MASTER

Implementing a real-time dynamic auralization method in a virtual reality environment with quality assessment through listening tests

Lekx, S.T.

Award date:
2019

Implementing a real-time dynamic auralization method in a virtual reality environment

With quality assessment through listening tests

Eindhoven University of Technology

Name: S.T. Lekx
Student number: s115861
Department: Architecture Building and Planning
Unit: Building Physics and Services
First supervisor: dr.ir. M.C.J. Hornikx
Second supervisor: F. Georgiou, M.Phil
Third supervisor: ir. C.C.J.M. Hak
Date: 10/04/2019

Summary

Virtual reality has many use cases already, and utilization of virtual reality is increasing in built environment applications as well. The focus of this utilization is on the visual aspect of virtual reality. The 3D visualization options virtual reality bring, can assist architects in communications with their clients. When focusing on acoustics and virtual reality, auralizations can do the same for acousticians as a communication tool with their clients. An auralized sound will take the characteristics of the modeled space into account and will sound like it is played back in that room.

Creating these auralizations is done with a calculation method called convolution. An auralization is a convolution of a filter signal and an anechoic source signal. Convolution calculations can be done with several different methods depending on the situation and the desired result. In this thesis, the focus lies on a real-time dynamic auralization method. Depending on the size of the signals, real-time digital signal processing can require a different approach. With short filter signals, no partitioning of the filter signal is necessary, and real-time processing can be achieved using linear or circular convolution methods.

In this thesis the Universally Partitioned Over-lap Save (UPOLS) method is used to create real-time auralizations. This method requires the partitioning of the source and the filter signals in blocks of a given length. Processing speed and latency is dependent on the size of these blocks. If neither the source nor the receiver move or rotate during the auralization, the situation remains static. However, rotations are allowed in the model used in this thesis. The convolution calculation method must adapt the auralizations to the movement of the user of the model. This movement is tracked using an Oculus VR headset. Combining the visual and the acoustic aspect of the complete model is done on the Unity platform. This platform allows for the execution of code through C# classes and can combine this with a visual model. Adapting the dynamic UPOLS method to this platform to provide continuous audio playback was difficult for the writer of this thesis. The calculation method was expanded and adapted to use GPU and CPU processing by an external Building Acoustics partner.

A listening test methodology was developed to assess the quality of the dynamic auralizations. Three different binaural room impulse responses, from an anechoic room and two different churches, and three different sound source samples (a male speaking voice, a drum sample and a pink noise sample) were used to create a total of nine different scenarios. For each of these scenarios, participants of the listening test were asked to assess their quality based on three different attributes. These attributes were responsiveness, smoothness and externalization. Their responses were collected in VR, using a graphical user interface that would pop up during the listening test. Participants could interact with this interface with Oculus touch controller. There was no need to take off the VR headset during the listening test.

Results of the listening test were positive across all different scenarios. The lowest median score was found on the externalization scenarios in the anechoic virtual space. Highest median scores across all participants were found on the smoothness scenarios in the Maranatha church with a score of 6,9, while the Catharina church scenarios scored the overall highest with a median score of 6,7 across all attributes. Lower scores in anechoic scenarios were expected due to the lack of reverberation. No overall significant differences can be found between the scores across different scenarios however. Meaning that, the auralization method works well, but improvements can be made across the board. Improvements for the auralization method are available in the form of a non-uniform partitioned convolution algorithm among others. The procedure for the listening test can also be improved by increasing the familiarity of the participants with VR and the listening test produce, by either increasing the length of the test session, or by increasing the length of the samples within the test session, for instance.

Table of contents

List of figures

Chapter 1: Relevant theory and background

1.1 Introduction

In the recent years the usage of virtual reality in architectural acoustics has increased a lot. Its versatility allows for lots of different applications and tests to be done. Several virtual reality applications are being developed with architectural acoustics in mind [1]–[3]. The increased availability of virtual reality capable hardware facilitates this increase in usage even further [4]. Despite this increase in usage of VR in acoustics, the use of virtual reality in the build environment is this mainly focused on the visual aspect. The 3D visuals can assist the architect in relaying his/hers intentions to the clients and it can be a great communication tool [5].

