Pontifícia Universidade Católica
do Rio de Janeiro

Marcelo Politzer Couto

HRTFs based 3D audio spatialization

Dissertação de Mestrado

Dissertation presented to the Programa de Pós-graduação em Informática of PUC-Rio in partial fulfillment of the requirements for the degree of Mestre em Informática.

Advisor: Prof. Bruno Feijo

Rio de Janeiro
April 2019

Pontifícia Universidade Católica
do Rio de Janeiro

Marcelo Politzer Couto

HRTFs based 3D audio spatialization

Dissertation presented to the Programa de Pós-graduação em Informática of PUC-Rio in partial fulfillment of the requirements for the degree of Mestre em Informática. Approved by the Examination Committee.

Prof. Bruno Feijo
Advisor
Departamento de Informática – PUC-Rio

Prof. Waldemar Celes Filho
Departamento de Informática – PUC-Rio

Dr. Augusto Cesar Espíndola Baffa
Departamento de Informática – PUC-Rio

Rio de Janeiro, April 16th, 2019

Marcelo Politzer Couto

Graduou-se como técnico em eletrônica pelo instituto de tecnologia ORT no ensino médio. Graduou-se em ciência da computação pela PUC-Rio enquando estagiava desenvolvendo sistemas de tempo real no laboratório PUC-GIGA. Pleiteou bolsa de mestrado pela CAPES durante seu mestrado e atualmente trabalha no desenvolvimento de firmware para roteadores na área de cybersecurity.

Bibliographic data

Politzer Couto, Marcelo

HRTFs based 3D audio spatialization / Marcelo Politzer Couto; advisor: Bruno Feijo. – Rio de janeiro: PUC-Rio, Departamento de Informática, 2019.

v., 40 f: il. color. ; 30 cm

Dissertação (mestrado) - Pontifícia Universidade Católica do Rio de Janeiro, Departamento de Informática.

Inclui bibliografia

1. Espacialização de áudio em 3D – Teses. 2. Computação Gráfica – Teses. 3. Processamento digital de sinais – Teses. 4. Espacialização de áudio em 3D;. 5. Computação Gráfica;. 6. Processamento digital de sinais;. I. Feijo, Bruno. II. Pontifícia Universidade Católica do Rio de Janeiro. Departamento de Informática. III. Título.

CDD: 004

Acknowledgments

I would like to thank first and foremost my family for their support and incentive to value and go after formal education. To my advisor for his patience and courage, we are taking a large sidestep outside his original field of study after all. Also, This study was financed in part by the Coordenação de Aperfeiçoamento de Pessoal de Nível Superior - Brasil (CAPES) Finance Code 001.

Abstract

Politzer Couto, Marcelo; Feijo, Bruno (Advisor). **HRTFs based 3D audio spatialization**. Rio de Janeiro, 2019. 40p. Dissertação de mestrado – Departamento de Informática, Pontifícia Universidade Católica do Rio de Janeiro.

Virtual Reality (VR) applications with low-latency head tracking require high-quality spatial audio effects. However, classic VR/game sound approaches cannot properly simulate the acoustic of the real world. Current audio research is moving towards 3D spatial audio to have a more realistic simulation. In 3D spatial audio, the listener has the sensation that sound comes from a particular direction in 3D space. In other words, the listener can localize a source based on audio and have a more coherent and immersive experience when paired with visual simulation. In this new context, game engines should provide sound designers with a set of 3D spatial audio tools. The following common effects are desirable in this type of toolbox: reverberations and reflections, which can be employed in the creation of caverns or churches (places with lots of echoes); modulation, which can increase the perceived variety of a recorded sound, by slightly varying its pitch (as in the sounds of footsteps); mixing and fading volumes, which can create dramatic moments
in storytelling and music reproduction. In this work, we propose a real-time audio engine to spatialize sound point sources in virtual environments. This engine is an open-source architecture that provides a basic set of audio effects and an efficient way to mix and match them. We implement 3D audio spatialization by leveraging recorded head-related impulse responses (HRIRs) and we produce special sound effects with digital signal processing (DSP) techniques. Although some powerful commercial audio SDKs for Virtual Reality are currently available (e.g. Oculus), our audio engine prototype may be a flexible option when adaptation, simplification, testing, and parameter tuning are necessary.

Keywords

3D audio spatialization; Computer Graphics; Digital signal processing;

Resumo

Politzer Couto, Marcelo; Feijo, Bruno. **Espacialização sonora em 3D baseada em HRTFs**. Rio de Janeiro, 2019. 40p. Dissertação de Mestrado – Departamento de Informática, Pontifícia Universidade Católica do Rio de Janeiro.

Aplicações de Realidade Virtual (VR) com rastreamento de movimentos da cabeça precisam de efeitos de espacialização de alta qualidade. A abordagem tradicional para RV/jogos (interpolação dos canais L+R para construção do estereo) se mostrou insuficiente por ser incapaz de simular a acústica do mundo real. Por isso a pesquisa na área tem migrado para espacialização 3D do áudio. O receptor tem a sensação de que o som veio de um local no espaço 3D. Em outras palavras, ele pode localizar o emissor apenas pelo áudio por consequência permite a construção ambientes mais imersivos e coerentes quando usados em conjunto de técnicas visuáis. Nesse novo contexto, motores de jogos devem prover aos designers de áudio uma gama de ferramentas especializadas para a espacialização de àudio 3D além as de uso geral, que encluem: reverberações e reflexões usadas na construção de ambientes como igrejas e cavernas (locais com ecos); modulação, para criar variações de frequência e aliviar na repetitividade de sons recorrentes (como os de passos e tiros); mix e fade de volumes, utilizado na criação de momentos dramáticos na história e reprodução musical. Nesse trabalho, nós propomos um motor de áudio de tempo real para espacialização de fontes sonoras pontuais em ambientes virtuais. Vai possuir uma arquitetura documentada e de código aberto que provê um conjunto de efeitos e a habilidade de os compor. Nós implementamos a espacialização de áudio em 3D sobre bancos de dados de respostas impulsionais da cabeça (HRIRs) e efeitos sonoros com técnicas de processamento digital de sinais (DSP). Apesar da existência de sistemas comerciais poderosos de áudio para VR estejam disponíveis (e.g. Oculus), nosso protótipo pode ser uma alternativa se a simplicidade, testabilidade e ajustes forem levados em conta.

Palavras-chave

Espacialização de áudio em 3D; Computação Gráfica; Processamento digital de sinais;

Table of contents

List of figures

List of Abreviations

ADC	–	Analog to Digital Converter
DAC	–	Digital to Analog Converter
DSP	–	Digital Signal Processing
FFT	–	Fast Fourier Transform
HRIRs	–	Head Related Impulse Response
HRTFs	–	Head Related Transfer Functions
LTI	–	Linear Time Invariant system
OLA	–	OverLap Add
STFT	–	Short Time Fourier Transform
WFs	–	Window Functions
OS	–	Operating System
GDC	–	Games Developer Conference
VRDC	–	Virtual Reality Developer Conference
CS	–	Computer Science
POV	–	Point of View

1
Introduction

In 2013 Oculus[1] released the first version of their virtual reality (VR) system to the consumer market. Since then other companies have joined with their own head-mounted display (HMD) products such as the HTC Vive. It seems that this new way of playing video games is here to stay. Developers are "moving" quickly to adapt their game engines to this new environment.

