Bachelorarbeit

Distance based amplitude panning

Markus Maier

Universität für Musik und darstellende Kunst Graz

Institut für Elektronische Musik und Akustik

Betreuer: Univ.-Prof. Dr. Gerhard Eckel

Graz, September 2012

Inhaltsverzeichnis

Einleitung 4

1 Konzeptionelle Grundlagen 6

1.1 Grundgleichungen . . . 6

1.1.1 Definitionen . . . 6

1.1.2 Relative Amplitude . . . 7

1.1.3 Spatial Blur . . . 9

1.2 Erweiterungen . . . 11

1.2.1 Gewichtung . . . 11

1.2.2 Delay . . . 12

1.2.3 Konvexe Hülle . . . 13

2 Implementation für Pure Data 15

2.1 External . . . 15

2.1.1 Portierung . . . 16

2.1.2 Änderungen . . . 16

2.1.3 Erweiterungen . . . 18

2.1.4 Verwendung . . . 20

2.2 Patches . . . 24

2.2.1 Help-Patch . . . 24

2.2.2 Signal-Abstraction . . . 26

3 Konvexe Hülle **28**
3.1 Speicherkonzept . . . 28
3.2 Konstruktion . . . 29
3.2.1 Nulldimensionale Hülle . . . 31
3.2.2 Eindimensionale Hülle . . . 32
3.2.3 Zweidimensionale Hülle . . . 33
3.2.4 Dreidimensionale Hülle . . . 37
3.3 Projektion . . . 42
3.3.1 Nulldimensionale Hülle . . . 43
3.3.2 Eindimensionale Hülle . . . 44
3.3.3 Zweidimensionale Hülle . . . 46
3.3.4 Dreidimensionale Hülle . . . 48

4 Vergleichstest **51**
4.1 Vergleich der Lautsprecheramplituden . . . 51
4.2 Positionsbestimmung durch Versuchspersonen . . . 52
4.3 Reflexion . . . 53

Literatur **55**

Einleitung

Die Spatialisationsmethode[1] Distance Based Amplitude Panning (kurz DBAP) wurde 2009 bei der International Computer Music Conference von den Entwicklern Trond Lossius (BEK[2]), Pascal Baltazar (GMEA[3]) und Théo de la Hogue (GMEA) vorgestellt. Unabhängig davon arbeitete ein zweites Team bestehend aus Dimitar Kostadinov (TEE[4]) und Joshua D. Reiss (CDM[5]) im gleichen Jahr ebenfalls an einer Formulierung des selben Prinzips, anfangs noch unter dem Namen Vector Distance Panning (kurz VDP).

Ziel von DBAP ist es, eine beliebige Anzahl von Lautsprechern in beliebiger Anordnung so anzusteuern, dass der Eindruck entsteht, eine oder mehrere virtuelle Schallquellen befänden sich an beliebig vorgebbaren Positionen im Raum. Die Spatialisationstechnik arbeitet nach dem Amplitude-Panning-Prinzip, die Positionierung der virtuellen Schallquellen wird also über die Amplituden der einzelnen Lautsprecher gesteuert. Somit kann DBAP als Erweiterung des Stereo-Prinzips auf bis zu drei Dimensionen aufgefasst werden. Die Amplituden der Lautsprecher werden dabei wenig rechenintensiv auf Basis der Abstände der virtuellen Schallquellen zu den Lautsprechern ermittelt.

Diese Methode bietet damit die Möglichkeiten, virtuelle Schallquellen in 1D, 2D oder 3D innerhalb oder am Rand der Lautsprecher-Anordnung zu positionieren (bzw. mit der Erweiterung durch eine konvexe Hülle mit Einschränkungen sogar außerhalb). Außerdem ist der Hörort nicht an einen Sweet-Spot gebunden, und das Spatialisationsergebnis ist weitgehend unabhängig von der Zuhörerposition, solange diese nicht näher an der nächsten Berandung als die spatialisierte virtuelle Schallquelle liegt.

[1] Die Positionierung von Schallquellen in einem Raum wird Spatialisation genannt.
[2] Bergen senter for Elektronisk Kunst, Norwegen
[3] Groupe de Musique Electroacoustique d'Albi-Tarn, Centre de Création Musicale, Frankreich
[4] Dept. of Theoretical Electrical Engineering, Technical University of Sofia, Bulgarien
[5] Centre for Digital Music, Queen Mary University of London, England

In Kapitel 1 sollen nun die konzeptionellen Grundlagen für die Anwendung in drei Raumdimensionen besprochen werden. Darauf basierend befassen sich die beiden anschließenden Kapitel mit der Implementation der Grundfunktionen inklusive Gewichtungsfaktoren (Kapitel 2) und der Erweiterung durch die sogenannte konvexe Hülle (Kapitel 3). Zum Abschluss wird in Kapitel 4 ein Vergleichstest einer bereits existierenden Implementation mit anderen Spatialisationsmethoden reflektiert.

1 Konzeptionelle Grundlagen

In diesem Kapitel soll das zu Grunde liegende Konzept mit seinen Grundgleichungen (Abschnitt 1.1) und Erweiterungen (Abschnitt 1.2) für DBAP in 3D mit beliebig vielen Lautsprecher- und Quellpositionen auf Basis von *DBAP – Distance-based amplitude panning* [LBH09] und *Calculation of DBAP equations* [Los10] abgeleitet und erklärt werden.

1.1 Grundgleichungen

1.1.1 Definitionen

Als Grundlage der Berechnungen dienen die Positionen der Lautsprecher und virtuellen Quellen als Punkte in einem kartesischen Koordinatensystem. Die Koordinaten bzw. Ortsvektoren der N_L Lautsprecher- und N_S Quell-Positionen seien in der folgenden Form gegeben.

$$L_i = \begin{pmatrix} x_i \\ y_i \\ z_i \end{pmatrix}, \ 1 \leq i \leq N_L \qquad S_j = \begin{pmatrix} x_j \\ y_j \\ z_j \end{pmatrix}, \ 1 \leq j \leq N_S$$

Dabei wird in dieser Arbeit der Index i immer für Lautsprecher, der Index j immer für Quellen verwendet.

Der Abstand d_{ij} des i-ten Lautsprechers zur j-ten Quelle kann nun mit Hilfe der Vektorrechnung ermittelt werden. Dazu berechnet man über die euklidische Norm die Länge des Vektors, dessen Endpunkte durch die Quelle und den Lautsprecher definiert sind.

$$d_{ij} = \left\| \vec{S_j} - \vec{L_i} \right\| = \left\| \begin{pmatrix} x_j - x_i \\ y_j - y_i \\ z_j - z_i \end{pmatrix} \right\| = \sqrt{(x_j - x_i)^2 + (y_j - y_i)^2 + (z_j - z_i)^2} \tag{1.1}$$

Da DBAP als Erweiterung des Stereo-Panning-Prinzips konzipiert ist, wird auch hier gefordert, dass die über alle Lautsprecher summierte Gesamtintensität I_j einer virtuellen Quelle konstant sein soll.

$$I_j = \sum_{n=1}^{N_L} I_{nj} \stackrel{!}{=} C \tag{1.2}$$

1.1.2 Relative Amplitude

DBAP verwendet nun den Ansatz, dass die relative Amplitude g_{ij} des i-ten Lautsprechers aufgrund der j-ten virtuellen Quelle indirekt proportional dem Abstand der beiden Punkte ist.

$$g_{ij} = \frac{k}{d_{ij}^a} \tag{1.3}$$

Dabei steht der Anordnungskoeffizient k für die Gegebenheiten durch die Anordnung und eventuelle Gewichtung (siehe dazu Abschnitt 1.2.1) der Lautsprecher, die Umgebungskonstante a für die Einflüsse der Umgebung, in der die Lautsprecher aufgestellt sind.

Anordnungskoeffizient k

Dieser Koeffizient soll dafür sorgen, dass die Forderung der konstanten Gesamtintensität (Gleichung 1.2) erfüllt wird. Vereinfachend werden diese Konstante C und der Proportionalitätsfaktor beim Übergang von Amplituden auf Intensitäten als 1 angenommen.

$$I_j = \sum_{n=1}^{N_L} g_{nj}^2 \stackrel{!}{=} 1 \tag{1.4}$$

Setzt man nun den Ansatz für die relative Amplitude (Gleichung 1.3) ein, kann daraus der Anordnungskoeffizient k berechnet werden.

$$\sum_{n=1}^{N_L} \left(\frac{k}{d_{nj}^a}\right)^2 = k^2 \sum_{n=1}^{N_L} \frac{1}{d_{nj}^{2a}} = 1 \qquad \Rightarrow \qquad k = \frac{1}{\sqrt{\sum_{n=1}^{N_L} \frac{1}{d_{nj}^{2a}}}} \tag{1.5}$$

Dieses Ergebnis kann anschließend wieder mit dem Ansatz für die relative Amplitude (Gleichung 1.3) zusammengeführt werden, und liefert somit die vollständige Gleichung für die relativen Amplituden.

$$g_{ij} = \frac{1}{\sqrt{\sum_{n=1}^{N_L} \frac{1}{d_{nj}^{2a}}}} \frac{1}{d_{ij}^a} = \frac{1}{\sqrt{\sum_{n=1}^{N_L} \frac{d_{ij}^{2a}}{d_{nj}^{2a}}}} \tag{1.6}$$

Umgebungskonstante a

Mit dieser Konstante soll die Abnahme der Amplitude bei der Schallausbreitung in einer bestimmten Umgebung modelliert werden. Im Freiraum ist die Amplitude in doppelter Entfernung nur noch halb so groß, d.h. der positiv definierte Rolloff pro doppelter Entfernung beträgt 6 dB, in geschlossenen Räumen etwa 4 bis 5 dB. Um die Umgebungskonstante a abhängig vom Rolloff pro doppelter Entfernung zu berechnen, sei die in Abbildung 1.1 gezeigte Anordnung von Punkten angenommen.

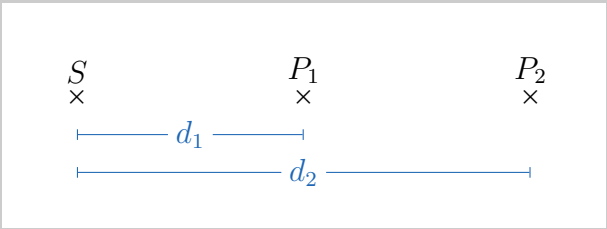


Abbildung 1.1: Anordnung von Punkten zur Berechnung der Umgebungskonstante a

Am Punkt S befindet sich die Schallquelle, an P_1 und P_2 wird die Schallamplitude betrachtet. Da P_2 im Vergleich zu P_1 doppelt so weit von S entfernt ist, muss die Amplitude G_2 (in dB) um den Rolloff R pro doppelter Entfernung niedriger als G_1 sein.

$$G_2 = G_1 - R \tag{1.7}$$

Die Amplitude in dB kann über den Logarithmus der relativen Amplitude dargestellt werden,

$$-R = G_2 - G_1 = 20 \log g_2 - 20 \log g_1 = 20 \log \frac{g_2}{g_1} \tag{1.8}$$

und der Ansatz für die relative Amplitude (Gleichung 1.3) in einem bestimmten Abstand sowie das Verhältnis der Abstände $d_2 = 2d_1$ werden eingesetzt.

$$-R = 20 \log \frac{\frac{k}{d_2^a}}{\frac{k}{d_1^a}} = 20 \log \frac{d_1^a}{d_2^a} = 20 \log \frac{d_1^a}{(2d_1)^a} = 20a \log \frac{1}{2} = -20a \log 2 \tag{1.9}$$

Durch Umformung kann nun die Bestimmungsgleichung für die Umgebungskonstante a abhängig vom Rolloff R pro doppelter Entfernung angegeben werden.

$$a = \frac{R}{20 \log 2} \tag{1.10}$$

1.1.3 Spatial Blur

Für den Fall, dass sich eine bestimmte virtuelle Quelle S_J exakt an der Position eines bestimmten Lautsprechers L_I befindet, ergibt sich bei der Berechnung der relativen

Amplituden nach Gleichung 1.6 im Nenner eine Division durch Null, da die Distanz d_{IJ} der beiden Punkte im Nenner des entsprechenden Summenterms Null wird. Es stellt sich aber im Grenzwert die folgende Amplitudenverteilung bezüglich der Quelle S_J ein.

$$\lim_{d_{IJ}\to 0} g_{iJ} = \begin{cases} 1 & \text{für } i = I \\ 0 & \text{für } i \neq I \end{cases} \tag{1.11}$$

Wandert also eine Quelle an die Position eines Lautsprechers, ist bei exaktem Zusammenfallen der Koordinaten plötzlich nur noch dieser Lautsprecher aktiv, und alle anderen liefern keinen Beitrag für diese Quelle mehr. Dieses Verhalten wirkt sich aber möglicherweise negativ auf das Hörereignis aus, da bei Abschaltung der anderen Lautsprecher der Eindruck entstehen könnte, dass die virtuelle Schallquelle in sich zusammenfällt. Zudem könnte sich auch eine Veränderung der Klangfarbe ergeben, da sich dann nicht mehr die (durch Bautoleranzen bedingte) akustische Färbung aller Lautsprecher mischt, sondern nur mehr jene eines einzigen Lautsprechers zum Tragen kommt. Somit könnte beim Übergang zwischen verschiedenen Lautsprechern eine Klangfarbenveränderung deutlich wahrnehmbar und damit störend werden.

Um diese möglichen negativen Beeinflussungen weitestgehend zu vermeiden, wird die Berechnung des Abstandes (Gleichung 1.1) um den sogenannten Spatial Blur b_j erweitert.

$$d_{ij} = \sqrt{(x_j - x_i)^2 + (y_j - y_i)^2 + (z_j - z_i)^2 + {b_j}^2} \tag{1.12}$$

Für den Anwender kann der Spatial Blur als eine Art leichte Unschärfe (engl. blur) der Quelle interpretiert werden. Mathematisch entspricht dieser Ansatz einer Verschiebung der Quelle S_j um b_j in eine hinzugefügte Dimension. Somit kann diese Quelle nicht mehr exakt mit einem Lautsprecher zusammenfallen, da sich die Koordinaten zumindest in dieser zusätzlichen Dimension um b_j unterscheiden. Der Spatial Blur darf allerdings nur relativ klein gewählt werden, denn je größer die Verschiebung in die zusätzliche Dimension wird, desto ähnlicher werden die Abstände der virtuellen Schallquelle zu den einzelnen Lautsprechern, und die Quelle würde ungewollt in Richtung Mitte des Arrays gezogen.

