.5.1 auf Seite 60 aufgestellt wurden.

(8) Jeder Sound der mittels dem RPC-Preset gedämpft werden soll muss noch damit verknüpft werden. Das RPC-Preset lässt sich am einfachsten auf einen Sound anwenden, indem man es aus dem Menü auf der linken Seite per Drag&Drop auf den Sound im Sound Bank - Fenster zieht.

(9) Der letzte Schritt besteht darin, das Projekt im angelegten Projektordner zu speichern und anschließend zu kompilieren („File > Build ...“).
Das Projekt lässt sich in Ubiqu3DA laden, wenn das Kompilieren fehlerfrei durchläuft und sich anschließend die folgenden Dateien im Projektordner (oder einem Unterordner von diesem) befinden:

- XACT-Projektdatei (*.xap)
- Cue List (*.scue)
- XACT-Einstellungsdatei (*.xgs)
- Sound Banks (*.xsb)
- Wave Bank (*.xwb)

## 5.8 Zusammenfassung

In diesem Kapitel wurde die Implementierung des Prototypen „Ubiqu3DA“ der Raumsimulation beschrieben. Dabei wurde anhand des Soundsystemmodells dieser Arbeit auf die einzelnen Hardware- sowie Softwarekomponenten inklusive der prototypischen Umsetzung der 3D-Sound-Anwendung eingegangen.
Es wurde das für XNA aufgestellte Lautstärkemodell erklärt und exemplarisch geschildert, wie man vorgehen muss, um ein XACT-Projekt zu erstellen, das vom Prototypen geladen und benutzt werden kann.
Im nächsten Kapitel erfolgt eine Evaluation des Prototypen anhand des im Kapitel 2 (ab Seite 5) aufgestellten Anforderungsprofils.

# 6 Evaluation des Prototypen

| | Prototyp „Ubiqu3DA“ |
|---|---|
| Verzögerungsfreiheit | *** |
| 3D-Positionierbarkeit | *** |
| Schallquellenunabhängigkeit | *** |
| Ubiquität | * |
| Wegweisungsfähigkeit | *** |
| Mehrbenutzerfähigkeit | ** |
| Koppelbarkeit | *** |
| Begleitungsfähigkeit | *** |
| Distanzierungsfähigkeit | *** |
| Anpassbarkeit | - |
| Flexibilität | *** |
| Praktikabilität | *** |
| Audioeingabefähigkeit | - |

**Tabelle 6.1:** Anforderungsevaluation des Prototyps

Anhand der Kriterien des in Abschnitt 2.4 auf Seite 16 aufgestellten Anforderungskatalogs wurde der Prototyp bewertet.
In Tabelle 6.1, auf Seite 71 ist die Bewertung im Überblick dargestellt.
„***“ bedeutet dabei, dass die Anforderung vollständig erfüllt ist.
„**“ steht für zum Teil erfüllt.
„*“ entspricht getroffenen Vorbereitungen, um die Anforderung zu erfüllen.
„-“ sagt aus, dass ein Kriterium nicht erfüllt wurde.

Im Folgenden wird auf jedes Kriterium des Anforderungskataloges eingegangen und erläutert, inwiefern es (nicht) erfüllt wurde.

**Verzögerungsfreiheit** Wird die Position einer Schallquelle, die gerade einen Klang abspielt, per Trackbar, Direkteingabe der Koordinaten oder mit dem „Trackpad“ (siehe Abbbildung 5.9, Seite 66) manipuliert, lässt sich keine Verzögerung zwischen der Eingabe der Position und der Ausgabe des Klanges im Raum wahrnehmen.
Das Kriterium *Verzögerungsfreiheit* ist somit vollständig erfüllt.

**3D-Positionierbarkeit** Mittels des Prototypen „Ubiqu3DA“ lassen sich Schallquellen an jeder Stelle des Raumes positionieren. Der Audiomanager (siehe Abschnitt 5.5, Seite 60) kümmert sich um die positionsabhängige Anpassung der Dämpfung für jede der beiden Lautsprecherebenen und ermöglicht somit auch eine vertikale Ortung der Schallquellen.
Das Kriterium *3D-Positionierbarkeit* ist dadurch vollständig erfüllt.