In audio, games used to consider only the horizontal axis of objects. A technique that was used even for 3D games. This solution, however, can not convey realistic 3D experiences and is considered insufficient for VR applications. Because of this limitation, game engines are moving towards full 3D audio spatialization. In it, audio is encoded with directionality in such a way that players become able to pinpoint the source's location when are using headphones.

Figure 1.1 shows a hypothetical scene from a game. In this scene, $source_1$ and $source_2$ are sound emitters and the ear icon is a listener. We illustrate that audio can have direct and indirect paths as indicated by the arrows in the figure. In $reflection_1$'s case, audio gets reflected by the wall before going towards the listener. Some of its energy gets absorbed by the wall before going towards the listener's ears.

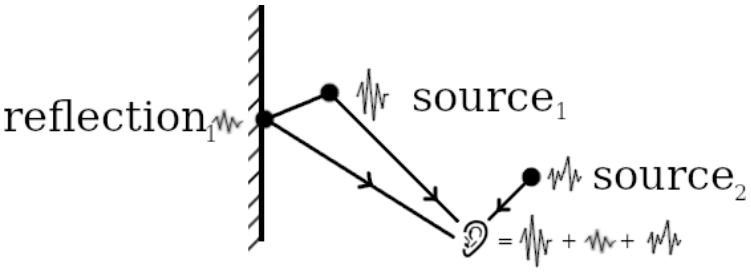


Figure 1.1: Scene with two audio sources and a listener. The listener receives audio from $source_1$, $source_2$ and the reflection of $source_1$.

To simulate a scene's acoustics precisely, we would need to consider that sound may get attenuated, delayed, reflected, refracted, pitch shifted, frequency filtered, and modulated on its way from source to listener. The path needs to

[1] https://www.oculus.com/

be ray traced on the scene's geometry to compute these effects accurately, an intensive computation. To achieve a real time solution, we ignore most of these effects and only implement point sources with direct rays (such as to ignore $reflection_1$ in Figure 1.1). Even with these simplifications, the system is still capable of 3D spatialization as discussed in Section 1.1 (localization) and 1.2 (spatialization).

The main contribution of this dissertation is to propose a digital signal processing Virtual Machine to help the implementation of audio spatialization in virtual environments, as presented in Chapter 4 (design and implementation). We propose a virtual machine dedicated to the execution of digital signal processing (DSP) instructions, which are the primitives of the above-mentioned effects. The proposed design allows users to compose complex operations in terms of simpler ones. By the end of Chapter 4, we give an example: a sequence of DSP instructions to implement the spatialization of sound sources such as $source_1$ and $source_2$ presented in Figure 1.1.

In Chapter 2 (state of the art), we present some foundational articles on the theory of 3D audio spatialization. Most of the theory is well established but used sparsely in this field. Also we show some recent articles in the area. Interesting development occurred since 2013, when this subject became a trend. Later, in this Chapter, we give an overview of what is being used in practice, that is: in game engines.

In Chapter 3 (theoretical background), we justify our design choices over the theory of various subjects. Section 3.1 (signals and systems) shows that the idea is to model audio as signal and the localization as a system. After that, we introduce different audio representations in section 3.2 (audio in a nutshell). We try to map it to signals and systems on a intuitive level. After that we cover how to leverage an already existing computer graphics coordinate system to represent the position of audio sources in section 3.4 (affine spaces). Finally, in section 3.5 (spatialization) we show how these pieces fit together to achieve the spatialization effect.

1.1 Localization

Localization is the human ability to infer a sound source location based on how it is perceived differently by each ear, characteristics of the sound itself and a supplementary mental model of how known sounds behave, even though two ears are not enough for a perfect triangulation. Lord Rayleigh (1) released an article on the subject, which eventually became the duplex theory of localization. This theory states that we use inter-aural level differences (ILD)

and inter-aural time differences (ITD) to infer objects positions.

1.1.1 Inter-aural level differences - ILD

When a sound is played off center of the head, one of the ears is in shadow, because the head is in the way. This causes a perceptible difference as shown in the Figure (1.2). This effect is more preeminent for higher frequencies, as lower frequencies refract around the head and dampen this effect.

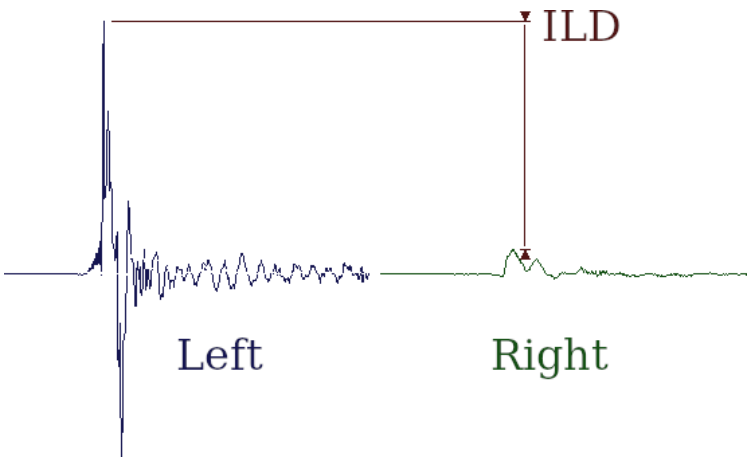


Figure 1.2: HRIR(90,0,1) that shows the ILD difference between left and right ear. HRIR is the impulse response that relates the source location and the ear location. In this case the source is 90 degrees (horizontal angle) and 0 degree (vertical angle) in relation to the head.

1.1.2 Inter-aural time differences - ITD

A second effect is based on the time it takes for the sound to travel around the head. The speed of sound is slow enough that the brain can detect the phase difference between each ear, as shown in the Figure (1.3), when the frequency increases the wavelength shortens. This happens until it gets shorter than the size of our heads. At this point, we can no longer infer direction, as described by Hartmann et al (2).

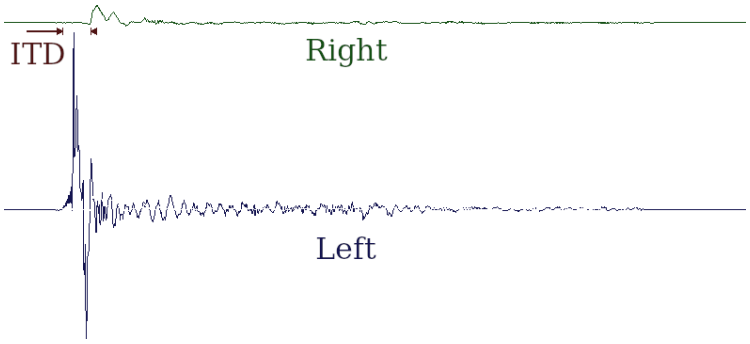


Figure 1.3: HRIR(90,0,1) that shows the ITD difference between left and right ear. The source and the head are in the same position described in the previous Figure.

1.2 Spatialization, HRIR and HRTF

In a process called audio spatialization, we use digital signal processing (DSP) to encode localization queues into a regular audio source. When played with headphones, this altered audio will be perceived as having a physical location. This is the basis of what is known as a 3D audio system.

The localization can be tough of as a function with spherical coordinates (θ, ϕ, ρ) as input and localization queues as output. The inputs are: θ for the horizontal angle, ϕ for vertical and ρ for the radius. Figure 1.3 shows a stereo recording for $(90, 0, 1)$. The blue signal is the left ear and the green signal is the right ear.