Einem Nachteil dieses Ansatzes gilt es aber noch entgegenzuwirken. Geht man nämlich bei der Eingabe der Koordinaten zum Beispiel von Meter auf Zentimeter über, müsste auch der Spatial Blur um den Faktor 100 vergrößert werden, damit sich die berechneten relativen Amplituden g_{ij} nicht ändern. Deshalb wird der Zusatzterm b_j in Gleichung 1.12 nach *Source Code of a DBAP External for Max/MSP* [LBH12] durch eine korrigierte Größe $\tilde{b_j}$ ersetzt.

$$d_{ij} = \sqrt{(x_j - x_i)^2 + (y_j - y_i)^2 + (z_j - z_i)^2 + \tilde{b_j}^2} \tag{1.13}$$

Diese korrigierte Größe $\tilde{b_j}$ entspricht einer Gewichtung des Spatial Blur b_j mit der totalen korrigierten Stichproben-Standardabweichung der Lautsprecher-Positionen, wobei $\bar{x}$, $\bar{y}$ und $\bar{z}$ die Koordinaten des über alle Lautsprecher-Positionen gemittelten Punktes $\bar{L}$ darstellen.

$$\tilde{b_j} = b_j \cdot \sqrt{\frac{1}{N_L - 1} \sum_{n=1}^{N_L} \left((x_n - \bar{x})^2 + (y_n - \bar{y})^2 + (z_n - \bar{z})^2 \right)} \tag{1.14}$$

1.2 Erweiterungen

Um DBAP nun weiter zu verbessern, kann die bisher abgeleitete Basisversion durch die folgenden Konzepte zusätzlich erweitert werden.

1.2.1 Gewichtung

Durch die Einführung von Gewichtungsfaktoren w_{ij} für jede Lautsprecher-Quelle-Kombination lässt sich eine Vielzahl an neuen Möglichkeiten eröffnen. So können bestimmte Quellen zum Beispiel auf Teilbereiche von Lautsprecheranordnungen beschränkt, oder Überblendungen zwischen verschiedenen Teilräumen mittels dynamischer Gewichtungsfaktoren realisiert werden. Außerdem ist durch diese Erweiterung auch eine eventuell notwendige Anpassung von Kompositionen oder Klanginstallationen an neue räumliche Gegebenheiten deutlich einfacher möglich.

Dazu wird der Ansatz für die relativen Amplituden (Gleichung 1.3) um den Gewichtungsfaktor w_{ij} erweitert.

$$g_{ij} = \frac{k}{d_{ij}^a} w_{ij} \tag{1.15}$$

Die angepasste Berechnung für den Anordnungskoeffizient k ergibt dann analog zu Abschnitt 1.1.2

$$k = \frac{1}{\sqrt{\sum_{n=1}^{N_L} \frac{w_{nj}^2}{d_{nj}^{2a}}}} \tag{1.16}$$

und führt somit zu folgender Bestimmungsgleichung für die relativen Amplituden.

$$g_{ij} = \frac{1}{\sqrt{\sum_{n=1}^{N_L} \frac{w_{nj}^2}{d_{nj}^{2a}}}} \frac{w_{ij}}{d_{ij}^a} = \frac{w_{ij}}{\sqrt{\sum_{n=1}^{N_L} \frac{d_{ij}^{2a} w_{nj}^2}{d_{nj}^{2a}}}} \tag{1.17}$$

1.2.2 Delay

Ist ein bestimmter Hörort H vorgesehen, kann eine weitere Verbesserung erreicht werden, indem die Lautsprechersignale so verzögert werden, dass alle Schallwellen gleichzeitig am Hörort eintreffen.

Die dafür notwendige Delay-Zeit t_i für den jeweiligen Lautsprecher kann aus den Abständen der einzelnen Lautsprecher zum Hörort unter Zuhilfenahme von Gleichung 1.1 ermittelt werden. Um t_i in Sekunden zu erhalten wird die Differenz der Abstände des am weitesten entfernten Lautsprechers und des i-ten Lautsprechers vom Hörort durch die Schallgeschwindigkeit c_s (in Meter pro Sekunde) dividiert.

$$t_i = \frac{\max(d_{1H}, \ \ldots, \ d_{N_L H}) - d_{iH}}{c_s} \tag{1.18}$$

1.2.3 Konvexe Hülle

Mit DBAP in seiner bisher abgeleiteten Form können virtuelle Schallquellen innerhalb und am Rand der Lautsprecher-Anordnung spatialisiert werden (siehe Abbildung 1.2a bzw. 1.2b). Versucht man allerdings, eine Quelle außerhalb des Arrays zu positionieren, stößt das bisherige Konzept an seine Grenzen (Abbildung 1.2c).

Mit zunehmender Entfernung der gewünschten Quellposition zum Lautsprecher-Array werden die Abstände der Quelle zu den einzelnen Lautsprechern – und damit auch die berechneten relativen Amplituden – immer ähnlicher. Der resultierende Spatialisationsort wandert deshalb angefangen vom Rand der Anordnung wieder zurück in Richtung Mitte[1], wo eine unendlich weit entfernte Quellposition spatialisiert würde.

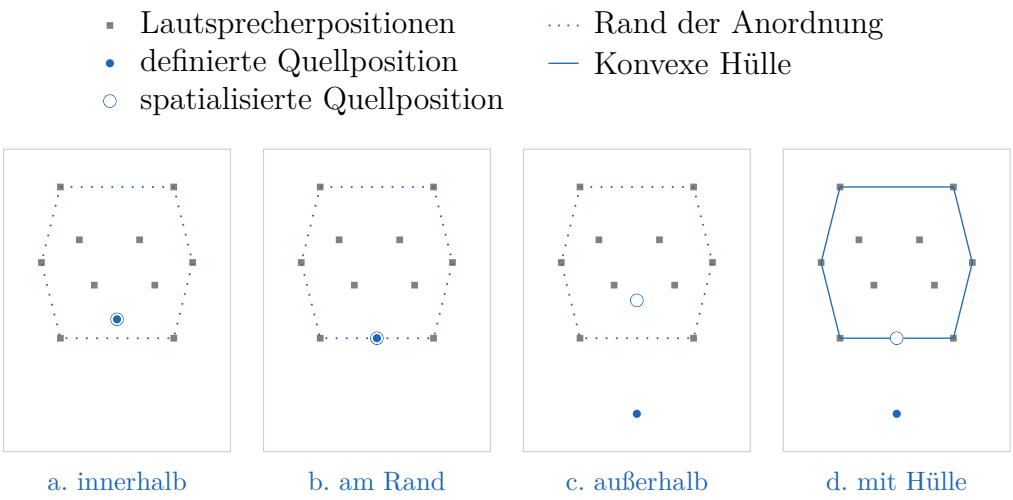


Abbildung 1.2: Spatialisierungsergebnisse mit und ohne Erweiterung durch eine konvexe Hülle

Um dieses unerwünschte Ergebnis zu vermeiden, ist eine Erweiterung des bisherigen Konzepts durch die sogenannte konvexe Hülle erforderlich. Für eine exakte Definition sei auf das Buch *Computational Geometry in C* [ORo98, Kapitel 3] verwiesen, das unter anderem auch die folgende, gut vorstellbare Beschreibung [ORo98, S. 63] enthält.

[1] Die Mitte der Lautsprecher-Anordnung entspricht dem Mittelwert aller Lautsprecherpositionen.

> *The convex hull of a set of points in the plane is the shape taken by a rubber band stretched around nails pounded into the plane at each point.*
>
> *The boundary of the convex hull of points in three dimensions is the shape taken by a plastic wrap stretched tightly around the points.*

Bei der Verwendung von DBAP mit der Erweiterung durch die konvexe Hülle wird diese nun zuerst aus den definierten Lautsprecher-Punkten berechnet. Bei der Positionierung einer virtuellen Schallquelle wird dann überprüft, ob sich deren Koordinaten innerhalb oder außerhalb der Hülle befinden. Im ersten Fall sind keine Änderungen nötig, da hier ja bereits das Grundmodell in der Lage ist, die Quelle korrekt zu spatialisieren. Selbiges gilt auch für den Grenzfall einer genau am Rand liegenden Quelle. Für den Fall, dass sich die Quelle außerhalb der Hülle befindet, wird sie auf kürzest-möglichem Weg auf die Hülle projiziert und dort spatialisiert (Abbildung 1.2d).

Damit bleibt nun die Richtung der außerhalb definierten Quelle für Hörorte innerhalb des Arrays weitgehend erhalten. Um auch die Entfernung der virtuellen Schallquelle zu kodieren, könnte zusätzlich die Distanz der ursprünglich definierten, außerhalb liegenden Position zum Projektionspunkt für ein eventuelles Postprocessing genutzt werden.

Die Algorithmen zur Konstruktion der konvexen Hülle aus den Lautsprecherkoordinaten sowie die Feststellung der Quellposition bezüglich der Hülle und die Vorgänge einer eventuellen Projektion werden im Implementationsteil zur konvexen Hülle (Kapitel 3) erläutert.

2 Implementation für Pure Data

Den größten Teil dieser Arbeit stellte die Implementierung von DBAP für Pure Data[1], kurz Pd, dar (für weiterführende Informationen zu Pd siehe [Pd]). Zum einen wurde ein External für Pd programmiert (Abschnitt 2.1), welches die DBAP-Funktionalitäten bereitstellt und als Objekt in Pd eingebunden werden kann. Zur Demonstration und Dokumentation des Externals wurde auch ein Help-Patch[2] angelegt (Abschnitt 2.2.1). Aufbauend auf dem External wurde zudem eine Signal-Abstraction[3] (ebenfalls mit zugehörigem Help-Patch) erstellt, die einen schnellen Einstieg in die Verwendung des Externals erleichtern soll (Abschnitt 2.2.2).

2.1 External

Das External basiert auf einer bereits vorhandenen Implementation *Source Code of a DBAP External for Max/MSP* [LBH12] des Entwicklerteams Trond Lossius, Pascal Baltazar und Théo de la Hogue für Max/MSP[4] und wurde in der Programmiersprache C verfasst. Da die Max/MSP-Implementation einer BSD Lizenz[5] unterliegt, hätte sie rechtlich betrachtet direkt portiert werden können, was auch zum Teil geschah (Abschnitt 2.1.1), einiges wurde aber auch aus verschiedenen Gründen verändert (Abschnitt 2.1.2). Außerdem wurde das Pd-External um einige Funktionalitäten, allen voran die konvexe Hülle in 3D, erweitert (Abschnitt 2.1.3).

[1] Pure data ist eine echtzeitfähige grafische Programmiersprache für Multimedia-Anwendungen.
[2] Pd-Dateien werden auch Patches genannt.
[3] Eine Abstraction ist ein Patch, der eine bestimmte Funktionalität bereitstellt, und in andere Patches als Objekt eingebunden werden kann.
[4] Max/MSP ist eine mit Pd eng verwandte echtzeitfähige grafische Programmiersprache.
[5] http://opensource.org/licenses/BSD-3-Clause

2.1.1 Portierung

Die Portierung umfasste den Großteil der Grundstruktur und des grundsätzlichen Programmablaufs des Externals. Eine zentrale Vorgabe war dabei, die Schnittstellen zur Steuerung und die daraus resultierenden Reaktionen des Programms aus Kompatibilitätsgründen nicht zu verändern.

Im Zuge der Portierung wurden gewisse Abkürzungen übernommen, die teilweise auch in den nachfolgenden Abschnitten verwendet werden. Für virtuelle Quellen wurde das Kürzel »src« (von engl. source) gewählt, als Pendant dazu steht »dst« (engl. destination) für Lautsprecher, und als Abkürzung für Anzahl wird »num« (engl. number) verwendet.

Aufgrund der Ähnlichkeit zwischen Max/MSP und Pd war es möglich, viele der verwendeten Code-Teile (fast alle Benutzereingaben und die kleineren internen Berechnungen) der Max/MSP-Implementation mit kleinen Anpassungen zu übernehmen, abgesehen von den bewussten Veränderungen, die im folgenden Abschnitt erläutert werden.

2.1.2 Änderungen

Speicherkonzept

Im Unterschied zur bereits vorhandenen Implementation, die auf eine statische Speicherverwaltung setzt, verfolgt die Pd-Version ein dynamisches Speicher-Management-Konzept. Während das Max/MSP-External etwa bereits beim Laden den gesamten Daten-Speicher für die maximale Anzahl an Lautsprechern und virtuellen Quellen anlegt (auch wenn nur wenige Lautsprecher bzw. Quellen verwendet werden), wird beim Pd-External dynamisch nur jener Speicher angefordert, der auch wirklich benötigt wird.

Wird also beispielsweise die Anzahl der virtuellen Quellen erhöht, wird damit auch zusätzlicher Speicher für die zugehörigen Daten (Koordinaten, Gain, Mute-Status, Blur, Gewichtungsfaktoren) angelegt. Im umgekehrten Fall wird bei einer Reduktion der in der Folge nicht mehr benötigte Speicher wieder freigegeben. Die selbe Methodik wird auch bei der Speicherverwaltung in Bezug auf die Konstruktion und Speicherung der konvexen Hülle angewendet.

Daten-Organisation

Das Konzept für die Organisation der Daten richtet sich eher nach dem Prinzip der Objektorientierung, wo es unter anderem eine der grundlegenden Ideen ist, zusammengehörige Daten (wie in diesem Fall etwa von Lautsprechern oder virtuellen Quellen) auch gemeinsam abzuspeichern. Im Gegensatz zur Max/MSP-Implementation, welche die Informationen zu einer virtuellen Quelle wie etwa Koordinaten, Gain, oder Mute-Status getrennt speichert, wird in der Implementation für Pd ein eigens dafür angelegter Datentyp für virtuelle Quellen verwendet, in dem nun alle Informationen, die zu einer Quelle gehören, gemeinsam abgespeichert werden können. Selbiges gilt auch für die Organisation der Lautsprecher-Daten und für die Daten der konvexen Hülle.