**Schallquellenunabhängigkeit** Jeder „SoundEmitterSceneNode“ verfügt über einen eigenen Positionsvektor mit den 3D-Koordinaten der Schallquelle. Es existiert keine theoretische Limitierung der erzeugbaren Schallquellen. Praktisch kann die Rechenleistung des Systems die Anzahl der maximalen Schallquellen einschränken.
Alle erzeugten Schallquellen können zudem unabhängig voneinander bewegt werden.
Das Kriterium *Schallquellenunabhängigkeit* ist vollständig erfüllt.

**Ubiquität** Ziel dieser Arbeit war es, einen Prototypen für einen Raum zu implementieren. „Ubiqu3DA“ lässt sich in der hier vorgestellten Version innerhalb eines Raumes einsetzen.
Das verwendete XNA-Framework bietet darüber hinaus Methoden, um eine Netzwerkfähigkeit zu implementieren und den Prototypen auf mehrere Räume zu erweitern.
Eine Umsetzung könnte sich in der Form gestalten, dass ein Server den Szenengraphen verwaltet und pro Raum ein externer Client mittels des Audiomanagers angesteuert wird. Der Client würde sich dann für den jeweiligen Raum um die Audioausgabe kümmern. Dabei könnte er Daten, wie beispielsweise die Sounds und die Positionen der Schallquellen im Raum, vom Server beziehen.
Das Kriterium *Ubiquität* wird somit nicht vom Prototypen erfüllt. Es wurden jedoch Vorbereitungen für eine spätere Implementierung getroffen.

**Wegweisungsfähigkeit** In der implementierten Raumsimulation lassen sich Richtungen beziehungsweise Wege anzeigen, indem für eine Schallquelle ein Bewegungspfad definiert wird.
So wäre es zum Beispiel denkbar, Hörer durch einen dunklen Raum zu führen, indem man einen Signalton durch den Raum laufen lässt, der Hindernisse umgeht. Der Nutzer könnte sein Ziel sicher erreichen, wenn er dem Ton folgt.
Das Kriterium *Wegweisungsfähigkeit* ist vollständig erfüllt.

**Mehrbenutzerfähigkeit** Der implementierte Prototyp sieht vor, dass mehrere Benutzer einen Raum betreten und unterschiedliche Positionen annehmen können. Zudem können an jeden Benutzer Schallquellen gekoppelt werden, die sich mit diesem (unabhängig von anderen Nutzern) mitbewegen.
Da XNA jedoch davon ausgeht, dass sich der Hörer in der Mitte aller Lautsprecher befindet (siehe Abschnitt 5.4.3, Seite 58), ist der Sweetspot aller Benutzer immer statisch in der Mitte des Raumes. Es hat sich im Laufe der Implementierung gezeigt, dass ein dynamischer Sweetspot, wie im Konzept dieser Arbeit angedacht (siehe Abschnitt 4.2, Seite 42), mit XNA nicht realisierbar ist.
Das Kriterium *Mehrbenutzerfähigkeit* ist somit nur zum Teil erfüllt.

**Koppelbarkeit** Diskrete Schallquellen lassen sich in „Ubiqu3DA“ auf einzelne Hörer im Raum ausrichten, indem man ihnen eine Position möglichst nahe des „Ziel“-Hörers zuweist und die Schallquelle danach mit diesem Hörer koppelt.
Das Kriterium *Koppelbarkeit* ist vollständig erfüllt.

**Begleitungsfähigkeit** Der Klang kann einen Hörer in der implementierten Raumsimulation dieser Arbeit über verschieden Positionen folgen, wenn man die entsprechende Schallquelle an den Hörer koppelt.
Durch diese Verbindung behält die Schallquelle immer den gleichen Abstand zum Nutzer, auch wenn sich dieser im Raum bewegt.
Das Kriterium *Begleitungsfähigkeit* ist vollständig erfüllt.

**Distanzierungsfähigkeit** Der Klang kann sich von einem Hörer entfernen, indem sich der Abstand der entsprechenden Schallquelle zum Hörer vergrößert. Dies ist beispielsweise der Fall, wenn sich Hörer und Schallquelle in unterschiedliche Richtungen bewegen.
Ab einer Distanz von 1414,21 Einheiten im normalisierten Raum zwischen Hörer und Schallquelle erfolgt eine lineare Dämpfung der Lautstärke (siehe Abschnitt 5.5.1, Seite 60).