This function is usually constructed from measurements. Gardner et al (3) show how they did this for a KEMAR dummy head. In short, the process is as follows: 1) Place a microphone in each ear of the dummy (at the origin of Figure 1.4). 2) Create a stereo recording, a channel for each ear, of the noise coming from a speaker positioned at θ, ϕ, ρ (one recording for each position, indicated by red dots in Figure 1.4). This group of recording is called the head related impulse response (HRIR). The KEMAR database consist of a list of '.wav' files with these recordings and their references to the values of θ, ϕ and ρ. The value of ρ is fixed.

We convert this HRIR function into its frequency domain representation. When this happens its name changes to head related transfer function (HRTF). We do this as an optimization to compute the convolution operation faster. This transformation is done only once as a preprocessing step.

In short, HRTF is how a particular head behaves to incoming sound (multiple angles), in the frequency domain.

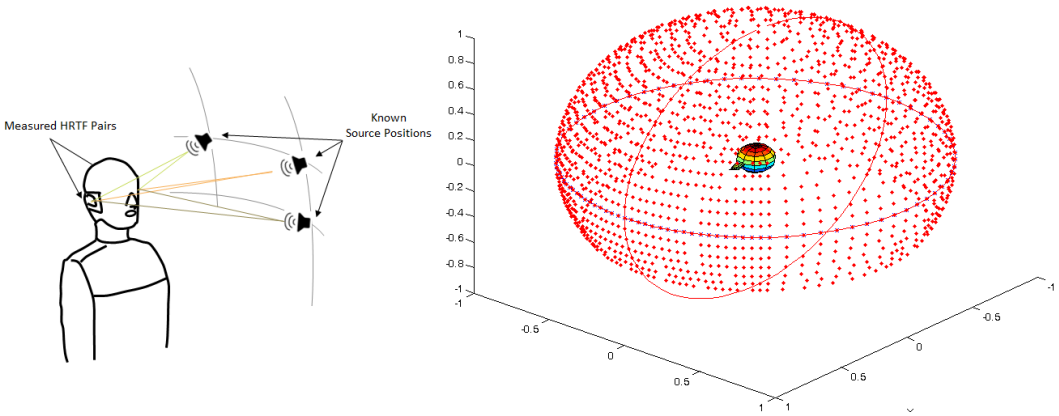


Figure 1.4: Head Related Impulse Response recording. A dummy head at the center and speakers at the red dots.

2
Related work

Most of the concepts used in audio localization are well known results from different fields of science, and spatialization arises from their use in conjunction. For example, the fourier transform is attributed to Fourier himself in his article "Théorie analytique de la chaleur" published in French in the year of 1822 and later translated to English by Alexander Freeman in 1878. The subject of audio localization is at least as old as 1907, when Lord Rayleigh published his studies in the article "On our perception of sound direction" (1). Books on digital signal processing, such as "Theory and application of digital signal processing", are at least as old as 1975 (4).

We can find initial references to sound rendering in Takala & Hahn (5) and some other local work by Maffra (6). However, there are recent developments in the area. Commercial use of 3D audio systems brought focus back into this field, with interesting advancements. Raghuvanshi et al (7) brought accurate modeling of reverberations, delays and attenuation based on wave simulations on scene geometry. Schissler et al (8) proposed a solution for area and volumetric sound sources (as opposed to punctual) by using spherical harmonics. Brungart et al (9) measured the latency effects on the accuracy of human localization for jet fighters missile detection systems. Gardner et al (3) compiled HRIR recordings of a KEMAR dummy and made them publicly available.

2.1
In game engines

At the time of writing this dissertation, game engines are still catching up to the recent VR requirements. Established engines are mostly incorporating dedicated audio systems to provide this functionality, more on this below.

2.1.1
Unity

According to Unity documentation [1], this game engine provides a built-in panning effect (not 3D audio) and a HRTF demo.

[1] https://docs.unity3d.com/2019.1/Documentation/Manual/AudioSpatializerSDK.html

In Unity, the HRTF demo uses the KEMAR (3) data set, has a log attenuation and a simple reverb. It states, however, limitations like artifacts due to the partial spatialization algorithm implemented. The authors of Unity documentation advise users to look for an external plugin for production system.

The Void[2] is an application based on "hyper reality", which is a kind of virtual reality with tactile elements. They achieve high level of realism by using physical props for the tactile part. They use a mixture of Unity's native audio, a no longer available plugin called *3dception* (bought by facebook) and pre-rendered audio as their stack. Their senior audio engineer goes into the details in a VRDC2016 [3] presentation.

An interesting piece of technology that The Void used is a chest piece to enhance the sound of explosions and other low frequency effects [4].

2.1.2 Unreal 4

Epic announced a new Audio system for the unreal engine 4 at GDC2017 [5] in wich Unreal's engineers implemented a real-time pipeline for audio and a variety of effects.

According to Unreal documentation [6], the march/2018 release of Unreal 4 added the Oculus Audio room modeling reverb with volumetric parameters and ambisonics source playback. In this version, they also added support for Resonance Audio (2.1.4) as a plugin.

2.1.3 Godot

In its current version (3.0), Godot implemented audio buses [7] , but from their list of effects there is no 3D audio support.

2.1.4 Resonance Audio - Google

Resonance Audio [8] is a dedicated 3D Audio spatialization engine by Google. It implements its audio spatialization via ambisonics, a more

[2] `https://www.thevoid.com/`
[3] `https://www.youtube.com/watch?v=nBrouOFtX_0`
[4] https://subpac.com/
[5] https://www.youtube.com/watch?v=ErejaBCicds
[6] https://www.unrealengine.com/en-US/blog/unreal-engine-4-19-released
[7] https://docs.godotengine.org/en/3.0/tutorials/audio/audio_buses.html
[8] `https://resonance-audio.github.io/resonance-audio/`

lightweight but less precise spatialization algorithm, possibly focused on the mobile market where the CPUs are less powerful than a proper desktop.

2.1.5 Oculus Audio - Facebook

Oculus Audio [9] provides a robust toolbox for 3D audio, with the following features: HRTFs based spatialization, head tracking, volumetric sources, near-field, environmental modeling with a shoebox model, ambisonics, attenuation and reflections.

2.1.6 Project Triton

Microsoft Research is getting very interesting results by using pre-computed wave simulations (a 7D function: listener's x,y,z source's x,y,z and t), as reported by Raghuvanshi and Snyder (7). They propose an encoding of impulse responses that removes two of the seven dimensions, making it possible to handle large scenes. They also propose a sound rendering method that generates artifact-free audio even for fast motion and sudden changes in reverberance. This system has been used commercially in a game called Gears of War 4 [10]. See their GDC2017 talk on the Project Triton (precomputed environmental wave acoustics)[11].

[9] https://developer.oculus.com/documentation/audiosdk/latest/concepts/audiosdk-features/

[10] https://gearsofwar.com/en-us/games/gears-of-war-4

[11] https://www.youtube.com/watch?v=qCUEGvIgco8

3
Theoretical background

We start with a review of signals and systems (section 3.1). We present time and frequency domains, continuous and discrete time, linear time invariant - LTI systems and their properties (Section 3.1.2). Also, we present the discrete fourier transform - DFT (Section 3.1.4), fast fourier transform - FFT (Section 3.1.5) and the fast convolution - overlap-add. The basic building blocks of audio spatialization are also presented.