Auch der Ansatz zur Speicherung der Gewichtungsfaktoren wurde etwas verändert. Im Unterschied zum Max/MSP-External, wo die Gewichtungsfaktoren aller Lautsprecher bezüglich einer virtuellen Quelle als der jeweiligen Quelle zugehörig aufgefasst werden, wird beim Pd-External jeder Gewichtungsfaktor genau der jeweiligen Quelle-Lautsprecher-Kombination zugeordnet. Deshalb werden die Gewichtungsfaktoren nicht in der Datenstruktur für virtuelle Quellen gespeichert, sondern in einer eigenen Tabelle (die als zweidimensionales Array implementiert ist) abgelegt, wobei die Zeilen die virtuellen Quellen und die Spalten die Lautsprecher darstellen. Damit entspricht jede Zelle genau einer Quelle-Lautsprecher-Kombination.

Visualisierung

Die Implementation für Max/MSP enthält eine Möglichkeit zur Visualisierung von Dominanz-Zonen einzelner Lautsprecher oder Auswirkungen der eingestellten Gewichtungsfaktoren. Die Pd-Implementation unterstützt zwar die zugehörigen Schnittstellen aus Gründen der Kompatibilität, die Funktionalität selbst wurde aber im Rahmen dieser Arbeit nicht implementiert.

Konvexe Hülle

Im Hinblick auf die Erweiterung durch die konvexe Hülle in 3D wurden auch die Algorithmen und Implementationen für die übrigen Dimensionszahlen analog zu jenen für 3D gewählt. Das Selbe gilt auch für die Vorgänge einer etwaigen Projektion von virtuellen Quellen auf die konvexe Hülle. Eine Beschreibung der jeweiligen Algorithmen findet sich in Kapitel 3.

2.1.3 Erweiterungen

Sichern und Laden von Daten

Die Pd-Implementation bietet die Möglichkeit, die zu einem bestimmten Zeitpunkt im Pd-External gespeicherten Daten für Lautsprecher, virtuelle Quellen oder Gewichtungsfaktoren in einer Text-Datei abzuspeichern, und sie zu einem späteren Zeitpunkt oder an einem anderen Ort wieder zu laden.

Speichert man die Daten für Lautsprecher oder virtuelle Quellen, wird für jeden Lautsprecher bzw. jede Quelle eine Zeile mit den zugehörigen Informationen in die angebene Datei geschrieben, beim Speichern der Gewichtungsfaktoren je eine Zeile pro Quelle-Lautsprecher-Kombination. Die jeweilige Syntax der einzelnen Zeilen ist in Tabelle 2.1 angegeben, als Dezimaltrennzeichen wird dabei wie in Pd selbst ein Punkt verwendet.

Daten	Syntax
Lautsprecher	`x y z`
virtuelle Quellen	`x y z; gain; mute; blur`
Gewichtungsfaktoren	`src-id dst-id weight`

Tabelle 2.1: Syntax der Zeilen in den gespeicherten Dateien

Da es sich um Text-Dateien handelt, können diese auch nachträglich verändert oder manuell erstellt werden, um sie danach in das External zu laden. Außerdem besteht die Möglichkeit, Kommentar-Zeilen zu verwenden, um Beschreibungen hinzuzufügen. Als solche gelten dabei Zeilen, die mit `//` oder `#` beginnen.

Selektor-System

Möchte man bestimmte Eigenschaften eines Objektes, zum Beispiel den Gain-Faktor einer virtuellen Quelle, verändern, müssen die ID-Nummer der ausgewählten Quelle und die neuen Daten als Parameter mit dem Befehl zur Änderung übergeben werden (für genauere Informationen zur Verwendung des Externals siehe Abschnitt 2.1.4). Die Änderung einer Eigenschaft für alle Quellen mit nur einem Befehl war (mit Ausnahme der Blur-Eigenschaft, die auch zuvor mit einem eigenen Befehl für alle Quellen gesetzt werden konnte) nicht möglich.

Im Pd-External wird nun für alle Befehle, die Eigenschaften von Lautsprechern und virtuellen Quellen sowie Gewichtungsfaktoren betreffen, ein erweitertes System verwendet, bei dem der Wirkungsbereich eines solchen Befehls durch den als Parameter übergebenen Selektor bestimmt wird. Ein solcher Selektor kann nun einerseits weiterhin eine ID-Nummer sein, andererseits ist aber auch der Selektor `all` implementiert, mit dem der Befehl auf alle gleichartigen Objekte angewendet wird (im obigen Beispiel könnten so die Gain-Faktoren für alle Quellen gleichzeitig gesetzt werden). Die Auswertung des Selektors übernimmt bei allen betroffenen Befehlen eine Unterfunktion, deren Aufgabe es ist, den Selektor auf Gültigkeit zu überprüfen und den Wirkungsbereich festzulegen. Es wäre so auch möglich, diese Unterfunktion um zusätzliche Selektoren zu erweitern, um beispielsweise ID-Bereiche oder Ähnliches zu ermöglichen.

Wie zuvor erwähnt gab es für die Blur-Eigenschaft bereits einen eigenen Befehl, um diese für alle Quellen zu ändern. Dieser wird aus Kompatibilitätsgründen beibehalten, er ist jedoch nicht mehr eigens ausimplementiert, sondern ruft intern nur mehr den Blur-Befehl mit dem Selektor `all` auf.

Konvexe Hülle in 3D

Die Implementation der konvexen Hülle in 3D, die die Konstruktion der Hülle und die etwaige Projektion von virtuellen Quellen auf diese Hülle umfasst, ist mit Abstand die größte und aufwändigste Erweiterung im Zuge der Programmierung dieses Pd-Externals. Deshalb soll an dieser Stelle für genauere Informationen auf Kapitel 3 verwiesen werden, welches sich ausschließlich mit der Implementation der konvexen Hülle beschäftigt.

2.1.4 Verwendung

In diesem Abschnitt soll nun auf den Umgang mit dem External bei der Verwendung in einem Pd-Patch eingegangen werden.

Instanziierung

Bei der Instanziierung des Externals, also beim Erstellen des Objektes in einem Pd-Patch, können bis zu vier optionale Parameter übergeben werden. Es ist dabei möglich, nur den ersten, die ersten beiden, die ersten drei oder alle vier der in Tabelle 2.2 aufgelisteten Argumente in genau der gezeigten Reihenfolge anzugeben, wobei für die hinteren weggelassenen Parameter die angegebenen Standardwerte verwendet werden.

Nr.	Bedeutung	Wertebereich	Standardwert
1	Lautsprecheranzahl	0 - 500	1
2	Quellenanzahl	0 - 250	1
3	Raum-Dimensionen	1 - 3	2
4	Rolloff in dB	beliebig	6

Tabelle 2.2: Übergabeparameter bei der Instanziierung des Externals

Die so definierten Werte werden als Startwerte für die jeweiligen Einstellungen verwendet, diese können aber während der Benutzung des Externals durchaus wieder verändert werden.

Steuerung

Zur Veränderung der Einstellungen des Externals oder der Eigenschaften von Lautsprechern und virtuellen Quellen akzeptiert das Inlet die in Tabelle 2.3 aufgelisteten Messages. In der Tabelle sind außerdem die zu verwendende Syntax (»sel« steht als Kurzform von engl. selector für einen Selektor, vgl. dazu Selektor-System in Abschnitt 2.1.3), die zulässigen Wertebereiche für die Parameter sowie die Initialwerte (diese werden verwendet, solange sie nicht durch eine Message bei der Verwendung überschrieben wurden)

angegeben. Existiert für eine Message aufgrund von Namenskonventionen oder zur Abkürzung ein Alias, so ist dieser ebenfalls in der Tabelle eingetragen. Zusätzlich soll auf einige nicht selbsterklärende Messages noch näher eingegangen werden.

`hull` Mit dieser Message können die Konstruktion der konvexen Hülle und die etwaige Projektion von virtuellen Quellen auf die Hülle aktiviert oder deaktiviert werden.

`update` Standardmäßig werden sofort mit einer Zustandsänderung des Externals die internen Daten neu berechnet und die aktualisierten relativen Amplituden ausgegeben. Dieses Verhalten kann mit dieser Message ein und ausgeschaltet werden, um z.B. ein zwischenzeitliches Updating während der Anpassung von Einstellungen zu vermeiden.

`bang` Sollte eine Neuberechnung der internen Daten des Externals und eine Ausgabe der relativen Amplituden zu einem bestimmten Zeitpunkt erwünscht sein, kann ein solches manuelles Update durch diese Message herbeigeführt werden.

`info` Hiermit kann der gesamte interne Zustand des Externals abgefragt werden. Dabei werden über das Info-Outlet Messages für alle Variablen ausgegeben, unterscheidbar durch einen jeweiligen Deskriptor als ersten Parameter.

`src_mute` Einzelne Quellen können mit dem Parameter `1` stumm geschalten bzw. mit dem Parameter `0` wieder hörbar gemacht werden.

`weight` Will man den Gewichtungsfaktor der durch die Selektoren ausgewählten Quelle-Lautsprecher-Kombination verändern (bzw. bei Verwendung des Selektors `all` auch mehrere), kann das mit dieser Message bewerkstelligt werden (z.B. dritte Quelle, vierter Lautsprecher, Gewichtung 0.9: `weight 3 4 0.9`).

`src_weight` Hiermit können Kombinationen der durch den Selektor bestimmten Quelle mit mehreren Lautsprechern beeinflusst werden. Dabei erhält die Kombination mit dem ersten Lautsprecher (ID-Nummer 1) den ersten übergebenen Gewichtungsfaktor, jene mit dem zweiten Lautsprecher den zweiten, und so weiter. Werden mehr Gewichtungsfaktoren übergeben als Lautsprecher definiert sind, werden die überzähligen Argumente ignoriert. Werden allerdings weniger Parameter übergeben, so werden die fehlenden als Null aufgefasst, wodurch die zugehörigen Lautsprecher für die angegebene Quelle deaktiviert werden.

Einstellung	Syntax	Wertebereich	Initialwert	Alias
Raum-Dimensionen	`dimensions` *value*	1 - 3	2	`dim`
Rolloff in dB	`rolloff` *value*	beliebig	6	—
Lautsprecheranzahl	`num_destinations` *value*	0 - 500	1	`num_dst`
Quellenanzahl	`num_sources` *value*	0 - 250	1	`num_src`
Master Gain	`master_gain` *value*	nicht-negativ	1	—
konvexe Hülle ein/aus	`hull` *value*	1 oder 0	0 (aus)	—
Updating ein/aus	`update` *value*	1 oder 0	1 (ein)	—
manuelles Update	`bang`	—	—	—
Info	`info`	—	—	—
Objekt-Eigenschaft	**Syntax**	**Wertebereich**	**Initialwert**	**Alias**
Lautsprecher-Position	`dst_position` *selector x y z*	beliebig	—	`dst_pos`
Quellen-Position	`src_position` *selector x y z*	beliebig	—	`src_pos`
Gain (Quelle)	`src_gain` *selector value*	nicht-negativ	1	—
Mute ein/aus (Quelle)	`src_mute` *selector value*	1 oder 0	0 (aus)	—
Blur (Quelle)	`blur` *selector value*	nicht-negativ	10^{-6}	`src_blur`
Blur (alle Quellen)	`blurall` *value*	nicht-negativ	10^{-6}	`src_blurall`
Gewichtungsfaktoren	**Syntax**	**Wertebereich**	**Initialwert**	**Alias**
ausgewählte Faktoren	`weight` *src-sel dst-sel value*	nicht-negativ	1	—
gereihte Faktoren	`src_weight` *sel weight1 weight2 ...*	nicht-negativ	1	—
Sichern und Laden	**Syntax**	**Wertebereich**	**Initialwert**	**Alias**
Lautsprecher-Daten	`save_dst` *path*	—	—	—
	`load_dst` *path*	—	—	—
Quellen-Daten	`save_src` *path*	—	—	—
	`load_src` *path*	—	—	—
Gewichtungsfaktoren	`save_weights` *path*	—	—	—
	`load_weights` *path*	—	—	—

Tabelle 2.3: Verfügbare Messages zur Steuerung des Externals

`load_*` bzw. `save_*` Mit diesen Messages werden die jeweiligen Daten aus einer als Parameter übergebenen Datei geladen (womit die jeweiligen internen Daten überschrieben werden) bzw. in die angegebene Datei gespeichert. Unterstützt werden absolute und relative Pfade, wobei letztere als relativ zum Arbeitsverzeichnis von Pd oder zum Speicherort jenes Patches interpretiert werden, welcher das External beinhaltet (bei Abstractions werden auch deren übergeordnete Patches berücksichtigt). Unter Linux und Macintosh wird auch der Pd-Search-Path durchsucht, unter Windows ist dies erst ab einer für Pd 0.43.0 kompilierten Version möglich.

Outlets

Während der Verwendung des Externals dienen die Outlets des Objektes dazu, die berechneten Ergebnisse und andere Informationen an das Umfeld zurückzugeben. In Tabelle 2.4 sind die Outlets, die das External von links nach rechts zur Verfügung stellt, mit der jeweiligen Syntax der ausgegebenen Messages aufgelistet.

Nr.	Funktion	Syntax
1	relative Amplituden	*`src-id dst-id coeff`*
2	Distanzen	*`src-id dist dst-id`*
3	Visualisierung	
4	Info	*`descriptor data...`*

Tabelle 2.4: Outlets des Externals

Am ersten Outlet werden bei Zustandsänderungen des Externals die neu berechneten relativen Amplituden für alle betroffenen Quelle-Lautsprecher-Kombinationen (welche jeweils durch `src-id` und `dst-id` bezeichnet werden) ausgegeben.

Ist die Verwendung der konvexen Hülle aktiviert, so gibt das zweite Outlet bei Zustandsänderungen des Externals für alle betroffenen Quellen (*`src-id`*) jeweils eine Message aus, welche die kürzeste Distanz (*`dist`*) der Quelle zur Hülle und den der Quelle am nächsten liegenden Hüllen-Lautsprecher (*`dst-id`*) beinhaltet. Dabei ist der Wert der Distanz negativ, wenn die definierte Quell-Position innerhalb der konvexen Hülle liegt, andernfalls positiv.

Das dritte Outlet, das für Visualisierungsdaten vorgesehen ist, wurde aus Kompatibilitätsgründen beibehalten. Es wird aber nicht verwendet, da die entsprechende Funktion im Rahmen dieser Arbeit nicht implementiert wurde (vgl. Visualisierung in Abschnitt 2.1.2).

Erhält das External die Message `info`, so werden die Informationen zum internen Status des Externals (Werte für Einstellungen, Eigenschaften, etc.) nacheinander in einzelnen Messages am vierten Outlet ausgegeben. Um welche Daten es sich handelt, wird durch den ***`descriptor`*** angegeben. Bei der Ausgabe der Koordinaten eines Lautsprechers (z.B. $id = 5$, $x = 7$, $y = 1.4$, $z = 2$) würde die Message `dst_position 5 7 1.4 2` lauten.