On the contrary, the acoustic evaluation of an indoor environment is usually done through room acoustic measurements or noise measurements and is then classified using reverberation time, speech intelligibility, noise levels and other characteristics. However, these characteristics might not be familiar to the clients that do not have a background in acoustics. Virtual reality technology can assist in understanding this by providing a tool which simulates the acoustics of an indoor space. Within virtual space the auralizations can be combined with the visual aspects. This creates a complete immersive experience that gives potential clients more information and would improve communication between the different parties. This leads right into the goal of this thesis. Which is to find and implement a method that facilitates real-time auralizations, using binaural room impulse responses. This method will take the head rotations of the listener as input and adapt the auralization based on them in real-time.

Section 1.2 will further explain auralizations and the process of their creation. Section 1.3 and 1.4 will discuss the modeling steps of the sound propagation and the reproduction respectively. Section 1.5 presents the research goals and objectives.

1.2 Auralizations

Auralizations are a major part of this thesis, so more information on the subject is provided here. The term auralization can be explained in different ways. Savioja et al. [6] describe it as a subset of the virtual acoustic concept referring to the modeling and reproduction of sound fields. Some further delving into the process of creating auralizations reveals a section division that can be made. The two phases of creating auralizations, modeling and reproduction, are as mentioned. Within those phases, the modeling phase can be split up into three stages: source modeling, propagation modeling and receiver modeling. In Figure 1.1 below a visual representation of this process can be found. Following that the respective phases are explained further.

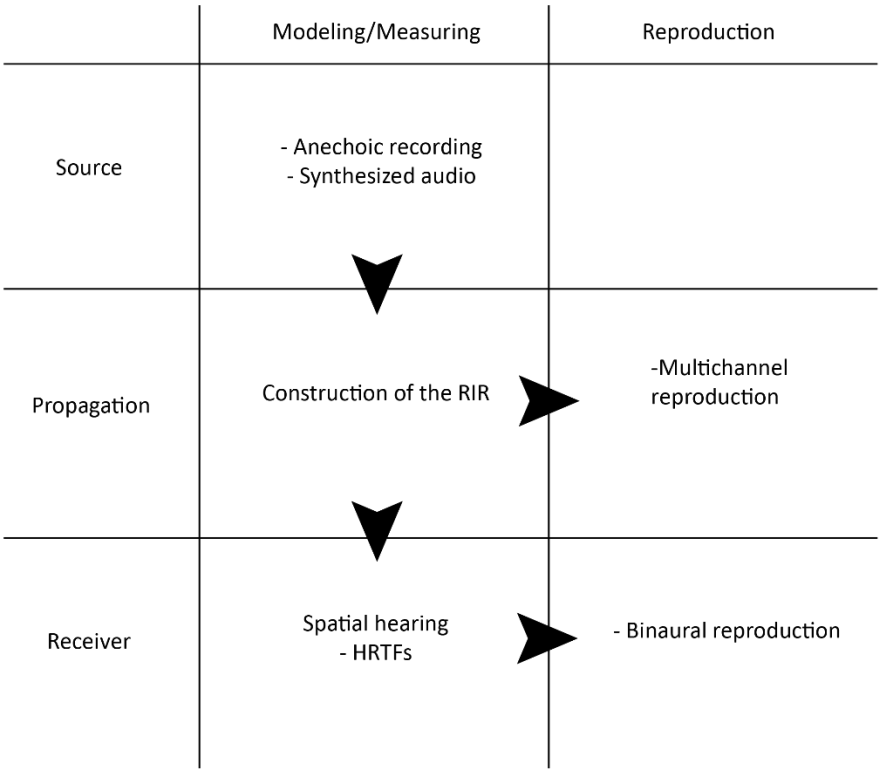

Figure 1.1 - The process of creating auralizations

The first step in the modeling phase is the source modeling. This source signal needs to be anechoic and can be either recorded in an anechoic room, or an anechoic signal can be modeled or generated. Any sound source reproduces sound with a specific directivity and this should be considered. Especially when a specific sound source is used or required for the auralization, like a musical instrument, the directivity of this source can be important.

Then, the propagation stage is considered. This step considers the effect of the sound propagation within the modeled space. The main goal of this stage is to compute the room impulse response (RIR) of the modelled space to use for auralization purposes. The RIR contains the acoustic characteristics of the room.