After presenting the above-mentioned basic concepts, we introduce audio 3.2 representations and their signals and systems equivalents. Finally, we discuss how to connect the audio system to other systems of a a game engine. This includes coordinate systems transformations into listener's space 3.4, and HRTF triangulation and lookup (Section 3.3.1).

3.1
Signals and systems

We would like to start by saying that this is only a short introduction on the subject of signals and systems with a focus on the spatialization aspects. A complete view of it requires studying long books such as (Oppenheim's signals and systems (10)) and although encouraged, does not fit this space. There is also a set of video lectures by in MIT's open courseware [1] that may help in the understanding.

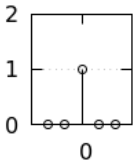
3.1.1
Signals

Signals are streams of data, functions of one or multiple variables such as time. From our example we have the following signals:

- Ps is a continous, constant signal.
- $Pd(x, y, z, t)$ is a continous, multi variable signal.
- $x(t)$ is a continous, single variable signal.
- $x[n]$ is a discrete, single variable signal.

[1] https://ocw.mit.edu/resources/res-6-007-signals-and-systems-spring-2011/video-lectures/

Besides those, we have a few special ones. One particularly interesting is the dirac's delta or impulse, it has the special property of being the convolution identity (more on this later). It is defined as:



$$\delta[n] = \begin{cases} 0, n \neq 0 \\ 1, n = 0 \end{cases}$$

Figure 3.1: Dirac's Delta

3.1.2 Linear Time-Invariant, (LTI)

Systems are functions of signals, or second order functions. They take signals in, transform them and give signals out. They can have different classifications depending on the guarantees they provide. We will focus on the LTI category, the one our spatialization database belongs.

As for notation, we will represent a system as a function application on a signal, like so: $y[t] = S(x[t])$, (x and y are signals, S is a system). We will also go into two ways of how to apply a system to a signal (convolution in time and in frequency) along with a small proof of why it is like so. Starting with the properties of a LTI system:

3.1.2.1 Linearity

It is said that a system is linear if it has the following properties:

– Additivity: the transformation of a sum is the sum of the transformations:

$$S(x_0[t] + x_1[t]) = S(x_0[t]) + S(x_1[t])$$

– Homogeneity: or the propagation of constants:

$$S(ax[t]) = aS(x[t])$$

3.1.2.2 Time invariant

Time invariance means that a system doesn't care about time origin. In other words, if we apply the system now or in the future, the only difference

is a shift in time α, formally:

$$\begin{cases} y[t] & = S(x[t]) \\ y[t-\alpha] & = S(x[t-\alpha]) \end{cases}$$

3.1.3 Convolution

Convolution is the transformation of the signal $x[n]$ by the system $h[n]$. Defined as:

$$y[n] = \sum_{k=-\infty}^{\infty} x[k]h[n-k]$$

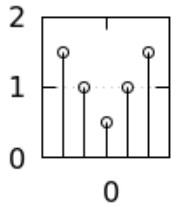
And has the "star" representation:

$$y[n] = x[n] * h[n]$$

You might have noticed that we have represented h in a similar way to a signal. That is no accident, In the following sub section we will show that LTI systems can be represented as a delayed impulse decomposition of one.

3.1.3.1 Delayed impulse signal decomposition

Consider the signal $x[n]$ defined as:



$$x[n] = \begin{cases} 1.5, n = -2 \\ 1.0, n = -1 \\ 0.5, n = 0 \\ 1.0, n = 1 \\ 1.5, n = 2 \\ 0, otherwise \end{cases}$$

We will transform the case by case definition into a continous one by taking a shifted $x[n]$ using the δ function. For $n = -2$ only we will have:

$$x_0[n] = 1.5, n = -2$$
$$x_0[n] = 1.5 \cdot \delta[n+2]$$
$$x_0[n] = x[-2] \cdot \delta[n+2]$$

If we apply this procedure to each branching case we will have:

$$\begin{aligned}
x_0[n] = 1.5, n = -2 &\quad\rightarrow\quad x_0[n] = x[-2] \cdot \delta[n+2] \\
x_1[n] = 1.0, n = -1 &\quad\rightarrow\quad x_1[n] = x[-1] \cdot \delta[n+1] \\
x_2[n] = 0.5, n = +0 &\quad\rightarrow\quad x_2[n] = x[+0] \cdot \delta[n+0] \\
x_3[n] = 1.0, n = +1 &\quad\rightarrow\quad x_3[n] = x[+1] \cdot \delta[n-1] \\
x_4[n] = 1.5, n = +2 &\quad\rightarrow\quad x_4[n] = x[+2] \cdot \delta[n-2]
\end{aligned}$$

Due to the linearity of LTI systems we can reconstruct $x[n]$ by summing all the parts:

$$x[n] = x_0[n] + x_1[n] + x_2[n] + x_3[n] + x_4[n]$$

And now expanded back:

$$\begin{aligned}
x[n] = x[-2] \cdot \delta[n+2] \\
+ x[-1] \cdot \delta[n+1] \\
+ x[+0] \cdot \delta[n+0] \\
+ x[+1] \cdot \delta[n-1] \\
+ x[+2] \cdot \delta[n-2]
\end{aligned}$$

Rewriting as a sum:

$$x[n] = \sum_{k=-2}^{2} x[k]\delta[n-k]$$

Generalizing:

$$x[n] = \sum_{k=-\infty}^{\infty} x[k]\delta[n-k]$$

If we apply a system S to both sides of the equality:

$$S(x[n]) = S(\sum_{k=-\infty}^{\infty} x[k]\delta[n-k])$$

And due to linearity, $x[k]$ can be extracted:

$$S(x[n]) = \sum_{k=-\infty}^{\infty} x[k]S(\delta[n-k])$$

Expanding $S(x[n])$ and applying $S(\delta[n-k])$, we obtain $y[n]$ and the system's impulse response $S(\delta[n-k])$ or $h[n]$. We then arrive at the definition of convolution.

In conclusion:

Convolution is defined as:

$$y[n] = \sum_{k=-\infty}^{\infty} x[k]h[n-k]$$

Commonly represented as:

$$y[n] = x[n] * h[n]$$

- Every LTI system has an equivalent signal representation defined by its delayed impulse decomposition.
- Convolution is the operation that transforms a signal by a systems in this form.
- δ is the identity function for this operation.
- Convolution is a linear, commutative operation.
- Let N_x be the length of x and N_h be the length of h. The result of their convolution has length: $N_x + N_h - 1$

3.1.4 Discrete Fourier transform - DFT

We will talk about a second type of signal decomposition, the discrete-time fourier transform. It is an expansion of a periodic function $x[n]$ in terms of an infinite sum of complex exponentials (or sines and cosines [2]).

For a periodic function $x[n]$ repeating every N samples, the series has the form:

Analysis equation:

$$a_k = \frac{1}{N}\sum_{n=0}^{N-1} x[n]e^{-ik\Omega_0 n}$$

[2]Euler's formula: $e^{in} = cos(n) + isin(n)$

Synthesis equation:

$$x[n] = \sum_{k=0}^{N-1} a_k e^{ik\Omega_0 n}$$

It is defined as the sum:

$$x_{2\pi}[\omega] = \sum_{n=-\infty}^{\infty} x[n] e^{-i\omega n}$$

And has the following properties:

– it has an inverse transformation such that:

$$x[n] = DFT^{-1}\{DFT\{x[n]\}\} \tag{3-1}$$

– convolution becomes a component wise multiplication:

$$x[n] * h[n] = DFT^{-1}\{DFT\{x[n]\} \cdot DFT\{h[n]\}\} \tag{3-2}$$

– has a fast implementation in $O(nlog(n))$ called Fast Fourier Transform (FFT).