2.2 Patches

Begleitend zum External wurden auch einige Pd-Patches angelegt. Dazu gehören ein Help-Patch zur Beschreibung des Externals selbst (Abschnitt 2.2.1), sowie eine Signal-Abstraction (ebenfalls inklusive Help-Patch), die eine mögliche Implementation der notwendigen Peripherie für das Sound-Processing anbietet (Abschnitt 2.2.2).

2.2.1 Help-Patch

Bei Objekten für Pd ist es üblich, einen sogenannten Help-Patch zur Verfügung zu stellen, der zur Unterstützung bei der Verwendung dienen soll. Da das External für die Verwendung mit einem Matrix-Objekt konzipiert ist, benutzt der Help-Patch (in Abbildung 2.1 dargestellt) das Objekt `matrix~` der Cyclone-Library für Pd. Als optionales grafisches Steuerelement kommt `room_sim_3d` aus der Library Iemgui zum Einsatz. Abbildung 2.2 zeigt eine zweite Seite, welche die Liste der verfügbaren Messages mit den zugehörigen Beschreibungen analog zu Abschnitt 2.1.4 enthält.

Zur besseren Demonstration der Spatialisation von bewegten virtuellen Quellen wurde ein Trajektorien-Modul (unter Verwendung der Libraries Flatspace, Mapping, Purepd und Zexy) in den Help-Patch integriert. Es bietet die Möglichkeit, eine Quelle auf einer der beiden folgenden Trajektorien automatisch durch den Raum zu bewegen.

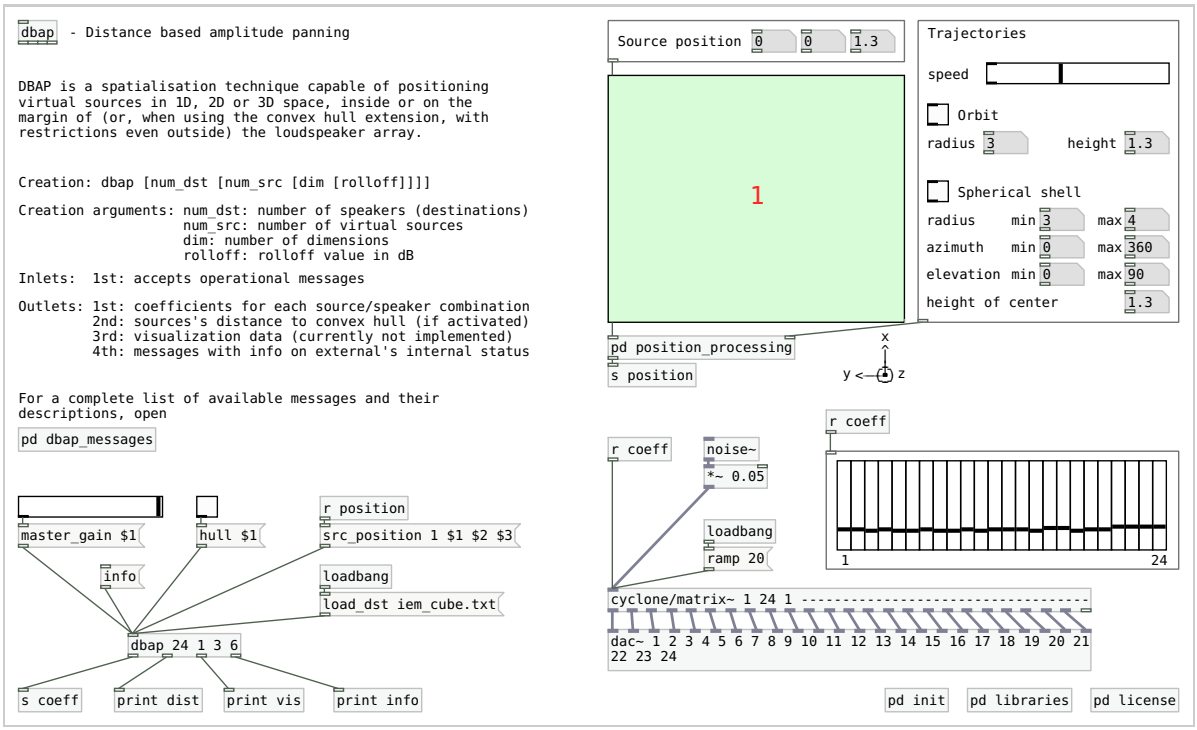


Abbildung 2.1: Help-Patch des DBAP-Externals

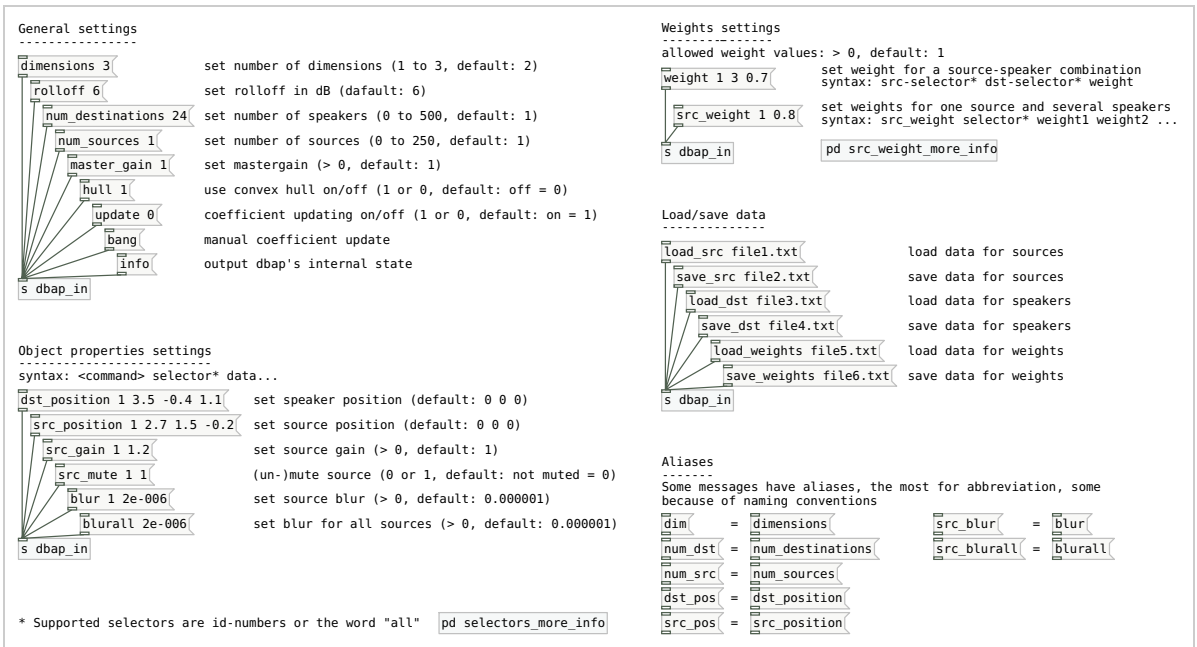


Abbildung 2.2: Beschreibung der verfügbaren Messages für das DBAP-External im Help-Patch

Orbit Die Quelle bewegt sich gegen den Uhrzeigersinn auf einem waagrecht im Raum liegenden Kreis, wobei der Radius und die Entfernung vom Boden (`height`) angepasst werden können.

Kugelschale Innerhalb einer durch minimalen und maximalen Radius sowie die Entfernung des Mittelpunktes vom Boden (`height of center`) konfigurierbaren Kugelschale, die noch zusätzlich auf kleinere Azimuth- und Elevationswinkel eingeschränkt werden kann, bewegt sich die Quelle auf einem zufälligen Weg.

2.2.2 Signal-Abstraction

Zur Verwendung des DBAP-Externals in einer für Standard-Anwendungen geeigneten Umgebung wurde eine Abstraction entwickelt, die als Objekt `dbap~` in einem Pd-Patch geladen werden kann. Ziel dieses Wrappers ist es, ein kompaktes Modul bereitzustellen, in welchem das External bereits mit einer möglichen Variante der für das eigentliche Sound-Processing notwendigen Peripherie verbunden ist. Da das External für die Verwendung mit einem Matrix-Objekt konzipiert ist, basiert die für diese Abstraction verwendete Variante auf der Verschaltung des Externals mit dem Objekt `matrix~` der Cyclone-Library für Pd, zusätzlich wird die Library Iemlib verwendet.

Instanziierung

Die Instanziierung wird prinzipiell analog zu jener des DBAP-Externals vorgenommen (vgl. Instanziierung in Abschnitt 2.1.4). Es gibt allerdings einen kleinen Unterschied: Da die Abstraction die notwendige Anzahl an Signal-Inlets bzw. Signal-Outlets für die angegebene Anzahl an virtuellen Quellen bzw. Lautsprechern zur Verfügung stellt, und diese bei der Instanziierung erstellt werden, gelten diese Anzahlen als Maximalwerte, da im Nachhinein keine Inlets oder Outlets mehr hinzugefügt werden können.

Inlets und Outlets

Der Wrapper bietet ein Inlet für Steuerungs-Messages, die an das DBAP-External weitergeleitet werden, gefolgt von so vielen Signal-Inlets wie bei der Instanziierung definiert

wurden (entweder durch Übergabe des entsprechenden Parameters, oder durch Verwendung des Standardwertes).

Signal-Outlets werden ebenfalls entsprechend der Definition zur Verfügung gestellt. Das letzte Outlet ist mit dem Info-Outlet des Externals verbunden und gibt somit dessen Info-Messages direkt aus.

Help-Patch

Als Unterstützung bei der Verwendung der Signal-Abstraction wurde analog zum External `dbap` auch für die Abstraction `dbap~` ein Help-Patch erstellt, der in Abbildung 2.3 zu sehen ist.

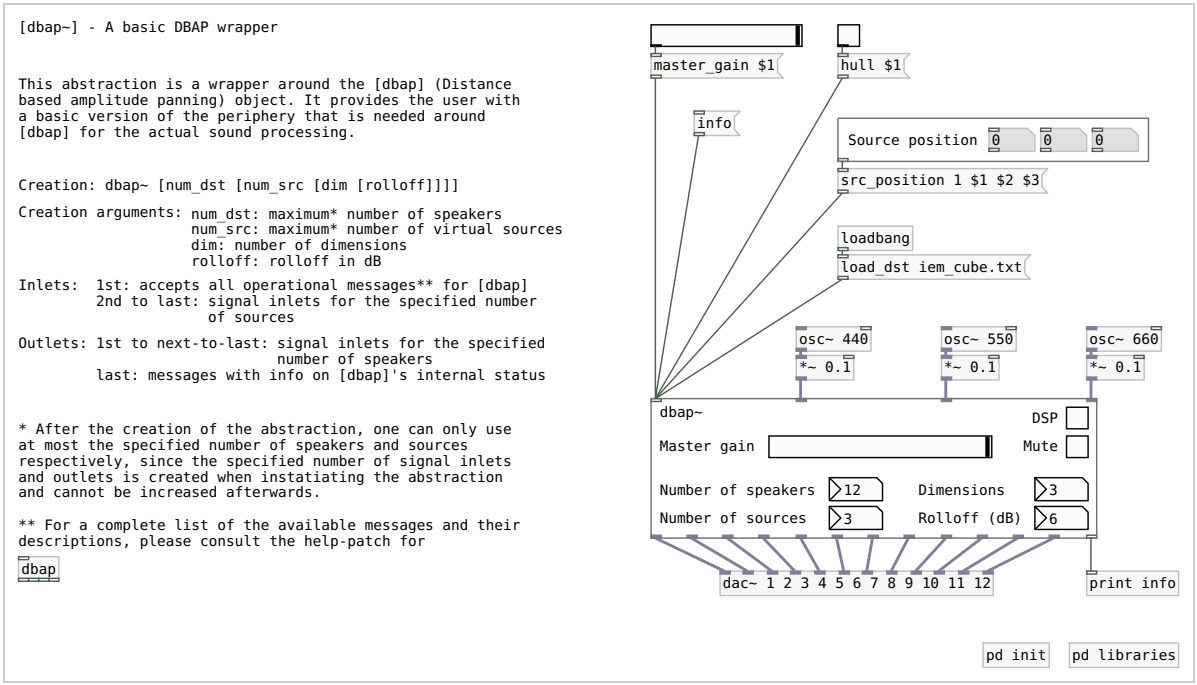


Abbildung 2.3: Help-Patch der DBAP-Abstraction

3 Konvexe Hülle

Dieses Kapitel widmet sich der konvexen Hülle, der größten Erweiterung im Zuge der Implementation des Pd-Externals. Dabei sind die Hüllen-Berechnungen selbst unabhängig von DBAP implementiert, um sie auch in anderen Programmen verwenden zu können.

Abschnitt 3.1 soll einen kurzen Überblick über das Speicherkonzept für die konvexe Hülle geben und in Abschnitt 3.2 werden die Algorithmen für die Konstruktion der Hülle für die verschiedenen Dimensionszahlen erklärt. Abschnitt 3.3 beschäftigt sich anschließend mit der Projektion von virtuellen Quellen auf die konvexe Hülle.

3.1 Speicherkonzept

Zur Speicherung der konvexen Hülle wurde eine eigene Struktur angelegt, in der während der Konstruktion alle relevanten Daten für die Verwendung der Hülle abgelegt werden. Damit sind alle Daten unabhängig von den Algorithmen für die verschiedenen Dimensionszahlen an einem zentralen Ort zusammengefasst.

Die gespeicherten Daten umfassen die Anzahl der Dimensionen der Hülle selbst, die jeweilige Anzahl der Randpunkte (engl. vertices), Kanten (engl. edges) und Flächen (engl. faces) der Hülle, die Randpunkte jeder einzelnen Kante und jeder Fläche, sowie im Fall einer 2D-Hülle den Normalvektor auf die Hüllen-Ebene.

Der benötigte Speicherplatz wird analog zu den anderen Strukturen ebenfalls dynamisch angefordert (vgl. Speicherkonzept in Abschnitt 2.1.2).