Das Kriterium *Distanzierungsfähigkeit* ist vollständig erfüllt.

**Anpassbarkeit** Persönliche Einstellungen für den Hörer, wie zum Beispiel eine Individuallautstärke, wurden im Prototyp dieser Arbeit nicht realisiert.
Das Kriterium *Anpassbarkeit* ist somit nicht erfüllt.

**Flexibilität** Durch das im Konzept entwickelte und im Prototyp dieser Arbeit umgesetzte Soundsystemmodell (siehe Abschnitt 4.3, Seite 45) sowie der zugrunde liegenden Architektur ist die Raumsimulation nicht von einer bestimmten Hardwarekonfiguration abhängig und lässt sich flexibel einsetzen.
Das Kriterium *Flexibilität* ist vollständig erfüllt.

**Praktikabilität** Der Programmcode von „Ubiqu3DA“ und die weiteren Bestandteile des Prototyps in dieser Arbeit sind vollständig dokumentiert. Dadurch wird dem Entwickler ein Nachschlagewerk an die Hand gegeben und der Einarbeitungsaufwand in das System minimiert.
Das Kriterium *Praktikabilität* ist vollständig erfüllt.

**Audioeingabefähigkeit** Eine Eingabemöglichkeit beispielsweise mittels Mikrofon wurde für die Raumsimulation nicht realisiert.
Die Umsetzung einer Mikrofoneingabe in XNA kann jedoch durch eine Implementierung der „SoundEffect“- (siehe Abschnitt 5.4.1, Seite 56) beziehungsweise der davon abgeleiteten „DynamicSoundEffectInstance“- Klasse erfolgen. Dadurch lassen sich Sounds auch zur Laufzeit aus Puffern streamen, was eine Ausgabe der Mikrofoneingabe über die Lautsprecher in Echtzeit ermöglicht.
Der Nachteil dieser Implementierung ist, dass sich nur die in Windows als Standard festgelegte Soundkarte ansprechen lässt, um den Klang auszugeben.
Das Kriterium *Audioeingabefähigkeit* ist nicht erfüllt.

# 7 Zusammenfassung und Ausblick

In dieser Arbeit wurde ein ubiquitäres 3D-Sound-System konzipiert und entwickelt.
Das Raummodell in Form eines Szenengraphen beschreibt dabei, wie Schallquellen im Raum repräsentiert werden und sich mehrere Räume verknüpfen lassen.
In den Graphen lässt sich ein Benutzermodell einfügen, welches definiert wie sich ein (gewichteter) Sweetspot für das Hauptgeschehen errechnen lässt, wenn sich mehrere Hörer in einem Raum aufhalten. Zudem wurde darauf eingegangen, wie sich diskrete an den Hörer gekoppelte Schallquellen realisieren lassen.
Anhand eines Soundsystemmodells wurde eine Hard- und Softwarehierarchie beschrieben, die es ermöglicht alle Komponetenebenen flexibel auszutauschen.
Die Architektur des Modells legt fest, wie das Raum-, Benutzer- und Soundsystemmodell zusammenwirken und dadurch das 3D-Sound-System-Modell formen.

Das entwickelte Modell wurde für einen Raum mit zwei 7.1 Soundsystemen prototypisch implementiert.
Mit diesem Prototyp ist es möglich, Schallquellen unabhängig voneinander zu erzeugen und durch den Raum zu bewegen. Sie können dem Anwender folgen, sich von ihm entfernen oder ihm Wege durch den Raum aufzeigen.

## 7.1 Weiterentwicklung

In diesem Abschnitt wird ein Ausblick gegeben, in welche Richtung der Prototyp weiterentwickelt werden kann.

### 7.1.1 Tracking der Benutzer

Durch die Implementierung eines Benutzertrackings ließen sich sämtliche Positionseingaben für die Benutzer des Systems optimieren. Eine manuelle Positionie-

rung der Hörer, wie im Prototypen dieser Arbeit, wäre dadurch überflüssig. Sobald ein Nutzer seine Position ändert, könnte das Trackingsystem eine Anpassung des entsprechenden Positionsvektors auslösen. Voraussetzung für den Mehrbenutzerbetrieb wäre eine eindeutige Identifizierbarkeit jedes einzelnen Hörers im Raum.