3.1.5 Fast Fourier Transform - FFT

Fast fourier transform is a $O(nlog(n))$ DFT algorithm (11). We implemented two variants: Gentleman-Sande and Cooley-Tukey. Both restricted to n being power of 2.

3.1.6 Fast Convolution

We use DFT convolution property (3-2) and FFT to implement fast convolution. This method has a complexity of $O(nlog(n))$ as opposed to the direct $O(n^2)$ approach. This becomes advantageous for values of n larger than around 30.

3.1.6.1 Sectioned Convolution & overlap-add - OLA

Convolution is defined for whole signals, or one period for the periodic ones. However in practice, to achieve spatialization, we need convolution to work on blocks smaller than that, in the ballpark of milliseconds.

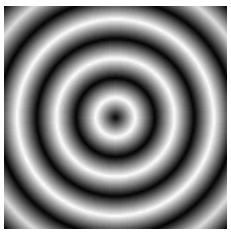
To do this we use a variant of fast convolution called sectioned convolution. This method, described in Rabiner et al chapter 2.25 (4), is capable of

convolving a signal with small slices of another. We can use this to apply the HRTF to small parts of our input signal at a time and with this keep a low audio latency.

In the overlap-add method we take N_a samples from $a[n]$ into a buffer of size $N_a + N_b$ paded with zeros. Do the same for b, that is, take N_b samples from $b[n]$ into a buffer of size $N_a + N_b$ also padded with zeros. Then we do an FFT on both buffers, multiply them by each other and do an inverse FFT of the result. This produces a full buffer with the convolution of a and b. Due to the fact that the convolution produces $N_a + N_b - 1$ samples, we will have a full buffer to pass forward.

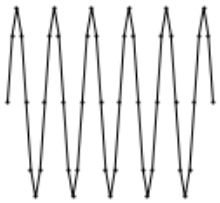
3.2 Audio in a nutshell

In this section we give a brief description of audio in its various forms in a attempt to ground the theory of signals and systems from Section 3.1. Our main focus are steps 4 and 5, where we are operating directly in. Where the digital signal processing happen and where audio buffers are pushed to the sound card.



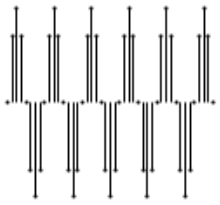
1. Sound is a periodic variation in pressure. Defined as a field $P(x, y, z, t) = Ps + Pd(x, y, z, t)$, where:

   - Ps: is the static pressure (due to medium)
   - $Pd(x, y, z, t)$: dynamic part (due to emiters)



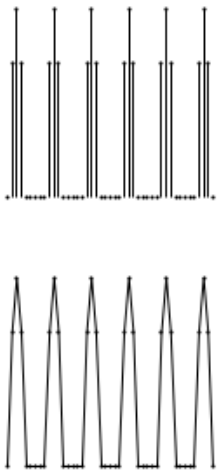
2. With a microphone positioned at x', y', z', its electrical output is be $x(t)$, the recording of $Pd(x', y', z', t)$.

   - $x(t)$ is scaled down into a $[-1, 1]$ range.



3. By aplying a Analog to Digital Converter we digitize the signal:

   - For a small $dt = 1/f$, we record $x(dt)$ and keep it as a sequence.



4. In this form we can easily apply mathematical functions to transform it, as an example:

   – $y[n] = max(0, x[n])$.

5. Every few milliseconds the OS will ask for audio data via a callback, in those we give it the samples we computed. With it, the audio hardware passes the samples through a Digital to Analog Converter to construct new analog signal and finally gives it to the headphones to be played.

3.3 Head related transfer function - HRTF

As discussed in the introduction, HRTF is a function from θ, ϕ, ρ to spatialization queues. A system (as in signals and systems) that can be convolved with an arbitrary signal to encode spatialization queues in it.

The KEMAR database provides this function as discrete θ, ϕ, ρ from the in ear microphone recordings (HRIR). To use this in practice, we will compute the HRTF by converting it into the frequency domain (apply the Fourier Transform) as a pre-processing stage.

During run-time we need to solve an interpolation problem. Since this function is only defined for a discrete set of angles we need some scheme for computing the HRTF of intermediate angles not obtainable directly from the samples, that is, how to solve this function for arbitrary angles.Moreover, we have to find a way to do queries quickly, in order to honor our real time promise.

3.3.1 Triangulation

We solve both of the above-mentioned problems by interpolating the triangulation of the sample points. If we assume that the HRTF is linear for small angles, we can linearly interpolate the vertices for arbitrary coordinates. What we do in practice is to do a weighted linear interpolation of the 3 closest recordings with their barycentric coordinates as weights. For lookups that are close to an exact angle the result is also close to the actual recording.

As for queries, they can be accelerated if we store the triangles adjacency and cache the coordinate lookups (as shown below).

The KEMAR recording has a fixed ρ. Because of that we will only use θ, ϕ as coordinates for a 2D triangle mesh. We will store triangles along with links to their neighbors. In Figure 3.2, triangle (1,3,5) has links to (1,2,3), (3,4,5) and (1,5,6).

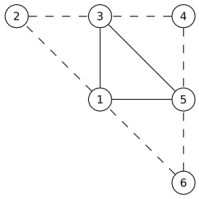


Figure 3.2: Triangle (1,3,5) and its neighbors

With this structure we can do better than a linear search on the triangles. Figure 3.3 shows a search starting in the blue triangle and ending in the green triangle. This difference adds up for a larger mesh. Another accelerator method is to cache triangle lookups per object. With this, when an object does its next lookup it will be the same triangle most of the time. When it isn't, it will likely still be close by. This should bring the lookup cost further down, close to O(1) for objects moving slowly.