3.2 Konstruktion

Die verwendeten Algorithmen zur Konstruktion der konvexen Hülle richten sich bis auf einige Änderungen nach den Beschreibungen der sogenannten inkrementellen Algorithmen in *Computational Geometry in C* [ORo98, Kapitel 1, 3 und 4]. Die nachfolgenden Informationen in diesem Abschnitt sollen nun die zu Grunde liegenden Algorithmen erläutern, für Details zur Implementation sei direkt auf den Quellcode des Pd-Externals verwiesen.

Der Pseudo-Code in Algorithmus 3.1 bietet als Einstieg eine Übersicht über den allgemeinen Teil der Hüllen-Konstruktion, der für alle Dimensionszahlen gleichermaßen notwendig ist.

```
// preparations
clear existing hull
load points array

// preprocessing
sort points array
mark duplicates of previous points

// construct hull
if (coordinate dimensions == 3) run 3D algorithm
else if (coordinate dimensions == 2) run 2D algorithm
else if (coordinate dimensions == 1) run 1D algorithm

// cleanup
delete vertices/edges/faces construction lists
delete points array
```

Algorithmus 3.1: Allgemeiner Teil der Konstruktion einer konvexen Hülle

Zu Beginn werden eventuell bereits existierende Daten einer zuvor konstruierten Hülle gelöscht und eine Kopie der Lautsprecher-Daten (ID-Nummern und Koordinaten) in ein Punkte-Array geladen, um die Originaldaten durch den Konstruktionsprozess nicht zu verändern.

Bei den inkrementellen Algorithmen für 2D und 3D wird dieses Punkte-Array grundsätzlich Eintrag für Eintrag abgearbeitet. Falls die jeweiligen Punkte nicht innerhalb der bis zum jeweiligen Zeitpunkt ermittelten vorübergehenden Hülle liegen, werden sie zu dieser hinzugefügt. So wird die konvexe Hülle nach und nach immer weiter aufgebaut, bis sie nach der Verarbeitung des letzten Punktes vollständig konstruiert ist.

Anders als in der Vorlage wird das Punkte-Array aber zuvor zusätzlich vorverarbeitet. Dabei werden die Punkte nach der x-, dann nach der y- und zuletzt nach der z-Koordinate jeweils aufsteigend sortiert. Damit kann einerseits die Prüfung, ob ein Punkt innerhalb liegt, entfallen, da zu bearbeitende Punkte sicher nicht innerhalb der bisherigen vorübergehenden Hülle liegen. Außerdem werden koinzidente Punkte (Punkte mit exakt gleichen Koordinaten) als Duplikate markiert, indem die ID-Nummer auf -1 gesetzt wird. So besitzt immer nur jeweils der letzte der Punkte mit gleichen Koordinaten eine gültige ID-Nummer, außer die Punkte ganz am Beginn des Arrays sind koinzident, dann behält der allererste seine ID-Nummer. Anhand dieser Markierung können koinzidente Punkte später einfach erkannt und übersprungen werden, um unnötige Rechenschritte zu vermeiden.

Danach wird der zur Anzahl der Raumdimensionen passende Konstruktionsalgorithmus aufgerufen, denn in einem n-dimensionalen Raum kann auch die konvexe Hülle maximal in n Dimensionen ausgedehnt sein. In einem dreidimensionalen Raum entspricht diese maximal ausgedehnte Hülle also der Oberfläche eines Körpers, in einem zweidimensionalen Raum dem Rand einer planaren Fläche, und in einem eindimensionalen Raum der Verbindungslinie zwischen zwei Punkten.

Anhängig von der Anordnung der Lautsprecher kann aber auch eine Degeneration der Hülle auftreten, d.h. die Dimensionsanzahl der Hülle selbst ist mathematisch betrachtet geringer als jene des Raumes, in dem die Lautsprecherkoordinaten definiert wurden. Werden die Lautsprecher beispielsweise in einem dreidimensionalen Raum bewusst oder unbewusst derart positioniert, dass alle Lautsprecher in einer schiefen Ebene oder auf einer Geraden im Raum liegen, so degeneriert die Hülle zu einer Fläche oder zu einer Strecke, was einer zweidimensionalen bzw. eindimensionalen Hülle entspricht. Für den Extremfall, dass alle Lautsprecher koinzident sind, entartet die Hülle zu einem einzigen

Punkt und wird damit nulldimensional. Analoge Überlegungen gelten auch für in zwei- oder eindimensionalen Räumen definierte Lautsprecheranordnungen.

Tritt also einer dieser Fälle ein, wird die Konstruktion der Hülle automatisch vom aufgerufenen auf den zur Anzahl der Hüllendimensionen passenden Algorithmus umgeleitet.

Sobald alle Punkte verarbeitet und die konvexe Hülle damit vollständig aufgebaut ist, wird der dimensionszahlabhängige Teil der Konstruktion wieder verlassen. Anschließend werden die nur temporär zur Konstruktion benötigten Daten wieder gelöscht, was die für den 2D- bzw. 3D-Algorithmus notwendigen Konstruktionslisten (sofern diese nicht aus Effizienzgründen schon im jeweiligen Algorithmus selbst gelöscht wurden) sowie das am Beginn angelegte Punkte-Array betrifft.

Die Art und Weise, wie die Konstruktion der konvexen Hülle im Detail vorgenommen wird, hängt nun von der Dimensionsanzahl der Hülle ab.

3.2.1 Nulldimensionale Hülle

Der in Algorithmus 3.2 dargestellte Konstruktionsablauf kommt nur im Spezialfall zur Anwendung, dass der 1D-Algorithmus eine Degeneration der Hülle feststellt, weil alle Punkte koinzident sind, und deshalb auf diese 0D-Konstruktion umleitet.

```
// construct hull
hull consists of only one point

// save hull data
set hull dimensions to 0
set vertices/edges/faces counters to 1/0/0
save hull vertex
```

Algorithmus 3.2: Konstruktion einer nulldimensionalen konvexen Hülle

Da in diesem Fall die Hülle nur aus einem einzigen Punkt besteht, müssen nur die Dimensionszahl der konvexen Hülle sowie die Zähler für Randpunkte, Kanten und Flächen gesetzt werden, ehe einer der koinzidenten Punkte als nulldimensionale Hülle gespeichert wird.

3.2.2 Eindimensionale Hülle

Auch die Konstruktion einer eindimensionalen konvexen Hülle gestaltet sich noch recht einfach, da in diesem Fall einiges an A-priori-Wissen über die Anordnung der Punkte und die Gestalt der Hülle verwertet werden kann.

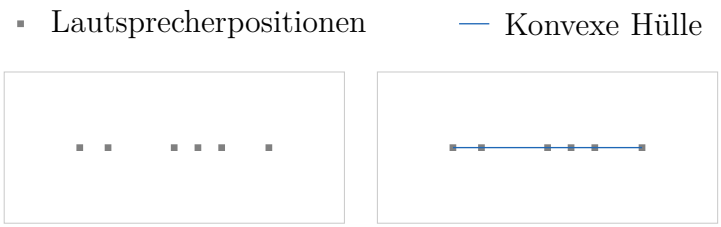
```
// test for degeneration
if (all points coincident) redirect to 0D algorithm

// construct hull
hull consists of first and last point in array

// save hull data
set hull dimensions to 1
set vertices/edges/faces counters to 2/1/0
save hull vertices and edge
```

Algorithmus 3.3: Konstruktion einer eindimensionalen konvexen Hülle

Zuerst wird überprüft, ob die Hülle degeneriert ist, d.h. ob alle Punkte koinzident sind. Ist dies der Fall, wird auf den 0D-Algorithmus umgeleitet, andernfalls hat die Hülle genau eine Dimension und alle Punkte liegen auf einer Gerade (siehe Abbildung 3.1a). Somit müssen die Begrenzungspunkte der konvexen Hülle jene Punkte auf der Gerade sein, die am weitesten voneinander entfernt liegen (Abbildung 3.1b).



a. Initialisierung b. Ergebnis

Abbildung 3.1: Konstruktion einer eindimensionalen konvexen Hülle

Durch die Sortierung des Punkte-Arrays müssen zur Bestimmung dieser Punkte nur der erste und letzte Eintrag des Arrays ausgelesen werden.

Zum Schluss werden die Dimensionsanzahl und die Zähler gesetzt, sowie die Randpunkte in Form ihrer ID-Nummern und die in diesem Fall einzige Kante der Hülle in Form der ID-Nummern ihrer Endpunkte gespeichert.

3.2.3 Zweidimensionale Hülle

Die 2D-Konstruktion erfolgt bis auf einige Änderungen entsprechend dem in *Computational Geometry in C* [ORo98, Kapitel 3] beschriebenen inkrementellen Algorithmus und ist in Algorithmus 3.4 angeführt.

```
// test for degeneration
if (all points coincident) redirect to 0D algorithm
if (all points collinear) redirect to 1D algorithm

// initialise hull
find 3 non-collinear points (NCPs)
add 1st and 2nd NCP to vertices construction list
mark 3rd NCP as first processing point
calculate hull plane's normal vector
initialise vertices counter (VC) to 2

// construct hull
for each (unprocessed point in array)
{
  if (current point is marked as duplicate)
    continue with next point
  find points of tangency (POTs)
  delete points between POTs
  insert current point between POTs
}

// save hull data
set hull dimensions to 2
set edges/faces counters to VC/1
save hull vertices and edges
```

Algorithmus 3.4: Konstruktion einer zweidimensionalen konvexen Hülle

Zu Beginn wird überprüft, ob die Hülle wirklich zweidimensional oder eventuell degeneriert ist. Für den Fall, dass alle Punkte koinzident sind, wird wieder auf den 0D-Algorithmus umgeleitet, sollten alle Punkte kollinear sein, also auf einer Gerade liegen, so kommt stattdessen der 1D-Algorithmus zum Einsatz.

Ist die Hülle wirklich zweidimensional, so wird die Initialisierung der Konstruktion vorgenommen. Dazu wird das Punkte-Array von vorne beginnend nach drei zueinander nicht-kollinearen Punkten (in der Folge mit NCP für engl. non-collinear point abgekürzt) durchsucht. Um festzustellen, ob drei Punkte kollinear sind, wird das Kreuzprodukt von zwei Vektoren berechnet, deren Endpunkte durch jeweils zwei der drei zu testenden Punkte definiert sind. Ergibt die Berechnung genau den Null-Vektor (alle Einträge gleich Null), so liegen die Punkte genau auf einer Geraden, andernfalls sind sie nicht kollinear und die Suche kann beendet werden.

Die ersten beiden NCPs werden anschließend in einer Punkte-Konstruktionsliste (kurz PKL) gespeichert, welche als doppelt verkettete Liste implementiert ist und die Randpunkte der jeweils während des aktuellen Verarbeitungsschrittes gültigen vorübergehenden Hülle enthält (siehe Abbildung 3.1a). Durch die Sortierung des Punkte-Arrays sind etwaige Punkte, die im Array zwischen den NCPs liegen, entweder koinzident mit einem NCP oder befinden sich auf der Verbindungslinie der ersten beiden NCPs. In beiden Fällen tragen sie nicht zur Hülle bei und können deshalb für die weitere Verarbeitung unbeachtet bleiben.

Im Unterschied zur Beschreibung in der Vorlage wird zusätzlich der Normalvektor der Hüllen-Ebene über das Kreuzprodukt von zwei nicht-parallelen Vektoren in der Ebene berechnet

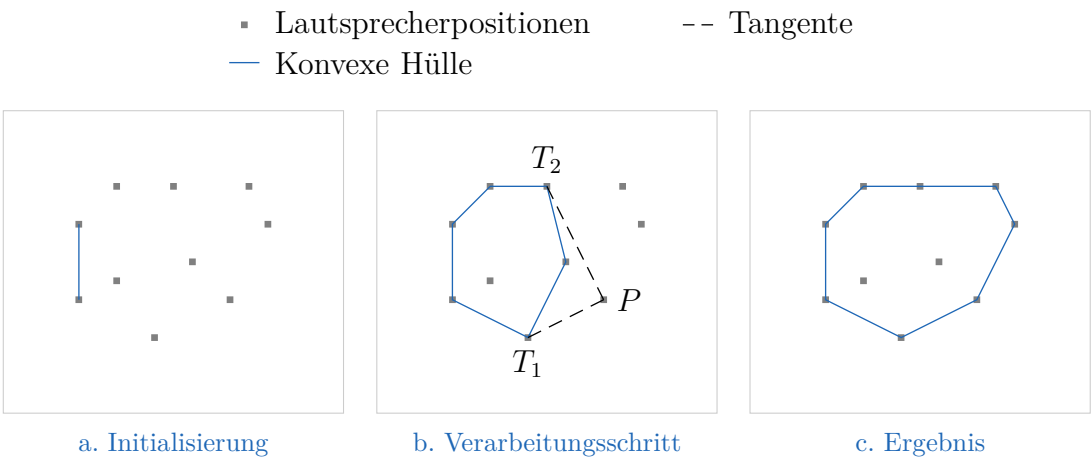
$$\vec{n} = \overrightarrow{AB} \times \overrightarrow{AC} \tag{3.1}$$

Dieser Normalvektor wird im weiteren Verlauf der Hüllen-Konstruktion für die leicht veränderte Berechnung eines vorzeichenbehafteten Flächeninhaltes benötigt.

Außerdem wird vor der Verarbeitung der restlichen Punkte noch der Zähler für Randpunkte auf die Anzahl der sich momentan in der PKL befindenden Punkte initialisiert.

Damit kann das eigentliche Processing zur Konstruktion der konvexen Hülle beginnen, welches ebenfalls ein wenig von der Vorlage abweicht. Dort werden nämlich zu Beginn alle Punkte in eine Kandidaten-Liste geladen, danach alle noch nicht als verarbeitet markierte Punkte überprüft und gegebenenfalls die PKL aktualisiert. Da allerdings in der Pd-Implementierung das Punkte-Array Vorteile für die Algorithmen aller Dimensionszahlen bietet, wird es hier weiterhin anstatt der Kandidaten-Liste verwendet und nun beginnend beim vorgemerkten dritten NCP bis zum letzten Eintrag abgearbeitet.