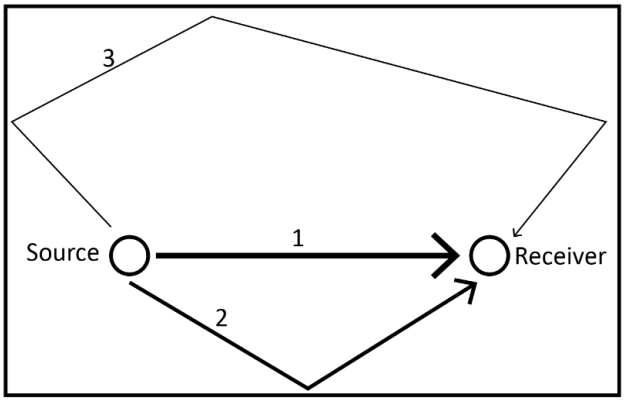

Figure 1.2 - Direct sound, early and late reflections (1,2 and 3)

It is constructed from different propagation paths the sound can take from the source to the receiver. These paths can be divided in three different categories: direct sound, early reflections and late

reflections. They are depicted as numbers 1, 2 and 3 respectively in the figure 1.2 above as well as referenced in figure 1.3.

When auralizations are made for an already existing location the RIR can be measured. However, these measurements are very time intensive and are only usable to create auralizations in the current situation. No further developments can be tested with those RIRs. Another way to model the RIR is to simulate them digitally. For these simulations, several methods are available, which are divided in three categories again. They are geometrical modelling, wave-based modeling, and energy-based modelling. In this thesis, only geometrical modelling techniques are used and they will be presented in the next section. More information on the other methods can be found in [7].

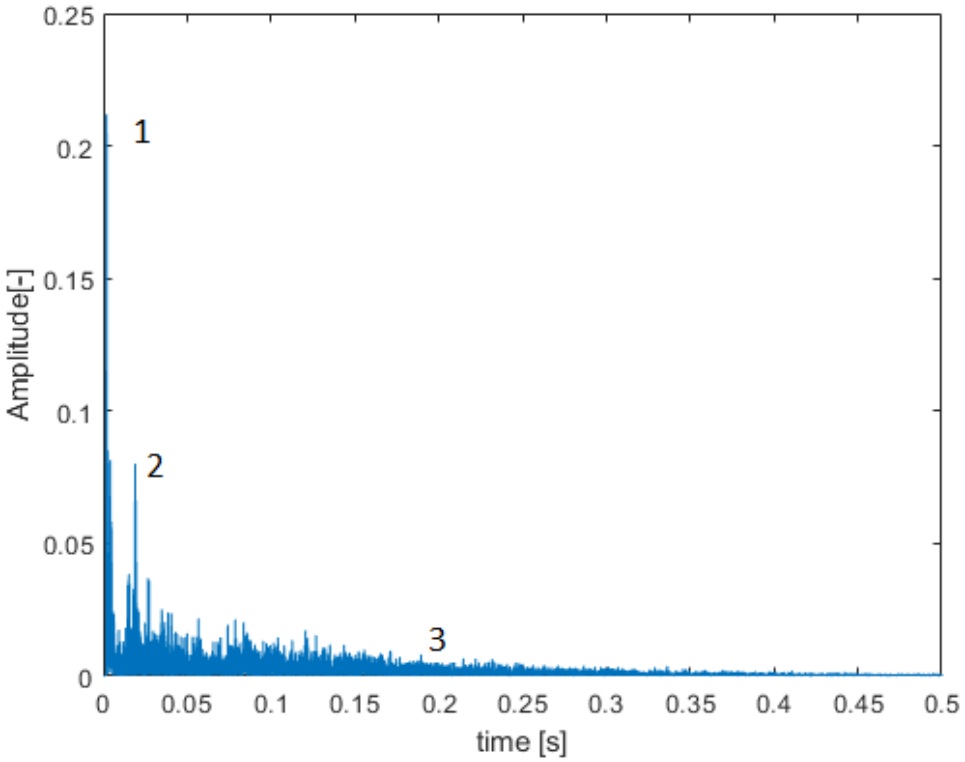

Figure 1.3 – Room impulse response

To generate, or measure, a RIR, a Dirac pulse is used to agitate the room. The impulse response is calculated from all paths the sounds travels from source to receiver. Reflections, absorption and diffraction from surfaces within the room are considered, and air attenuation as well. Together all the different paths combine to a complete room impulse response. An example can be found in the figure 1.3 above this paragraph. For this figure, the absolute impulse response has been used.