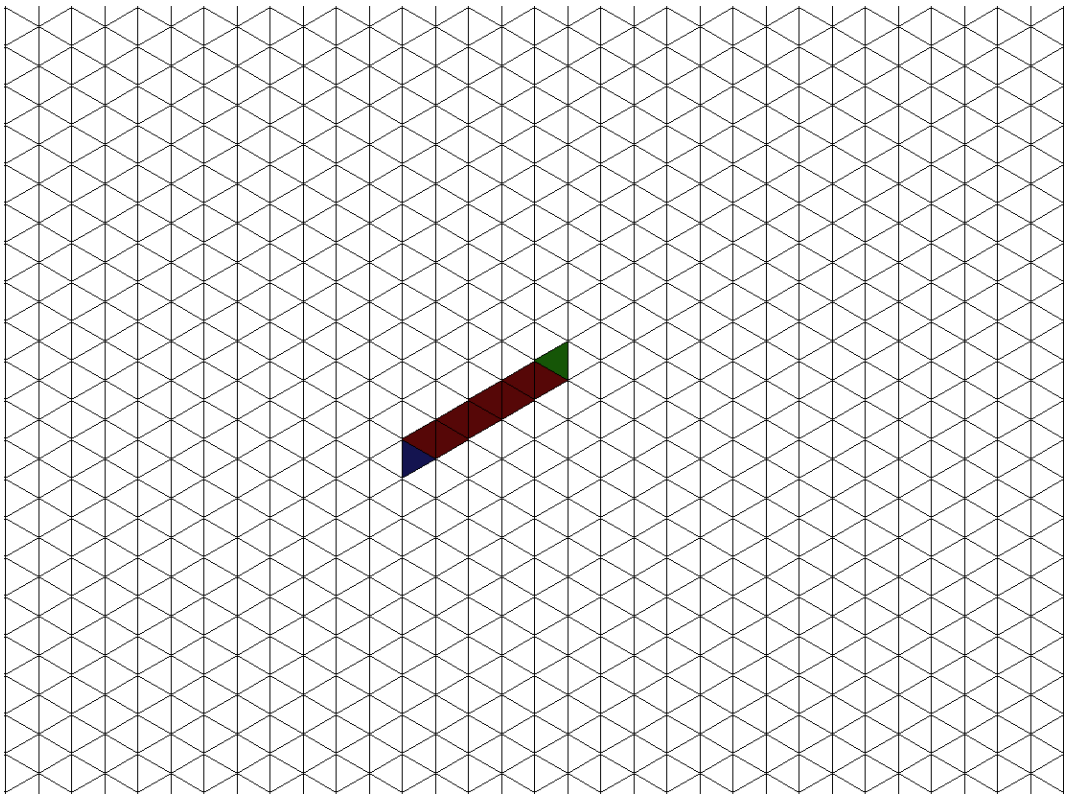


Figure 3.3: Mesh traversal

Due to time constraints, the demo implemented a linear search on the triangles. The algorithms described here were discussed during the preliminary design, but they were left for a future improvement of the system.

3.4 Affine Spaces

The HRTF function is defined in relation to the head, that is, the head is stationary at the origin and sound sources are played around it. For our engine to spatialize correctly we need a way to to change the coordinate system of sound sources match this scenario so that we can do the HRTF lookup correctly. This is a known problem with a known solution in the field of computer graphics. This is the problem of implementing a camera. There they use a coordinate system for the camera called eye space in which the camera is always at the center pointing forward and all objects are transformed into this space before the drawing is done. In the end the camera is at the center and objects positions relative to it. This exactly what we need. In other words we put our "ears" together with our "eyes".

A common technique to implement this is to use the properties of affine spaces in 4x4 matrices (to also encode 3d positions). In Figure 3.4 we show the commonly used coordinate spaces in graphics as rectangles: object, world, eye and clip. Edges represent transitions between them and are encoded as the 4x4 matrices mentioned above. As an example when vertices in object space are multiplied by the model matrix they transition to world space. This is a change of basis in linear algebra terms.

Figure 3.4 shows arrows pointing backwards. They represent the inverse of a matrix (transitions vertices in the opposite direction).

The solution is to transform sources to world space with their model matrices and then to eye space by transforming the result by the listener's view matrix.

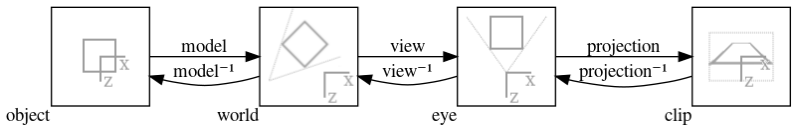


Figure 3.4: Commonly used spaces in computer graphics

3.5 Spatialization

We implement spatialization by convolving the HRIR with each sound we want to play. The process is as follows. Firstly, select the recording that

best matches (θ, ϕ, ρ), as explained in 3.3.1 (triangulation). Then, transform the data to the listener's space, as explained in section 3.4 (affine spaces). After that, bring $x[t]$ into the frequency domain by applying DFT to it. Then, do an element wise multiplication in the frequency domain between $x[t]$ and $HRTF[t]$ for each ear (L+R). Finally, bring them both back to the time domain with a DFT^{-1} for each. This produces a stereo signal that can be send directly to the headphones. For multiple sound sources, we do this process separately for each source, summing all (L) channels and all (R) channels before sending them to the headphones.

Below are the equations for the $x_l[n]$ (left) and $x_r[n]$ (right) ears:

$$\begin{aligned} x[n] * h[n] &= DFT^{-1}\{DFT\{x[n]\} \cdot DFT\{h[n]\}\} \\ HRTF_l[n], HRTF_r[n] &= HRTF_{database}(\theta, \phi, \rho)[n] \\ x_l[n] &= DFT^{-1}\{DFT\{x[n]\} \cdot HRTF_l[n]\} \\ x_r[n] &= DFT^{-1}\{DFT\{x[n]\} \cdot HRTF_r[n]\} \end{aligned}$$

Figure 3.5 presents a graph representation of the above-mentioned process. The reader should notice that HRTF generates two pairs of signals in the complex plane (a pair for the right channel and another one for the left channel). In this case, each channel has a real part ($HRTF_r$) and an imaginary one ($HRTF_l$). We indicate this output with the symbol C^2.

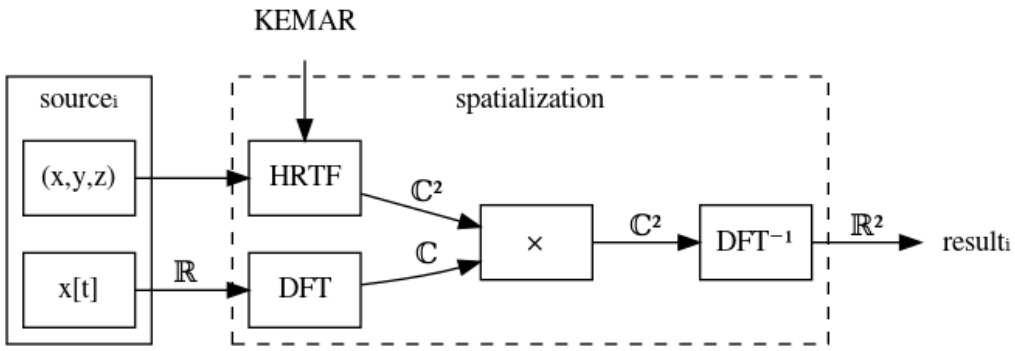


Figure 3.5: Spatialization effect implemented as the fast convolution between x[t] with HRTF.

The elements in the Figure 3.5 are:

– (x,y,z) - Object position in listener's space.

– x[t] - audio to be spatialized.
– HRTF - $HTRF_{database}$ function lookup. It yields $HRTF_l$ and $HRTF_r$.
– DFT - discrete fourier transform (implemented with FFT).
– $\times$ - element wise multiplication between array elements.
– DFT^{-1} - inverse DFT (implemented with FFT).

As a note, we did a small simplification here to ease the algorithm's understanding. Check 3.1.6.1 (Sectioned convolution) for the complete story.

3.5.1 DAG - Directed Acyclic Graph

In the present work, we use graphs to design any process for 3D audio spatialization. See the graph in Figure 3.5. In the spatialization of Figure 3.5, each node represents a DSP operation and each one of its edges represents a flow of data. Incoming edges are arguments and outgoing edges are its results. A node must have its inputs ready before it is able to execute, that is: an edge can also be though of as a dependency between nodes. Also, this graph must be acyclic, i.e.: no operation can depend on the result of itself, directly or indirectly.