Wie schon im 1D-Algorithmus werden auch hier Duplikate von Punkten (markiert durch eine ID-Nummer von -1) übersprungen, da sie ohnehin nur den Punkt mit gleichen Koordinaten in der vorübergehenden Hülle ersetzen und damit unnötigen Mehraufwand produzieren würden. Ist der Punkt kein Duplikat, werden jeweils eine untere und eine obere Tangente an die bisherige vorübergehende Hülle gesucht, die durch den zu bearbeitenden Punkt P gehen (siehe Abbildung 3.2b).



a. Initialisierung b. Verarbeitungsschritt c. Ergebnis

Abbildung 3.2: Konstruktion einer zweidimensionalen konvexen Hülle

Diese Tangenten berühren die vorübergehende Hülle an genau zwei Randpunkten, die unterer und oberer Tangentenpunkt (T_1 bzw. T_2) genannt werden. Untersucht man nun die Randpunkte in gegen den Uhrzeigersinn verlaufender Reihenfolge, so können die Tangentenpunkte aufgrund der folgenden Eigenschaft ermittelt werden: Denkt man sich eine gerichtete Trägergerade der Strecke vom Punkt vor dem Tangentenpunkt zum Tangentenpunkt selbst, und eine zweite gerichtete Trägergerade der Strecke vom Tangenten-

punkt zum Punkt danach, so liegt P bezüglich der ersten Gerade auf der einen Seite und bezüglich der zweiten Gerade auf der anderen. Bei der Betrachtung der Trägergeraden mit dem geringeren Abstand zu P kann dieser auch direkt auf der Geraden liegen.

Um die Lage von P bezüglich der Trägergeraden zu bestimmen, wird das Vorzeichen des vorzeichenbehafteten Flächeninhalts jenes Dreiecks herangezogen, welches durch die Endpunkte A und B der Strecke und durch P definiert wird. Liegt P links der gerichteten Trägergeraden durch A und B, was gleichbedeutend damit ist, dass A, B und P gegen den Uhrzeigersinn angeordnet sind, so trägt der Flächeninhalt ein positives Vorzeichen. Für den Fall, dass P rechts der Trägergeraden liegt, sind die Punkte im Uhrzeigersinn angeordnet und Vorzeichen ist negativ. Liegt P genau auf der Trägergeraden, so wird der Flächeninhalt Null.

Zur Ermittlung des Vorzeichens des Flächeninhalts wird aber wie schon zuvor erwähnt nicht die in der Vorlage vorgeschlagene Methode verwendet, da diese nur für den zweidimensionalen Raum gedacht ist. In der Pd-Implementierung ist es jedoch notwendig, auch schiefe Ebenen im dreidimensionalen Raum zu unterstützen, falls der 3D-Algorithmus aufgrund einer Degeneration der Hülle auf den 2D-Algorithmus umgeleitet wird. Darum wurde hier eine Variante auf Basis von *Real-Time Collision Detection* [Eri05, Abschnitt 3.4] gewählt, welche für die Berechnung einer dem vorzeichenbehafteten Flächeninhalt proportionalen Größe $\tilde{A}$ (es ist ja nur das Vorzeichen von Interesse) den bei der Initialisierung berechneten Normalvektor $\vec{n}$ der Hüllen-Ebene benötigt.

$$\tilde{A} = \vec{n} \cdot (\overrightarrow{AB} \times \overrightarrow{AC}) \tag{3.2}$$

Mit Hilfe dieses vorzeichenbehafteten Flächeninhalts werden nun solange die Punkte in der PKL untersucht, bis die beiden Tangentenpunkte T_1 und T_2 ermittelt sind. Auch an dieser Stelle wurde wieder eine kleine Änderung gegenüber der Vorlage vorgenommen: Um möglichst wenige Punkte bei der Suche durchlaufen zu müssen, wird wieder die Sortierung des Punkte-Arrays ausgenützt, aufgrund derer sich nachfolgend hinzugefügte Punkte meist in der Nähe des zuvor verarbeiteten Punktes befinden. Darum beginnt die Suche nach den Tangentenpunkten beim zuletzt zur Hülle hinzugefügten Punkt, und läuft für T_1 im Uhrzeigersinn und für T_2 gegen den Uhrzeigersinn.

Sind die beiden Tangentenpunkte T_1 und T_2 gefunden, werden alle (gegen den Uhrzeigersinn gesehen) dazwischenliegenden Punkte in der PKL gelöscht, denn diese Punkte liegen nun unter Berücksichtigung von P innerhalb der neuen vorübergehenden Hülle. Zum Abschluss der Verarbeitung des aktuellen Punktes wird dieser zwischen T_1 und T_2 in die PKL eingefügt damit die Strecken $\overline{T_1P}$ und $\overline{PT_2}$ den gelöschten Teil der vorübergehenden Hülle ersetzen, der Randpunkte-Zähler wird aktualisiert und es kann mit dem nächsten Punkt fortgesetzt werden.

Nachdem auch der letzte Punkt des Punkte-Arrays abgearbeitet wurde, ist die Hülle vollständig und kann abgespeichert werden. Dabei werden die Dimensionsanzahl der Hülle gesetzt, die Anzahl der Randpunkte und die Anzahl der Kanten (diese ist im 2D-Fall gleich jener der Randpunkte, darum muss kein gesonderter Kanten-Zähler geführt werden) gespeichert sowie die Randpunkte und Kanten der Hülle analog zum 1D-Algorithmus in der Hüllen-Struktur abgelegt.

3.2.4 Dreidimensionale Hülle

Der 3D-Algorithmus richtet sich zum größten Teil nach der Vorlage des inkrementellen Algorithmus für 3D-Hüllen in *Computational Geometry in C* [ORo98, Kapitel 4] und wurde zur Übersicht in Algorithmus 3.5 zusammengefasst.

Wie auch schon in den zuvor beschriebenen Algorithmen wird zu Anfang auf eine Degeneration der Hülle überprüft. Sind alle Punkte koinzident, wird auf den 0D-Algorithmus umgeschaltet, sind sie alle kollinear kommt die 1D-Konstruktion zum Einsatz, und liegen alle in einer Ebene so wird der 2D-Algorithmus angewendet.

Ist die Hülle nicht degeneriert, wird die Initialisierung der 3D-Konstruktion durchgeführt. Dabei werden wie in der 2D-Variante wieder drei nicht-kollineare Punkte (NCPs) ermittelt und zur Punkte-Konstruktionsliste (PKL) hinzugefügt, im Array dazwischen liegende Punkte können wiederum vernachlässigt werden. Außerdem werden die Kanten zwischen jeweils zwei der NCPs in der Kanten-Konstruktionsliste (KKL), die analog zur PKL aufgebaut ist, abgelegt.

```
// test for degeneration
if (all points coincident) redirect to 0D algorithm
if (all points collinear) redirect to 1D algorithm
if (all points coplanar) redirect to 2D algorithm

// initialise hull
find 3 non-collinear points (NCPs)
add 1st, 3rd, 2nd NCP to vertices construction list
create the 3 edges in edges construction list
find 4th non-coplanar point
move 4th point right after 3rd NCP in points array
mark 4th point as first processing point
create twin faces in faces construction list
initialise vertices/edges/faces counters to 3/3/2

// construct hull
for each (unprocessed point in array)
{
  if (current point P is marked as duplicate)
    continue with next point
  find faces visible from P
  mark edges interior to visible region
  find border edges of visible region
  create triangle faces between border edges and P

  // cleanup construction lists
  delete visible faces and interior edges
  mark vertices of all other edges as part of hull
  delete all vertices not marked as part of hull
  remove marks from all vertices
}

// save hull data
set hull dimensions to 3
save hull vertices and faces
```

Algorithmus 3.5: Konstruktion einer dreidimensionalen konvexen Hülle

Im nächsten Schritt wird ein vierter Punkt gesucht, der nicht koplanar zu den NCPs ist, also nicht in der gleichen Ebene liegt. Zur Überprüfung der Koplanarität wird das Volumen des durch die drei NCPs und den vierten Punkt definierten Tetraeders[1] bestimmt, das im koplanaren Fall verschwindet und sonst einen Wert verschieden von Null liefert (die Berechnung wird etwas später im Zuge der Sichtbarkeit von Flächen erläutert). Ist ein nicht-koplanarer vierter Punkt gefunden, wird dieser an die Array-Position direkt nach dem dritten NCP verschoben. Dies ist notwendig, um etwaige dazwischen liegende Punkte bei der anschließenden Verarbeitung nicht zu übergehen, da diese die Hülle noch beeinflussen könnten. Zusätzlich wird dieser vierte Punkt als erster zu verarbeitender Punkt vorgemerkt.

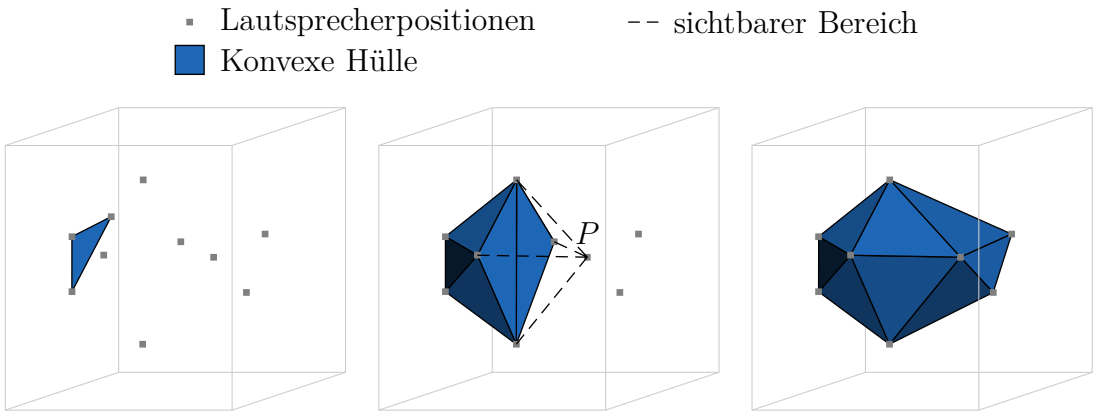
Weiters muss die Flächen-Konstruktionsliste (FKL) initialisiert werden, welche die von jeweils drei Randpunkten begrenzten Oberflächenteile der aktuellen vorübergehenden Hülle beinhalten soll. Dazu werden zwei sogenannte Zwillingsflächen erstellt, welche beide von den drei NCPs begrenzt werden und somit deckungsgleich sind (Abbildung 3.3a). Aufgrund der Tatsache, dass bei einer Fläche die Punkte im Uhrzeigersinn, bei der anderen aber gegen den Uhrzeigersinn angeordnet werden, sind die Richtungsvektoren der Flächen entgegengesetzt gerichtet und zeigen somit beide nach außen.

Um die Initialisierung abzuschließen, müssen noch die Zähler für Randpunkte, Kanten und Flächen auf die korrekten Werte gesetzt werden, dann kann die Verarbeitung der restlichen Punkte beginnen.

Auch bei der 3D-Konstruktion wird nun ein Punkt nach dem anderen abgearbeitet, wobei Duplikate wiederum übersprungen werden. Im nächsten Schritt wird für jede Fläche in der FKL überprüft, ob sie unter Berücksichtigung des gerade zu bearbeitenden Punktes P auch Teil der neuen vorübergehenden Hülle sein wird. Dies geschieht unter Benutzung der folgenden Eigenschaft: Stellt man sich vor, man würde von P aus in Richtung Hülle schauen, so werden jene Flächen auf der P zugewandten Seite der Hülle von P aus sichtbar sein (engl. visible face), jene auf der abgewandten Seite werden durch die sichtbaren Flächen abgeschirmt und sind deshalb von P aus nicht sichtbar (siehe Abbildung 3.3b).

[1] Ein Tetraeder ist eine Pyramide mit einem Dreieck als Grundfläche.

Das Ergebnis der konvexen Hülle in der Pd-Implementation soll eine triangulierte[2] Oberfläche mit möglichst wenigen Teilflächen darstellen. Um dies zu erreichen, werden im Unterschied zur Vorlage auch Flächen als sichtbar gewertet, die genau parallel zur Sicht-Ebene liegen. Damit kann ausgeschlossen werden, dass mehrere koplanare Teilflächen nebeneinander entstehen.



a. Initialisierung b. Verarbeitungsschritt c. Ergebnis

Abbildung 3.3: Konstruktion einer dreidimensionalen konvexen Hülle

Zur mathematischen Bestimmung der Sichtbarkeit einer Fläche wird das vorzeichenbehaftete Volumen (bzw. eine proportionale Größe $\tilde{V}$) jenes Tetraeders berechnet, der von der zu untersuchenden Dreiecksfläche (definiert durch die Eckpunkte A, B, und C) und dem Punkt P aufgespannt wird.

$$\tilde{V} = \overrightarrow{PA} \cdot (\overrightarrow{PB} \times \overrightarrow{PC}) \tag{3.3}$$

Unter Einhaltung der Bedingung, dass A, B, und C von außerhalb der Hülle gesehen immer gegen den Uhrzeigersinn angeordnet sind, steht ein negatives Volumen oder ein Volumen von Null für die Sichtbarkeit der Fläche, ein positives Volumen bedeutet, dass die Fläche von P aus nicht sichtbar ist.

Dies ist in gewisser Weise analog zum 2D-Algorithmus, wo die vorzeichenbehaftete Fläche eines Dreiecks verwendet wurde, um die Lage eines dritten Punktes bezüglich der

[2] Die Zerlegung einer Fläche in dreieckige Teilflächen wird Triangualtion genannt.

gerichteten Trägergeraden von zwei anderen zu bestimmen. Genau genommen hätte man dort auch die Sichtbarkeit der Kanten (ähnlich wie hier jene der Flächen) überprüfen können, um zur neuen vorübergehenden Hülle zu gelangen. Dies hätte aber die zusätzliche Führung einer KKL benötigt, und wurde deshalb über die Tangentenpunkte gelöst, welche direkt die Grenzen des sichtbaren Bereichs liefern.

Sind alle sichtbaren Flächen gefunden und markiert, können nun die Kanten untersucht werden, um die Grenze des sichtbaren Bereichs der 3D-Hülle zu ermitteln. Dazu werden alle Kanten durchlaufen, und jene für die spätere Löschung markiert, die zwei sichtbare Flächen trennen und damit komplett innerhalb des sichtbaren Bereichs liegen (engl. interior edges). Alle Kanten, die hingegen zwischen einer sichtbaren und einer nicht sichtbaren Fläche liegen, befinden sich am Rand des sichtbaren Bereichs und werden deshalb als Rand-Kanten (engl. border edges) bezeichnet.

Von jeder dieser Rand-Kanten als Basis wird nun eine neue Dreiecksfläche mit P als Spitze erzeugt, die als Teil der neuen vorübergehenden Hülle zur FKL hinzugefügt wird. Dabei entsteht eine Art Pyramide bzw. Kegel aus den einzelnen Flächen, der auf die alte vorübergehende Hülle aufgesetzt wird (siehe Abbildung 3.3b).