After this stage, reproduction can already be done through multichannel methods [6]. However, for this thesis, binaural reproduction is necessary, and therefore receiver modeling, the third step, must be followed as well. In the receiver modeling stage, the characteristics of the receiver are considered. Section 1.4 will detail this step further.

1.3 Geometrical modelling

Geometrical modelling is one of the main categories in sound propagation modeling methods. Within this method, the wave characteristics of sound are not used, but instead they are assumed to be rays and mostly follow the characteristics of light when considering its propagation. Diffraction and scattering are still taken into account however. A caveat for this method is that the wavelength of the sound is assumed to be small in comparison to the size of the room that is being modelled. The

advantages of this category of modeling methods is that they are not computationally expensive and are they are very fast in comparison to the wave-based modeling methods.

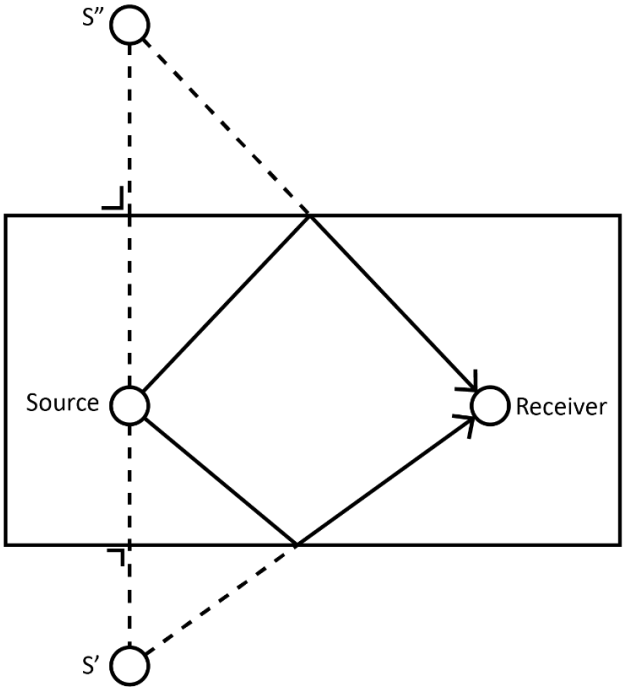

Figure 1.4 - Image source method example, reproduced after [8]

Figure 1.4 above is an example of the image source method, which is one of the most commonly used geometrical modeling methods within architectural acoustics [9]. Within the Image Source method, the reflections are modeled as image sources of the original source. The first order reflection is represented in the figure above. It is a source that is mirrored on the wall the reflection hits. For a second order reflection, a mirrored source of the mirrored source would have to be made, and so on for further reflection orders. The computational cost of the Image Source method increases drastically with an increasing order of reflections. To make sure the image sources are valid, visibility must be checked. If a line of sight between the image source and the original source cannot be drawn within the boundaries of the room, or the path is blocked by other walls than the wall the image source originates from, that respective image source will not be used. After all valid image sources are established, contribution from each image source is delayed according to its respective travel time, attenuated by distance (1/r) and air and wall absorption and then arranged in time based on their respective travel times [10].

To accommodate higher reflections orders without increasing the computational cost, a hybrid geometrical modelling method is used. This method combines the image source method with the ray tracing method. Such a hybrid method is used in ODEON [11] room acoustic modelling, which was used in this research. For this method, the ray tracing technique is used in two different ways. For the early reflections, the viability of the potential image sources is being checked using ray tracing. Then, on the transition from early reflections to late reflections, the rays are used as energy carriers and sources are generated at points where the rays collide with the room. More in depth information on this method can be found in [12].