This graph is a well-known category of graph called a Directed Acyclic Graph (DAG). All DAGs have a property called topological ordering that states the following: there exist a configuration of nodes such that if we put all of them into a line, edges will point to the right of itself. This operation is called topological sorting and can be accomplished in $O(m+n)$ complexity as described in chapter 3.6 of Tardos et al. (12). The graph G is defined below with its graphical representation in Figure 3.6. And after sorting we get the list of vertices $L = (a, b, c, d)$, or visually in Figure 3.7.

$$G = \{V = \{a, b, c, d\}, E = \{a \to b, a \to c, b \to c, c \to d\}\}$$

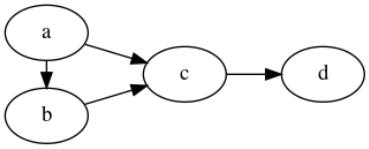


Figure 3.6: Example of a DAG

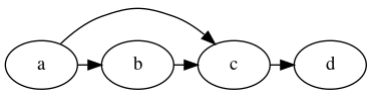


Figure 3.7: Example of a topologically sorted DAG. Nodes in order are: (a,b,c,d)

4
Design and Implementation - The Virtual Machine

In this chapter, we put the theory from Chapter 3 into practice and discuss a possible implementation of a real-time audio spatialization engine. Also, we propose a digital signal processing Virtual Machine (called DSP-VM) to help the implementation of audio spatialization in virtual environments. In the first section (4.1), we explore the system properties and show how they guide our proposed solution. After that, in section 4.2, we go into the data structures and algorithms of our proposed solution. Finally, in Section 4.3, we describe the implementation we arrived at (using the C programming language).

4.1
Properties

For a successful design, we must at least implement the graph of DSP operations (Figure 3.5) presented in section 3.5 (spatialization). Ideally, our model should be able to support effects, audio synthesis, and mixing as discussed before.

In practical terms, this graph is a Directed Acyclic Graph (DAG) as discussed in Section 3.5.1. In this approach, we split our design into two parts, one responsible for defining the chain of DSP operations and another as an executor of those operations. The executor receives a list of topologically ordered operations specified by the first part and executes them in the controlled real-time environment.

A second point deserving our attention is the data that flows between nodes. Audio hardware processes multiple audio samples at a time. This hardware stores the samples in a circular queue before sending them to the speakers. A common configuration is a buffer with a capacity of **2*periods** so that every time the audio hardware consumes **period** from it, a system interrupt calls the application to fill in **period** more. This is usually done via a callback registered with the OS's audio API (Linux case described in ALSA asynchronous playback [1]). Dimensioning these buffers is a balancing

[1] `https://alsa.opensrc.org/Asynchronous_Playback_(Howto)#The_Callback_Notification`

act. Large buffers will increase efficiency due to smaller interrupt overhead and better CPU cache locality. On the other hand, small buffers are required for low latencies and better application responsiveness. There is also an additional restriction for the FFT algorithm. It requires buffers to be a powers of 2, as discussed before in subsection 3.1.5 (FFT).

Brungart et al. (9) measured the effects of latency in the human ability to localize virtual objects and observed no perceivable detriments for latencies of up to 70ms. With this we can compute a buffer size that meets all criteria and we arrived at 1024 frames as derived below:

For a sample rate of 48kHz and a full buffer, with $2 * period$ samples we get an upper bound of:

$$samples = latency * frequency$$

$$samples = (0.070/2) * 48000$$

$$samples = 1680$$

The chosen value is then the power of two immediately below 1680, 1024. Which gives our system a best case latency of about 42ms.

4.2 Design

Our design, up to this point, is a loop over an array of DSP instructions executed in sequence. They operate on buffers with 1024 samples each and as the last step feeds a final buffer to audio hardware. This means we will have quite a few of those buffers going around carrying data from one node to the next. An option would be dynamically allocate them and pass their pointers around. But, even with correct ownership by reference counting, this does not satisfies our real time considerations as we will discuss in the 4.4. We will sidestep this issue by preallocating a block of memory capable of storing a large number of buffers, a technique known as a pool allocator.

This pretty much describes a register based virtual machine, in our case, only capable of executing specialized DSP operations. It receives a program with instructions, sources, listeners, and effects, and produces a buffer ready to be queued by the audio hardware.

4.3 implementation

We implemented the DSP-VM (section 4.3.1) described in the previous section and a demo application (section 4.3.2) that uses it to spatialize white

noise.

4.3.1 DSP-VM

The implementation was done in the C programming language starting from a DSP library from scratch. The library implements important functions that the VM will use as primitives (including the FFT algorithms). A different module used this library to implement the VM instructions (some are one-to-one mappings and some are complex operations, such as the HRTF). A simplified list of the available operations is shown in listing 4.1.

```
read_wav(float o[], int sourceid);
add (float o[], float x[], float y[]);
fft (float xr[], float xi[]);
ifft(float xr[], float xi[]);
hrtf(float h[2][2], int sourceid);
mul (float o[], float x[], float y[]);
mulk(float o[], float x[], float k);
```

Listing 4.1: Pseudo code of implemented DSP operations

The next step is to define the virtual machine instructions. We implement them as a union of structs (listing 4.2). This is a simple way to implement polymorphism in C. All instructions inherit the "any" instruction by defining it as their first member. In this approach, all instructions can be cast safely to any instruction during execution.

```
union t_snd_inst {
  struct t_snd_inst_any {
    uint16_t op;
  } as_any;

  struct t_snd_inst_clear {
    t_snd_inst_any _;
    uint16_t x;
  } as_clear;

  struct t_snd_inst_add {
    t_snd_inst_any _;
    uint16_t o, x, y;
  } as_add;

  struct t_snd_inst_hrtf {
    t_snd_inst_any _;
    uint16_t src, xr[2], xi[2];
  } as_hrtf;
  ...
};
```

Listing 4.2: Instruction set ("clear", "add" and "hrtf" examples)

Each instruction encodes its own parameters. For example, the "clear" instruction encodes which register to clear. In the case of add, it encodes both of its inputs and its output.

A pseudo code program that implements spatialization is presented in listing 4.3. In it, the encode prefix is used to fill in one of the structs defined above. It does not execute the instruction.

```
spatialize(program *p, uint16_t it) {
        // registers
```

```
        uint16_t x [2] = {0,1};
        uint16_t yl[2] = {2,3},
                 yr[2] = {4,5};
        uint16_t h[2][2] = {{6,7},{8,9}};

        encode_read_wav(p, x[0], it);
        encode_clear(x[1]);

        encode_fft (p, x[0], x[1]);
        encode_hrtf(p, h, it);

        encode_mul (p, yl[0], x[0], h[0][0]); // yl = x * hrtf_left  (real part)
        encode_mul (p, yl[1], x[1], h[0][1]); // yl = x * hrtf_left  (imag part)
        encode_mul (p, yr[0], x[0], h[1][0]); // yr = x * hrtf_right (real part)
        encode_mul (p, yr[1], x[1], h[1][1]); // yr = x * hrtf_right (imag part)

        encode_ifft(p, yr[0], yr[1]);
        encode_ifft(p, yl[0], yl[1]);
}
```

Listing 4.3: Pseudo code of the spatialization operation

An advantage of this design is that all instructions fit in the size of a t_snd_inst. We use this property to avoid the allocation of instruction individually. With this, we also avoid the need for an array of pointers to instructions and do directly an array of them.