Dabei ist auf einige Dinge zu achten: Mit jeder neuen Fläche müssen auch die entsprechenden neuen Kanten der Fläche erzeugt, in die KKL eingefügt und mit den angrenzenden Flächen verlinkt werden. Es darf aber für zwei aneinander grenzende Flächen nur eine Kante erzeugt werden. Außerdem muss sichergestellt werden, dass die Eckpunkte jeder Einzelfläche von außen gesehen gegen den Uhrzeigersinn angeordnet sind, um konsistent zur restlichen Hülle zu sein. Dies wird erreicht, indem die Orientierung jener sichtbaren Fläche kopiert wird, welche die gleiche Rand-Kante wie die neue Fläche beinhaltet.

Danach werden die Konstruktionslisten für die Untersuchung des nächsten neuen Punktes vorbereitet, indem nun die zuvor markierten sichtbaren Flächen und die innerhalb des sichtbaren Bereichs liegenden Kanten gelöscht werden. Die Endpunkte der restlichen Kanten werden im Zuge dessen als auf der Hülle liegend markiert, alle anderen Punkte aus der PKL gelöscht, und danach die Markierungen wieder entfernt. Anschließend kann mit dem nächsten Punkt fortgesetzt werden.

Ist auch der letzte Punkt des Arrays verarbeitet, so ist die Hülle damit vollständig konstruiert. Analog zu den anderen Algorithmen werden nun die relevanten Daten der Hülle gespeichert. Dies umfasst das Setzen der Dimensionsanzahl der Hülle und das Ablegen der Randpunkte und Flächen in der Hüllen-Struktur.

3.3 Projektion

Die bei der Konstruktion der konvexen Hülle ermittelten und gespeicherten Daten können nun während der Benutzung des DBAP-Externals herangezogen werden, um virtuelle Quellen auf die Hülle zu projizieren. Dabei soll der Projektionspunkt jener Punkt auf der Hülle sein, der den geringsten Abstand zum Quellpunkt aufweist. Die dazu verwendeten Algorithmen, mit denen die Projektionen abhängig von der Dimensionsanzahl der Hülle durchgeführt werden, basieren auf den Ausführungen in *Real-Time Collision Detection* [Eri05, Kapitel 3 und 5].

Ähnlich wie bei der Konstruktion der Hülle übernimmt auch hier eine übergeordnete Funktion (Algorithmus 3.6) den Aufruf der Unterfunktionen und jene Aufgaben, die für alle Dimensionszahlen gleichermaßen notwendig sind.

```
// calculate projection P of source point S onto hull,
// calculate signed distance from S to P,
// and get index of hull point CP closest to P
if (hull dimensions == 3) run 3D algorithm
else if (hull dimensions == 2) run 2D algorithm
else if (hull dimensions == 1) run 1D algorithm
else if (hull dimensions == 0) run 0D algorithm

// return results
if (distance positive) return P, distance and CP's index
else return S, distance and CP's index
```

Algorithmus 3.6: Allgemeiner Teil der Projektion auf eine konvexe Hülle

Zum einen wird auf den jeweils zur Dimensionsanzahl der Hülle passenden Algorithmus zur Berechnung des Projektionspunktes und der Ausgabewerte verzweigt, wobei diese

Unterfunktionen sowohl die Projektion auf die Hülle als auch einige zusätzlich berechnete Daten zurückgeben.

Zum anderen übernimmt dann die übergeordnete Funktion die Rückgabe dieser Zusatzinformationen an die aufrufende DBAP-Funktion. Dazu gehören der Abstand zwischen Definitions- und Projektionspunkt der virtuellen Quelle inklusive dessen Vorzeichen, welches für außerhalb der Hülle liegende Definitionspunkte positiv und für innerhalb liegende negativ definiert ist. Aufgrund dieses Vorzeichens kann dann die aufrufende DBAP-Funktion entscheiden, ob die Quellposition innerhalb oder außerhalb der Hülle liegt und die Berechnungen anschließend mit den Definitions- oder den Projektionskoordinaten durchgeführt werden sollen. Außerdem wird die ID-Nummer jenes Lautsprechers auf der Hüllenoberfläche zurückgegeben, der den geringsten Abstand zum Projektionspunkt aufweist.

Die Vorgangsweisen bei der Projektion und der Berechnung der Ausgabewerte unterscheiden sich nun je nach Dimensionsanzahl der konvexen Hülle.

3.3.1 Nulldimensionale Hülle

Obwohl eine nulldimensionale konvexe Hülle in der Praxis äußerst selten auftreten wird, deckt die Implementierung auch diesen in Algorithmus 3.7 gezeigten Fall ab.

```
// calculate projection P of source point S
P is equal to the single hull vertex

// determine signed distance
calculate distance between S and P

// find hull point closest to P
closest point CP is the single hull vertex

// return results
return P, positive distance and CP's index
```

Algorithmus 3.7: Projektion eines Punktes auf eine nulldimensionale konvexe Hülle

Da hier die Hülle nur aus einem Punkt besteht, ist auch die Projektion auf diese Hülle immer identisch mit dem Hüllenpunkt selbst. Der Abstand des ursprünglichen Quellpunktes S zum Projektionspunkt P wird (wie auch in allen folgenden Algorithmen) über die Länge des Vektors von S nach P analog zu Gleichung 1.1 berechnet.

Da es sich um eine nulldimensionale konvexe Hülle handelt, kann der Quellpunkt nur entweder außerhalb oder genau auf der Hülle liegen. Die Überprüfung der Lage der Quelle kann deshalb entfallen und für die Distanz kann standardmäßig ein positives Vorzeichen gesetzt werden.

Da auch der am nächsten liegende Lautsprecherpunkt dem einzigen Hüllenpunkt entspricht, ist dieser ebenfalls bekannt und es muss abschließend nur noch der Projektionspunkt zurückgegeben werden.

3.3.2 Eindimensionale Hülle

Ist die Hülle eindimensional, hat sie die Form einer Strecke, die von zwei Endpunkten begrenzt wird. Ein Überblick über die Vorgangsweise ist in Algorithmus 3.8 zusammengefasst.

```
// calculate projection P of source point S
calculate projection onto edge

// determine signed distance
calculate distance between S and P

// find hull point closest to P
closest point CP is one of edge's two vertices

// return results
return P, positive distance and CP's index
```

Algorithmus 3.8: Projektion eines Punktes auf eine eindimensionale konvexe Hülle

Definitionsgemäß soll jener Punkt auf der Strecke das Ergebnis der Projektion sein, der den geringsten Abstand zur Quellposition aufweist. Um diesen Punkt zu ermitteln, wird

die Quellposition zuerst orthogonal auf die Trägergeraden der Strecke zu projiziert, und dann sichergestellt, dass der Projektionspunkt nicht außerhalb der Endpunkte liegt.

Die Projektion $\tilde{P}$ des Quellpunktes S auf die Trägergerade der Begrenzungspunkte A und B (siehe Abbildung 3.4) wird mathematisch mit Hilfe der Vektorrechnung durchgeführt.

$$\vec{\tilde{P}} = \vec{A} + s \cdot \overrightarrow{AB} \tag{3.4}$$

Der Parameter s kann dabei mittels zweier Skalarprodukte berechnet werden.

$$s = \frac{\overrightarrow{AS} \cdot \overrightarrow{AB}}{\overrightarrow{AB} \cdot \overrightarrow{AB}} \tag{3.5}$$

Anschließend muss noch überprüft werden, ob der berechnete Punkt $\tilde{P}$ innerhalb der Strecke $\overline{AB}$ liegt, wozu der berechnete Parameter s herangezogen werden kann. Wenn $0 \leq s \leq 1$ gilt, liegt $\tilde{P}$ innerhalb der Strecke und kann als Ergebnis P der Projektion auf die Strecke verwendet werden. Andernfalls liegt $\tilde{P}$ außerhalb, wobei für $s \leq 0$ der Punkt A bzw. für $s \geq 1$ der Punkt B als Ergebnis der Projektion gilt.

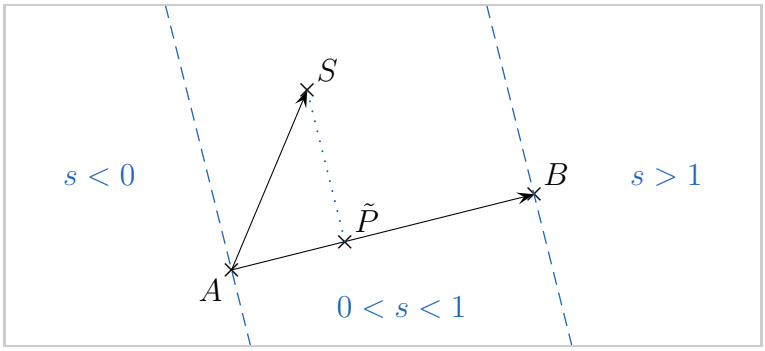


Abbildung 3.4: Lage eines Projektionspunktes bezüglich einer Strecke

Ähnlich wie bei der nulldimensionalen Hülle, kann der Quellpunkt auch hier nur entweder außerhalb oder genau auf der konvexen Hülle liegen. Das Vorzeichen der Distanz zwischen S und P kann also immer als positiv angenommen werden.

Die Suche nach jenem Lautsprecherpunkt auf der Hüllenoberfläche, der P am nächsten ist, gestaltet sich bei einer 1D-Hülle ebenfalls recht einfach. Da die Hülle nur durch

zwei Randpunkte begrenzt wird, muss lediglich festgestellt werden, welcher der beiden Randpunkte den geringeren Abstand zum Projektionspunkt P aufweist.

3.3.3 Zweidimensionale Hülle

Der Projektionsalgorithmus für eine 2D-Hülle, der in Algorithmus 3.9 zu sehen ist, unterscheidet sich im Wesentlichen nur in zwei Punkten von der 1D-Variante.

Handelt es sich wirklich um eine reguläre zweidimensionale Hülle in einem zweidimensionalen Raum, so setzt sich die Hülle im Gegensatz zu jener in 1D aus mehreren Kanten zusammen. Deshalb wird für jede Kante ein Projektionspunkt P' inklusive Distanz berechnet, und jene Daten (Projektionspunkt, Distanz und zugehörige Kante) gespeichert, die zur geringsten Distanz gehören. Bevor diese Daten zurückgegeben werden können, wird noch das Vorzeichen der Distanz anhand der Lage von S bestimmt (also negativ gesetzt, wenn S innerhalb der Hülle liegt, sonst positiv), und der dem Projektionspunkt am nächsten liegende Hüllenpunkt ermittelt (einer der Endpunkte der gespeicherten Kante).

Wenn die zweidimensionale Hülle aber aufgrund einer Degeneration im dreidimensionalen Raum entstanden ist, so ist aus Konsistenzgründen zur 3D-Variante auch die von den Kanten eingeschlossene Fläche Teil der Hülle. Dies hat den Nebeneffekt, dass sich der Quellpunkt S nun ahnlich wie in den vorherigen Algorithmen immer außerhalb oder genau auf der Hülle befindet. Das Vorzeichen der Distanz von S zum Projektionspunkt auf der Hülle kann deshalb in diesem Fall immer positiv definiert werden.

Die Projektion auf diese Hüllfläche soll auch hier definitionsgemäß jenen Punkt auf der Hülle liefern, der den geringsten Abstand zum Quellpunkt S aufweist. Zur Berechnung dieser Projektion wird S zunächst orthogonal auf die Hüllen-Ebene projiziert. Dazu werden ein Punkt A in der Hüllen-Ebene und deren Normalvektor $\vec{n}$ benötigt, der analog zu Gleichung 3.1 berechnet werden kann. Bei der Projektion wird der Quellpunkt S nun soweit in Richtung der Ebene verschoben, bis er auf ihr zu liegen kommt, was mathematisch mit folgender Gleichung angesetzt werden kann.

$$\vec{S'} = \vec{S} - s \cdot \vec{n} \tag{3.6}$$

```
// initialisation
initialise distance d_min with highest possible value

// project source point S onto hull
for each (hull edge)
{
  calculate projection P' of S onto edge
  calculate distance between S and P'
  if (distance < d_min)
  {
    save edge as edge_min
    save distance as d_min
    save P' as projection P
  }
}

// find hull point closest to P
closest point CP is one of edge_min's two vertices

// determine correct projection and return results
if (coordinate dimensions == 3)
{
  calculate projection S' of S onto hull plane
  calculate distance d_plane between S and S'

  if (S' inside hull) return S', d_plane and CP's index
  else return P, d_min and CP's index
}
else // coordinate dimensions == 2
{
  set distance sign (S inside/outside hull)
  return P, signed d_min and CP's index
}
```

Algorithmus 3.9: Projektion eines Punktes auf eine zweidimensionale konvexe Hülle

Dabei wird der Parameter s analog zu Gleichung 3.5 berechnet.

$$s = \frac{\overrightarrow{AS} \cdot \vec{n}}{\vec{n} \cdot \vec{n}} \tag{3.7}$$

Anschließend wird festgestellt, ob S' innerhalb der Hüllfläche liegt. Ist dies der Fall, so gilt S' als endgültiger Projektionspunkt und der Abstand zwischen S und S' als endgültige Distanz. Liegt der Punkt S' außerhalb der Hüllfläche, so muss er zusätzlich auf die nächstliegende Kante projiziert werden. Das Ergebnis dieses Vorganges entspricht aber dem am Beginn dieser Funktion für den Fall einer regulären zweidimensionalen Hülle berechneten Punkt P, weshalb dieser und dessen Abstand zum ursprünglichen Quellpunkt S als Endergebnisse verwendet werden können.

Damit muss nur noch der dem endgültigen Projektionspunkt am nächsten liegende Hüllenpunkt gefunden werden. Dieser ist unabhängig von der Lage der Projektion S' bezüglich der Hüllfläche, sowie identisch mit jenem Hüllenpunkt, der für den Fall einer regulären zweidimensionalen Hülle ermittelt wurde und kann damit ebenfalls von dort übernommen werden.

3.3.4 Dreidimensionale Hülle

Der 3D-Projektionsvorgang (Algorithmus 3.10) unterscheidet sich konzeptionell nur wenig von jenem für eine 2D-Hülle. Außerdem muss der Fall einer degenerierten Hülle nicht überprüft werden, da sich der Quellpunkt räumlich nicht außerhalb der drei Hüllen-Dimensionen befinden kann.