1.4 Reproduction

The final stage of the auralization concerns the receiver modeling. As mentioned in section 1.2, this step is required for binaural reproduction of the auralizations. Within this stage, the head related transfer functions (HRTF) are used to model the propagation of the sound to the ears of the listener. HRTFs contain three types of information. These are:

- The interaural time delay (ITD) which signifies the difference in time of arrival of the sound between the two ears.
- The interaural level delay (ILD) which signifies the difference in loudness of the sound when it arrives at the ears.
- Specular reflections and absorption around the head, shoulders and pinnae of the subject.

HRTFs are unique for each person, each ear, and each angle of incidence of the sound [13]. A visual representation of a set of HRTFs can be found in Figure 1.5 below.

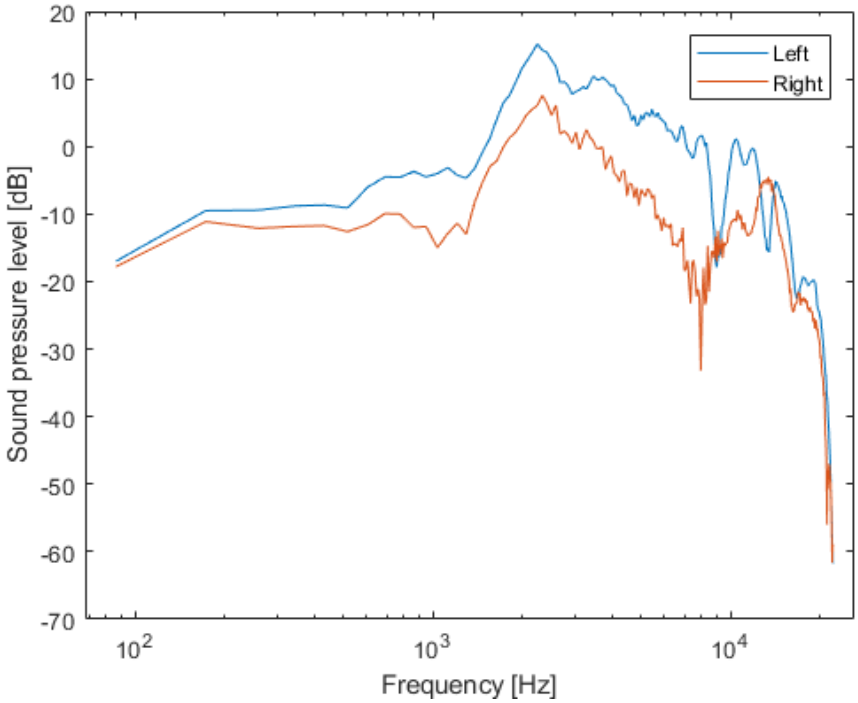

Figure 1.5 - Set of HRTFs from the KEMAR dummy head at 0° elevation and 30° azimuth

To completely model the propagation of the sound for a binaural auralization, the HRTF need to be incorporated inside the RIR. A room impulse response that contains the information of a HRTF is called a Binaural Room Impulse Response (BRIR). Since these are unique for each angle of incidence, they must be updated with movement of the head when head rotations are allowed in the virtual model. Allowing head rotations will enhance the ability of the user to localize and externalize the virtual sound sources [14][15]. More elaboration on the virtual reality aspect will be done in chapter 3.

With the receiver modeling complete, the final step in the creation of auralization is the reproduction. For this step, a mathematical operation called convolution is used. This operation applies the modelled filter to the sound source signal and generates the output auralization signal. When no movement from either the source or the receiver is involved in the model, the scenario is called static. In this case we can treat the auralization system as a linear time invariant (LTI) system that is fully described by a single function. For this auralization, this function is the BRIR. This function is of a finite length in this scenario.