A disadvantage is that every instruction occupies the size of the largest of them. Therefore, care must be taken to avoid enlarging the instructions too much.

A program is an array of instructions, constants and registers (listing 4.4). The program defines the sequence of DSP instructions to execute. Registers hold intermediate values passed between instructions and constants store the twiddle factors and the HRTF table. In this way, data can be shared between programs.

```
struct Source {
  vec3 pos;
  WAV  wav;
};
struct t_snd_prog {
  Instructions insts[];
  Source sources[];

  float constants[];
  float regs[];
};
```

Listing 4.4: Pseudo code of the DSP-VM Program definition

4.3.2 Demonstration

To ensure that our design was useful in practice, we implemented a demo application to show the spatialization in action. This demo spatializes a white noise according to the mouse's position. The movement of the mouse is linked to θ and ϕ and changes the HRTF accordingly (indicated by the light gray triangle in figure 4.1). A second view of the application Figure 4.2 shows the intermediate buffers used in the computation of the spatialization.

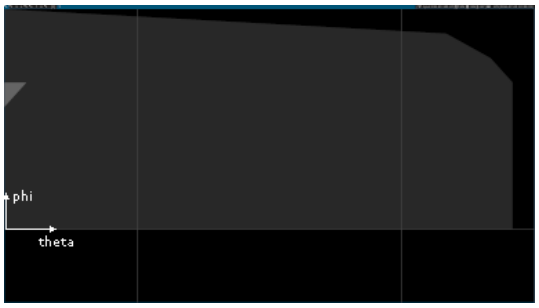


Figure 4.1: Interactive application with HRTF mouse selection (the light gray triangle selects the position of the sound source)

Figure 4.2: Interactive application with HRTF mouse selection (Buffer view). The various audio signals correspond to the output of each node of the spatialization pipeline (e.g. HRTF generates two pairs of signals in the complex plane - a total of 4 signals)

4.4 Real time considerations

We have timing restrictions based on the OS audio callback function. Failure to feed it in proper time will cause audio glitches and, depending on the situation, may damage audio equipments. To make sure we stay within time, we avoid operations with variable time. This includes operations like dynamic memory allocation, acquiring and releasing locks, algorithms with amortized cost and functions that take an unknown amount of time to execute. Ideally we should also operate on memory sequentially to avoid stalls. The design decision of separating the DSP-VM program "compilation" from execution helps in achieving these points (because these restrictions are only required by the execution part of the virtual machine, and it is unrestricted otherwise). A good read on the subject is the unpublished C++ guideline for the implementation of latency preserving libraries [2].

[2] https://docs.google.com/viewer?a=v&pid=forums&srcid=MTE5NTAwNjk0ODI0NDg0MTc0MjkBMTcwMDg5NzU5MzEwMzQ5ODQ4NzABNVhhNWtXMExCd0FKATAuMQFpc29jcHAub3JnAXYy&authuser=0

Those are guidelines and sometimes impossible to follow strictly. As an example, we break the amortized cost rule for triangle lookup. Common sense is also advised.

4.5 Results

The demo works and demonstrates the spatialization effect correctly and in real time. It took on average 70us for a single audio source to execute the complete spatialization pipeline on the prototype implementation (average time measurement over 1024 samples). Extrapolating this result to our latency window of 40ms, we can compute about 500 individual audio sources on a single CPU core.

5
Conclusion

We have successfully implemented the virtual machine described along the dissertation. It is capable of audio spatialization in real time and meets the real-time requirements we have discussed. We made sure that it has a reasonable design by implementing the demo along with it and showed that it can be used in practice.

```
Flat profile:
  %   cumulative   self              self     total
 time   seconds   seconds  calls  us/call  us/call  name
 48.34  0.28      0.28     2744   102.18   102.18  t_math_fft_rn
 18.99  0.39      0.11     2744    40.14   142.32  t_math_ffti_rn
 15.54  0.48      0.09                             t_math_fft_nr
 10.36  0.54      0.06                             t_snd_hrtf_op
  3.45  0.56      0.02     2744     7.30     7.30  t_snd_ola_ch.constprop.17
  3.45  0.58      0.02                             t_snd_mul_c2i_c1i_op
```

5.1
Future Works

The virtual machine was conceptually divided into two pieces (a compiler part and an execution part), but little attention was given to the first one. There is a vast body of literature in the subject of compilers, register allocation, code optimization and etc. An optimizing compiler might leverage the fact that various DSP operations could be rearranged and may show faster ways to achieve the same results.

The subject of audio culling was also not explored. It presents unique challenges when compared to graphics, due to the audio's larger ability to refract. Because of that, techniques such as depth buffer would not work as it does for graphics. However, global illumination techniques may work.

In the demo, we implemented a linear search over the HRTF triangles. We would like to implement the algorithm described in section 3.3.1 and measure its impact on the spatialization's time.

The speed of spatialization is heavily dependent on the floating point speed of the platform. An interesting avenue for future works is to re-implement

the DSP operations in such a way that we can leverage the single instruction multiple data (SIMD) CPU operations and make time comparisons for both. A third point of comparison would be to implement the DSP library as fixed point operations and also measure the impact.

Bibliography

[1] Lord Rayleigh OM. XII. On our perception of sound direction. The London, Edinburgh, and Dublin Philosophical Magazine and Journal of Science. 1907;13(74):214–232.

[2] Hartmann W, Rakerd B, Crawford Z. Transaural experiments and a revised duplex theory for the localization of low-frequency tones. The Journal of the Acoustical Society of America. 2016;139:968–85.

[3] Gardner B, Martin K. HRTF Measurements of a KEMAR Dummy-Head Microphone. MIT Media Lab Perceptual Computing; 1994.

[4] Rabiner LR, Gold B. Theory and application of digital signal processing. Prentice-Hall signal processing series. Prentice-Hall; 1975.

[5] Hahn J, Geigel J, Won Lee J, Gritz L, Takala T, Mishra S. An integrated approach to motion and sound. The Journal of Visualization and Computer Animation. 1995 04;6:109 – 123.

[6] de Souza Maffra SAR. Propagação de som em ambientes acústicos virtuais bidimensionais. 2003 07;.

[7] Raghuvanshi N, Snyder J. Parametric Wave Field Coding for Precomputed Sound Propagation. ACM Trans Graph. 2014 Jul;33(4):38:1–38:11. Available from: http://doi.acm.org/10.1145/2601097.2601184.

[8] Schissler C, A N, Mehra R. Efficient HRTF-based Spatial Audio for Area and Volumetric Sources. IEEE Transactions on Visualization and Computer Graphics. 2016 April;22(4):1356–1366.

[9] Brungart D, Simpson B, McKinley R, Kordik A, Dallman R, Ovenshire D. The Interaction Between Head-Tracker Latency, Source Duration, and Response Time in the Localization of Virtual Sound Sources. In: Proceedings of ICAD 04- Tenth Meeting of the INternational Conference on Auditory Display; 2004. p. 1–7.

[10] Oppenheim A, Willsky A, Nawab S. Signals and Systems. Prentice-Hall signal processing series. Prentice Hall; 1997.

[11] Chu E, George A. Inside the FFT Black Box: Serial and Parallel Fast Fourier Transform Algorithms. Computational Mathematics. Taylor & Francis; 1999.

[12] Kleinberg J, Tardos E. Algorithm Design. United states ed ed. Addison Wesley; 2005.