### 7.1.2 Weitere Lautsprecherebenen

Das 3D-Sound-System könnte um zusätzliche Lautsprecherebenen erweitert werden. Dadurch ließe sich der Abstand zwischen den einzelnen Lautsprechern verkleinern und die Präzision der Abbildung des Klanges im Raum erhöhen.
Vor allem sehr große Räume sollten davon profitieren.

### 7.1.3 Verbesserung der Mehrbenutzerfähigkeit

Um die Mehrbenutzerfähigkeit auszubauen, könnte ein Audiomanager implementiert werden, der Schallquellen im Raum mittels Wellenfeldsynthese (WFS) positioniert. Dadurch könnte man den Sweetspot über den kompletten Raum erstrecken und jedem Anwender den gleichen Höreindruck ermöglichen.
Ein Nachteil der WFS ist jedoch der hohe Rechenaufwand und die Anzahl der für einen realistischen Eindruck benötigen Lautsprecher. Das kommerzielle System IOSONO nutzt beispielsweise mindestens 32 Lautsprecher (eine Hardwareeinheit), um ein 2D-Schallfeld zu erzeugen [TCK09].
Theodoropoulos et al. haben Vergleiche mit 8, 16, 24 und 32 Lautsprechern bei Berechnung der WFS mittels CPU, GPU und FPGA durchgeführt und gezeigt, dass sich WFS gut parallelisieren und dadurch sehr effizient mittels Grafikkarten und FPGAs berechnen lässt [TCK09].

### 7.1.4 Erweiterung auf mehrere Räume

Um den implementierten Prototypen zu einem ubiquitären 3D-Sound-System zu erweitern, ist es erforderlich mehr als einen Raum simulieren zu können. Dazu könnte der Prototyp zu einem Server umgestaltet werden, der den AudioManager jeden Raumes über ein Netzwerk anspricht. In jedem Raum sollte ein Client existieren, auf dem der Audiomanager implementiert ist, der sich um die Audioausgaben kümmert und seine Daten vom Server bezieht.

Zusammenfassend lässt sich feststellen, dass vor allem ein automatisches Tracking der Benutzer im Raum die Benutzbarkeit des 3D-Sound-System wesentlich verbessern könnte, da die Positionen der Nutzer nicht mehr „von Hand“ festgelegt werden müssten.

# Abbildungsverzeichnis

# Tabellenverzeichnis

# Literaturverzeichnis


[Aar02] Aaron E. Walsh. *dr. dobb's | understanding scene graphs | july 01, 2002*. 2002. URL: http://www.drdobbs.com/article/print?articleId=184405094\&siteSectionName=jvm (siehe Seite 7).

[Beg00] DR Begault. *3-D Sound for Virtual Reality and Multimedia*. April. Cambridge, MA: Academic Press Professional, 2000, Seite 246 (siehe Seite 61).

[BS11] Jean Bresson und Marlon Schumacher. *REPRESENTATION AND INTERCHANGE OF SOUND SPATIALIZATION DATA FOR COMPOSITIONAL APPLICATIONS*. 2011 (siehe Seite 20).

[Cre] CreativeLabs. *OpenAL*. URL: http://connect.creativelabs.com/openal/default.aspx (siehe Seite 27).

[Fai] Fairaudio.de. *HiFi Lexikon - fairaudio*. URL: http://www.fairaudio.de/lexikon.html (siehe Seiten 9, 10, 15).

[Gay02] Lalya Gaye. „A flexible 3d sound system for interactive applications“. In: *CHI '02 extended abstracts on Human factors in computing systems - CHI '02*. New York, New York, USA: ACM Press, Apr. 2002, Seite 840. ISBN: 1581134541. DOI: 10.1145/506443.506625. URL: http://dl.acm.org/citation.cfm?id=506443.506625 (siehe Seite 2).

[Gul] Jens Gulden. *JJack*. URL: http://jjack.berlios.de/ (siehe Seite 30).

[HDM03] H Hoffmann, R Dachselt und K Meissner. *An independent declarative 3D audio format on the basis of XML*. 2003. URL: http://www.icad.org/node/2686 (siehe Seiten 23, 26, 38).

[HKMA07] Tobias Höllerer, JoAnn Kuchera-Morin und Xavier Amatriain. *The allosphere: a large-scale immersive surround-view instrument*. Aug. 2007. DOI: 10.1145/1278240.1278243. URL: http://dl.acm.org/citation.cfm?id=1278240.1278243 (siehe Seite 2).