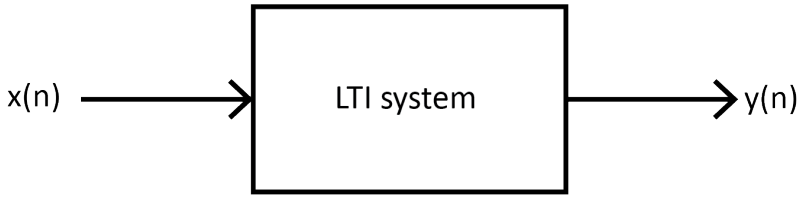

Figure 1.6 - LTI system

As seen in figure 1.6, this system has an input and an output. The input in this LTI system is the source signal, and the output is the auralization. This output is calculated with the following formula, which is only valid for a convolution of two strings of data of indefinite length:

$$y(n) = x(n) * h(n) = \sum_{k=-\infty}^{+\infty} x(k)h(n-k) \tag{1.1}$$

With y(n) is the output array of indefinite length, x(n) is the source array of indefinite length, h(n) is the filter array of indefinite length, k is the length of the filter array, n is the time indicator and $*$ represents the convolution operator. This type of convolution is called a continuous convolution. However, since this thesis will concern itself with signal of finite lengths, for the source signal as well as the filter signals, a slight modification needs to be made to this formula to make it valid for a convolution operation on signals of a discrete lengths.

$$y(n) = x(n) * h(n) = \sum_{k=max\{0,n\}}^{min\{n-M+1,N-1\}} x(n-k)h(k) \tag{1.2}$$

With the same parameters as defined in Eq. 1.1, but the limits of the summation are adjusted to fit the signals of finite length [16]. With the addition of head rotation into this scenario, the system is no longer static, and therefore no longer be described by as an LTI system, but as a dynamic time variant system. However, with such a case the system is still assumed to be a LTI system, but it is sampled in time to create a dynamic case [6].
Even though the signal lengths are known and finite for the project in this thesis, the convolution calculations will still take too long to implement them in a real-time environment. A partitioned convolution algorithm can be used to speed up the computation time and reduce latency. To further speed up the convolution operation, the Fourier transform will be used. Performing the convolution operation in the frequency domain requires less computation steps and is quicker [17]. This procedure and method will be further explained in chapter 2.

1.5 Research scope

The main goal of this thesis is to implement a low-latency auralization method suitable for real-time applications. To achieve this goal, several sub goals must be achieved in this work. These are:

1. Develop a real-time capable convolution calculation method, based on a partitioned convolution method.
2. Adapt this method to utilize head-tracking information and crossfading between different binaural room impulse responses
3. Integrate the adapted method with the respective visual information on the Unity platform. The calculation method must be executed through C# code.
4. The completed model then must be evaluated through a listening test within the virtual reality environment.

This listening test methodology will have to be designed and tested. Finally, the results of this listening test will have to be evaluated and discussed.

Chapter 2 will elaborate on different convolution calculation methods. Chapter 3 will follow this up by discussing the chosen convolution method and the implementation of this method in a real-time environment. Chapter 4 discusses the listening test used to assess the quality of this convolution method. Finally, chapter 5 presents the results of the listening test, the method and the project itself. Finalizing with the conclusions and recommendations for further work.

Chapter 2: Convolution methods

2.1 Introduction

The focus of this chapter is the convolution methods and partitioned variants of the convolution methods. Section 2.2 presents the circular convolution method and a calculation example of the time domain linear convolution method. Section 2.3 delves deeper into the Fourier transform and its role in speeding up convolution calculations. Section 2.4 presents the different partitioned convolution algorithm options. Section 2.5 present the partitioned convolution methods. Section 2.6 introduces the universally partitioned overlap-save algorithm that partitions both the source and the filter signal. Finally, section 2.7 explains the different possibilities the UPOLS method has when filters must be exchanged during a running convolution.

2.2 Linear and circular

Discrete convolution was introduced in section 1.4. To further illustrate this method, a calculation example of a basic convolution calculation is given in the Figure 2.1.