Da sich die Oberfläche einer dreidimensionalen konvexen Hülle aus deren Teilflächen zusammensetzt, werden im 3D-Algorithmus verglichen mit der 2D-Variante statt den Kanten alle Flächen durchlaufen. Für jede Fläche wird ein Projektionspunkt P' und die Distanz berechnet, und die zum geringsten Abstand zwischen Quelle und Projektion gehörenden Daten (Projektionspunkt, Distanz und zugehörige Fläche) gespeichert.

Ähnlich der Projektion auf eine Strecke wird hier die Quelle zuerst in die Ebene der Dreiecksfläche projiziert (analog zur Ebenen-Projektion in Abschnitt 3.3.3), und an-

schließend sichergestellt, dass sich der Projektionspunkt P' nicht außerhalb des Dreiecks befindet.

```
// initialisation
initialise distance_min with highest possible value

// project source point S onto hull
for each (hull face)
{
  calculate projection P' onto face
  calculate distance between S and P'
  if (distance < distance_min)
  {
    save face as face_min
    save distance as d_min
    save P' as projection P
  }
}

// determine signed distance
set distance sign (S inside/outside hull)

// find hull point closest to P
closest point CP is one of face_min's three vertices

// return results
return P, signed d_min and CP's index
```

Algorithmus 3.10: Projektion eines Punktes auf eine dreidimensionale konvexe Hülle

Um zu ermitteln, wo der in die Ebene projizierte Punkt $\tilde{P}$ bezüglich der Dreiecksbegrenzungen liegt, wird wieder die Vektorrechnung benutzt. Allgemein kann jeder Punkt einer Ebene als Linearkombination von zwei nicht-parallelen Basisvektoren in dieser Ebene dargestellt werden. So wird also $\tilde{P}$ als Kombination von zwei Vektoren angesetzt, die aus den Eckpunkten A, B und C der Dreiecksfläche berechnet werden (siehe Abbildung 3.5).

$$\vec{\tilde{P}} = \vec{A} + s \cdot \overrightarrow{AB} + t \cdot \overrightarrow{AC} \tag{3.8}$$

Die Parameter s und t errechnen sich dabei wieder analog zu Gleichung 3.5.

$$s = \frac{\overrightarrow{A\tilde{P}} \cdot \overrightarrow{AB}}{\overrightarrow{AB} \cdot \overrightarrow{AB}}, \qquad t = \frac{\overrightarrow{A\tilde{P}} \cdot \overrightarrow{AC}}{\overrightarrow{AC} \cdot \overrightarrow{AC}} \tag{3.9}$$

Diese Parameter können nun ausgewertet werden, um die Lage von $\tilde{P}$ bezüglich des Dreiecks und damit den Projektionspunkt P' zu ermitteln. Denn nur wenn die Bedingungen $s \geq 0$, $t \geq 0$ und $s + t \leq 1$ erfüllt sind, liegt $\tilde{P}$ innerhalb des Dreiecks und kann als Projektion P' verwendet werden. Ist $s \leq 0$, so muss P' auf der Strecke $\overline{AC}$ liegen und wird durch eine weitere Projektion von $\tilde{P}$ auf diese Strecke berechnet. Ist $t \leq 0$ so gilt das Selbe für die Strecke $\overline{AB}$, und für $s + t \geq 1$ wird $\tilde{P}$ auf die Strecke $\overline{BC}$ projiziert.

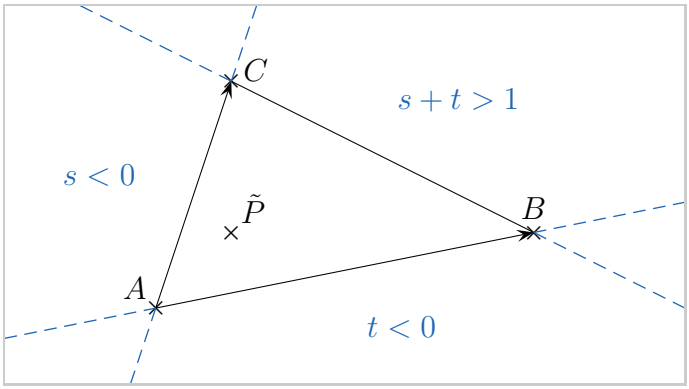


Abbildung 3.5: Lage eines Projektionspunktes bezüglich einer Dreiecksfläche

Sind schlussendlich der endgültige Projektionspunkt P und der Abstand zum ursprünglichen Quellpunkt S berechnet, muss noch das Vorzeichen der Distanz gesetzt werden. Definitionsgemäß wird ein negatives Vorzeichen gewählt, wenn die ursprüngliche Quellposition innerhalb der konvexen Hülle liegt, andernfalls erhält die Distanz ein positives Vorzeichen. Analog zum 2D-Algorithmus wird auch hier jener Eckpunkt der zuvor ermittelten Dreiecksfläche mit der kürzesten Distanz zum Projektionspunkt als Lautsprecherpunkt mit dem geringsten Abstand zurückgegeben.

4 Vergleichstest

Wie eingangs erwähnt, haben sich zwei Entwicklergruppen unabhängig voneinander mit der Formulierung des DBAP-Prinzips beschäftigt. Die beiden vorangegangenen Kapitel hatten die Implementation des Pd-Externals zum Inhalt, dessen Ausgangspunkt die DBAP-Implementation für Max/MSP des einen Teams bestehend aus Trond Lossius, Pascal Baltazar und Théo de la Hogue darstellt. Die andere Gruppe, die sich aus Dimitar Kostadinov, Joshua D. Reiss und Valeri Mladenov zusammensetzt, führte einen Vergleichstest des DBAP-Prinzips mit den Spatialisationsmethoden Vector Base Amplitude Panning (kurz VBAP) und Ambisonics durch, der in *Evaluation of Distance based amplitude panning for spatial audio* [KRM10] nachgelesen werden kann. Für weiterführende Informationen zu VBAP und Ambisonics sei auf [Pul97] bzw. [Fur90] verwiesen.

Dieser Vergleichstest aus dem Jahr 2010, der im Listening Room des Centre for Digital Music an der Queen Mary University of London stattfand, gliedert sich in zwei große Teile. Zum einen wurden die mit den verschiedenen Methoden berechneten relativen Amplituden der einzelnen Lautsprecher bei vorgegebenen virtuellen Quellpositionen bzw. Quellbewegungen verglichen (Abschnitt 4.1). Zum anderen wurde ein Hörtest durchgeführt, bei dem die Versuchspersonen die Positionen von virtuellen Quellen im Raum bestimmen sollten, ohne zu wissen, welche Spatialisationstechnik gerade verwendet wurde (Abschnitt 4.2).

4.1 Vergleich der Lautsprecheramplituden

Da bei der Spatialisation mit VBAP immer nur jenes Lautsprecherdreieck aktiv ist, auf dessen Fläche die virtuelle Quelle positioniert werden soll, wurde für den Vergleich

der Lautsprecheramplituden von DBAP und VBAP nur ein ausgewähltes Set von drei Lautsprechern verwendet. Während sich die Quelle von einem Punkt zu einem anderen bewegte, wurden die von beiden Methoden berechneten Amplitudenwerte der drei Lautsprecher aufgezeichnet und anschließend gegenübergestellt. Am Ergebnis ist zu erkennen, dass sich die Wertreihen qualitativ ähneln, jene für DBAP aber etwas glatter verlaufen.

Für den Vergleich der Amplituden von DBAP und Ambisonics wurde ein Array aus 16 Lautsprechern verwendet, und ebenfalls eine Gegenüberstellung der jeweiligen Amplitudenwerte für zwei verschiedene Quellpositionen vorgenommen. Hier konnte allerdings bei der Gegenüberstellung der Amplitudenwerte wenig Ähnlichkeit festgestellt werden.

4.2 Positionsbestimmung durch Versuchspersonen

In einem zweiten Teil wurde ein Hörtest mit 12 Versuchspersonen durchgeführt, welche die Positionen von verschieden virtuellen Quellen bestimmen sollten.

Bei der Gegenüberstellung der ersten beiden Techniken wurde für DBAP wiederum nur jeweils das von VBAP bespielte Lautsprecherdreieck ausgewählt. Zum Vergleich wurden drei Quellpositionen herangezogen, wobei jede mit einem anderen Tonbeispiel (Gitarre, weibliche Stimme, Rockband) bespielt wurde und sowohl die Positionen als auch die Methoden zufällig gereiht wurden. Außerdem sollten die Versuchspersonen eine Quellposition zusätzlich einmal von außerhalb des Sweet-Spots bestimmen. Am Ende des Versuchs wurden die Antworten gemittelt und das Ergebnis zeigte, dass sich auch hier DBAP und VBAP recht ähnlich sind.

Für den Vergleich von DBAP mit Ambisonics (dritter Ordnung) wurden alle 16 Lautsprecher des Arrays verwendet, und zwei Positionen zur Bestimmung gewählt, denen jeweils ein unterschiedliches Rockband-Tonbeispiel zugeordnet wurde. Bei der Betrachtung der gegenübergestellten Ergebnisse schnitt Ambisonics vor allem bei einer Quellposition hinter den Versuchspersonen deutlich schlechter ab, da die meisten Personen die Quelle vorne wahrnahmen. Bei DBAP war bei einer Hörposition außerhalb des Sweet-Spots keine große Änderung der Lokalisation zu sehen, interessant war allerdings, dass

Ambisonics entgegen der Theorie außerhalb des Sweet-Spots bessere Ergebnisse lieferte als innerhalb.

4.3 Reflexion

Wie die Ergebnisse zeigen, sind sich DBAP und VBAP recht ähnlich. Dies ist auch verständlich, wenn man sich vor Augen hält, dass diese beiden Spatialisationsmethoden relativ eng miteinander verwandt sind. Denn beide basieren auf der Auswertung der relativen Position von Lautsprecher und virtueller Quelle zueinander mit Hilfe der Vektorrechnung, wobei DBAP nur die Distanz verwendet, VBAP hingegen zusätzlich auch die Richtung. Mit diesem Wissen lässt sich auch erklären, dass DBAP einen glatteren Verlauf der relativen Amplituden bei einer Quellenbewegung aufweist, denn die VBAP-Amplituden werden durch die Richtungsinformationen zusätzlich beeinflusst.

DBAP und Ambisonics allerdings sind komplett verschiedene Ansätze. Ambisonics versucht, das komplette Schallfeld innerhalb der Anordnung zu reproduzieren, DBAP verfolgt dagegen einen funktionalen Ansatz zur gezielten Positionierung einer virtuellen Schallquelle. Damit erklärt sich auch der große Unterschied im Vergleich der relativen Amplituden, da schon die grundlegenden Prinzipien sehr unterschiedlich sind.

Da VBAP jeweils nur drei Lautsprecher verwendet, wurden auch im Hörversuch beim Vergleich mit DBAP jeweils nur drei Lautsprecher bespielt, was für den direkten Vergleich auch sinnvoll erscheint. Allerdings wäre es interessant zu testen, inwieweit sich die Ortung der Quelle verändert, wenn im Fall von DBAP dessen Vorteil ausgenutzt würde, alle Lautsprecher zu verwenden.

Im Hörversuch wurde zudem jeder Quellposition ein eigenes Tonbeispiel zugeordnet. Deshalb wäre es durchaus möglich, dass die Eigenschaften der Tonbeispiele die Ortung beeinflusst haben könnten. In einem neuerlichen Hörversuch wäre es denkbar, alle Positionen mit dem selben Tonbeispiel oder jede der Positionen mit verschiedenen Sounds zu bespielen, um diesen möglichen Einfluss weitgehend auszuschließen.

Leider wurden aufgrund der nicht gegebenen Vergleichbarkeit mit den anderen Techniken jene Eigenschaften nicht getestet, die DBAP zusätzlich bietet. Erstens wurde auf eine

Bewegung der virtuellen Quellen über ein Lautsprecherdreieck hinaus im durchgeführten Hörversuch verzichtet. Hier könnte man herausarbeiten, ob sich die Theorie bestätigt, dass bei DBAP akustisch glattere Trajektorien als bei VBAP möglich sind, da immer alle Lautsprecher aktiv sind. Zweitens wäre es durchaus interessant herauszufinden, wie gut die Positionierung einer Quelle innerhalb oder außerhalb des Lautsprecher-Arrays funktioniert, denn dies ist eine Facette, die DBAP sowohl von VBAP als auch von Ambisonics unterscheidet. Außerdem wurde aufgrund der eingeschränkten Aufstellungsmöglichkeiten der Vergleichstechniken nicht getestet, inwiefern sich verschiedene systematische oder auch willkürliche Lautsprecheranordnungen auf das Spatialisationsergebnis auswirken.

Für einen möglichen neuerlichen Test wäre es also wünschenswert, diese Fragestellungen zu integrieren und dementsprechende Versuche unter Verwendung von mehreren verschiedenen Quellpositionen durchzuführen.

Literatur

[Eri05] Christer Ericson. *Real-Time Collision Detection*. Morgan Kaufmann Publishers Inc., 2005.

[Fur90] Roger K. Furness. »Ambisonics – An Overview«. In: *Proceedings of the Audio Engineering Society 8th International Conference*. 1990, S. 181–190.

[KRM10] Dimitar Kostadinov, Joshua D. Reiss und Valeri Mladenov. »Evaluation of Distance based amplitude panning for spatial audio«. In: *Proceedings of the 2010 IEEE International Conference on Acoustics, Speech and Signal Processing*. 2010, S. 285–288.

[LBH09] Trond Lossius, Pascal Baltazar und Théo de la Hogue. »DBAP – Distance-based amplitude panning«. In: *Proceedings of the 2009 International Computer Music Conference*. 2009, S. 489–492.

[LBH12] Trond Lossius, Pascal Baltazar und Théo de la Hogue. *Source Code of a DBAP External for Max/MSP*. 2012. URL: https://github.com/jamoma/JamomaModular/tree/master/implementations/MaxMSP/jcom.dbap.

[Los10] Trond Lossius. *Calculation of DBAP equations*. 2010.

[ORo98] Joseph O'Rourke. *Computational Geometry in C*. 2. Aufl. Cambridge University Press, 1998.

[Pd] *Pure Data*. URL: http://www.puredata.info.

[Pul97] Ville Pulkki. »Virtual Sound Source Positioning Using Vector Base Amplitude Panning«. In: *Journal of the Audio Engineering Society* 45.6 (1997), S. 456–466.