[HVV98] P M Hofman, J G Van Riswick und a J Van Opstal. „Relearning sound localization with new ears.“ In: *Nature neuroscience* 1.5 (Sep. 1998), Seiten 417–21. ISSN: 1097-6256. DOI: 10.1038/1633. URL: http://www.ncbi.nlm.nih.gov/pubmed/10196533 (siehe Seite 14).

[Ilm02] Tommi Ilmonen. *Mustajuuri – a sound processing application and toolkit*. 2002 (siehe Seite 19).

[Int98] Interactive Audio Special Interest Group. *3D Audio Rendering and Evaluation Guidelines - Level 1.0*. 1998 (siehe Seiten 5, 6, 8, 10, 12–14).

[JAC] JACK. *JACK Audio*. URL: http://jackaudio.org/ (siehe Seite 29).

[JOA] JOAL. *Java Bindings for the OpenAL API*. URL: http://jogamp.org/joal/www/ (siehe Seite 28).

[KST05] Eric Klein, GS Schmidt und EB Tomlin. „Dirt cheap 3D spatial audio“. In: *Linux Journal* (2005) (siehe Seiten 2, 19).

[Lee04] Newton Lee. „IOSONO“. In: *Computers in Entertainment* 2.3 (Juli 2004), Seite 3. ISSN: 15443574. DOI: 10.1145/1027154.1027162. URL: http://dl.acm.org/citation.cfm?id=1027154.1027162 (siehe Seiten 2, 19).

[LS99] Maruricio Lumbreras und Jaime Sánchez. „Interactive 3D sound hyperstories for blind children“. In: *Proceedings of the SIGCHI conference on Human factors in computing systems the CHI is the limit - CHI '99*. New York, New York, USA: ACM Press, Mai 1999, Seiten 318–325. ISBN: 0201485591. DOI: 10.1145/302979.303101. URL: http://dl.acm.org/citation.cfm?id=302979.303101 (siehe Seiten 22, 26).

[McG10] R McGee. „Sound Element Spatializer“. Dissertation. 2010 (siehe Seite 20).

[Mic] Microsoft. *XNA Developer Center*. URL: http://msdn.microsoft.com/en-us/centrum-xna.aspx (siehe Seite 30).

[Pop] Stephen Pope. *CREATE Signal Library (CSL) Project*. URL: http://fastlabinc.com/CSL/ (siehe Seite 28).

[sof] Id software. *id Software*. URL: http://www.idsoftware.com/games/doom/ (siehe Seite 1).

[SPS+09] Oliver Schmidt, Ron Porath, Martin Schmid u. a. *Physik & Musik 7*. 2009. URL: http://www.educ.ethz.ch/unt/um/phy/sw/musik/p07.pdf (siehe Seite 11).

[Sta03] Philipp Stampfl. *3deSoundBox – a Scalable , Platform-Independent 3D Sound*. 2003 (siehe Seite 21).

[Tao] Tao. *Tao*. URL: http://www.mono-project.com/Tao (siehe Seite 28).

[TCK09] Dimitris Theodoropoulos, Catalin Bogdan Ciobanu und Georgi Kuzmanov. „Wave field synthesis for 3D audio“. In: *Proceedings of the 6th ACM conference on Computing frontiers - CF '09*. New York, New York, USA: ACM Press, Mai 2009, Seite 127. ISBN: 9781605584133. DOI: 10.1145/1531743.1531764. URL: http://dl.acm.org/citation.cfm?id=1531743.1531764 (siehe Seite 76).

[TU ] TU Berlin. *jReality*. URL: http://www3.math.tu-berlin.de/jreality/ (siehe Seite 29).

[Un4] Un4seen. *BASS*. URL: http://www.un4seen.com/ (siehe Seite 30).

[Wei93] Mark Weiser. „Some computer science issues in ubiquitous computing“. In: *Communications of the ACM* 36.7 (Juli 1993), Seiten 75–84. ISSN: 00010782. DOI: 10.1145/159544.159617. URL: http://dl.acm.org/citation.cfm?id=159544.159617 (siehe Seite 6).

[ZA] Kokkini Zita und Fons Adriaensen. *ambdec*. URL: http://kokkinizita.linuxaudio.org/ambisonics/index.html (siehe Seiten 29, 30).