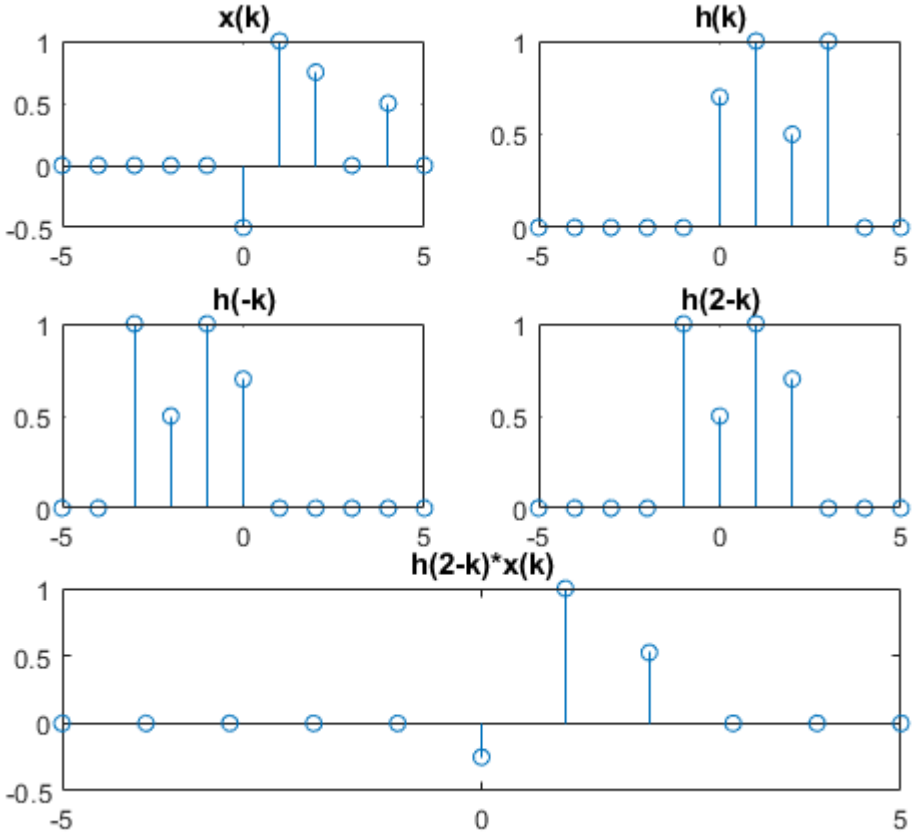

Figure 2.1 – Linear convolution calculation example

For this example, two randomly selected signals are generated. A source signal x(n) and a filter h(n). Both signals are plotted on a [-5,5] window in the top row of the figure. As can be deducted from the convolution Eq 1.1, the source signal is multiplied by a 'flipped and shifted' version of the filter: H(n-k). This is achieved in two steps. Step one is the flip of the signal, demonstrated by h(-k). Compared to the previous figure, the signal h has been flipped around the 0 axis. The shifting is indicated by the n in the h(n-k). The result of the shifting of h(-k) when n = 2 is chosen, is shown by h(2-k). After that, h(2-k) and the source signal x(k) are multiplied. The results of the multiplication of this flipped and shifted filter can be found in the bottom row of the figure. Now the results for y(n) with n = 2 is then a summation of all the values of the h(2-k)*x(k). For this example, this is: y(2) = 1,275. To complete the

convolution for this source signal and this filter, these calculations are done for all values of n that contain data. This type of convolution is called linear convolution.
Circular convolution is a different convolution method. This method assumes some periodicity in the signals used for the convolution. An example of a circular convolution method that produces a linear convolution result can be found in figure 2.2.

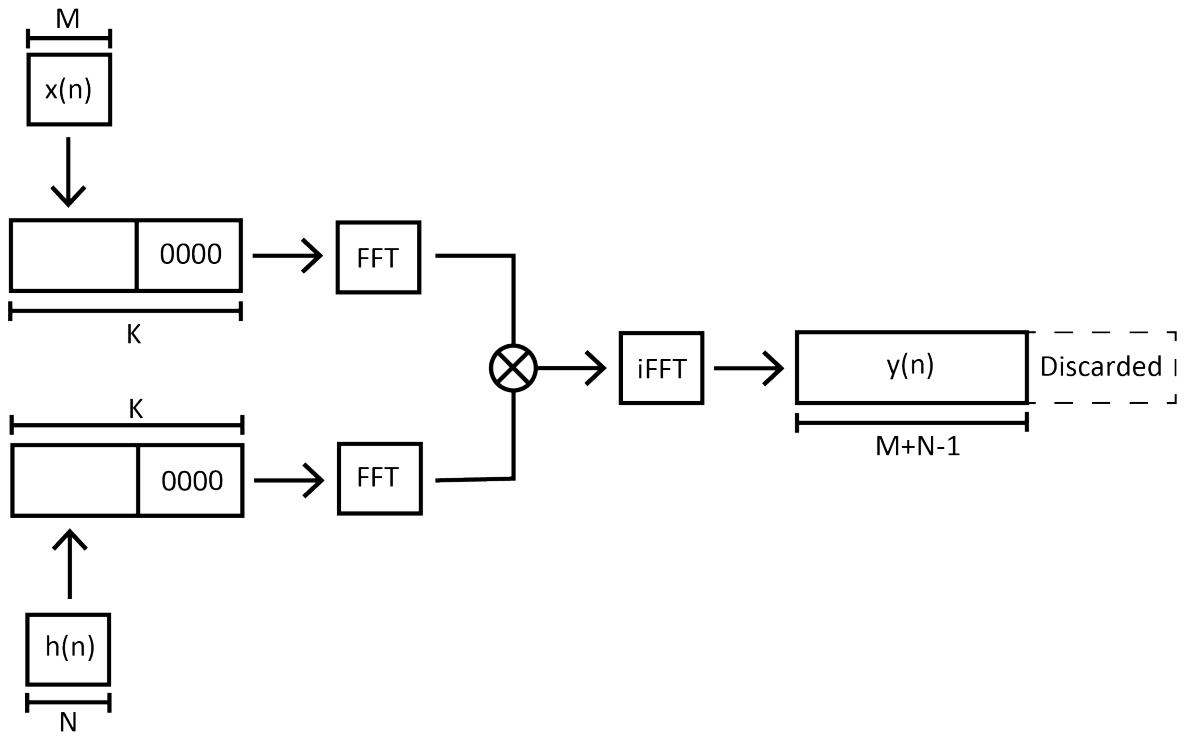

Figure 2.2 Circular convolution diagram

For this example, x(n) is the source signal of length M. h(n) is the filter signal with length N. The output signal y(n) is of length M+N-1. K is the transform size, with K ≥ M+N-1. Both the source and the filter signal are zero-padded to length K. After which they are transformed from the time domain to the frequency domain using a Fourier transform. More information on the Fourier transform is given in section 2.3. In the frequency domain, both signals are pairwise multiplied. The result of this multiplication is then transformed back to the time domain. Finally, to ensure the output signal is not time-aliased, any samples after a length of M+N-1 are discarded, resulting in the output signal y(n). As mentioned before in section 1.4, this method of performing the convolution operation is faster than the linear convolution method mentioned.

2.3 Fourier transform

Signals can be exchanged from the time domain to the frequency domain by using the Fourier transform. At any point, the LTI system can be described in either the time or frequency domain. The data in either domain can be related to the other through the use of a Fourier transform [18]. A representation of the LTI system including Fourier transform operations can be seen in Figure 2.3. An example of the same signal in time and frequency domain can be seen in figure 2.4.

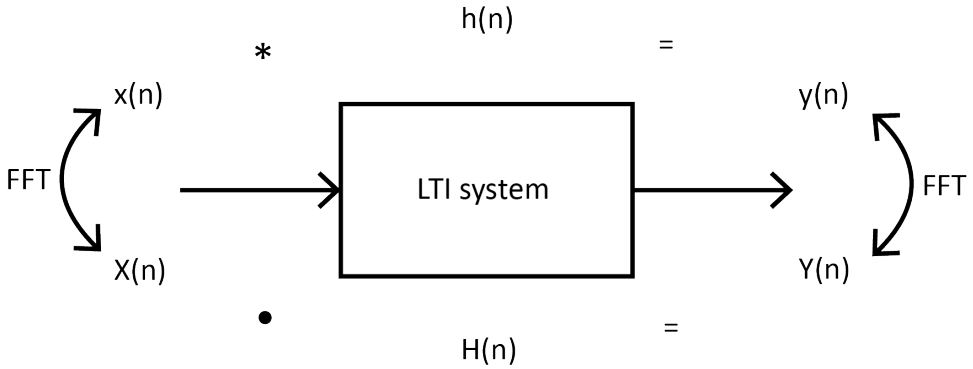

Figure 2.3 - Visualization of a LTI system with the inclusion of the Fourier transform

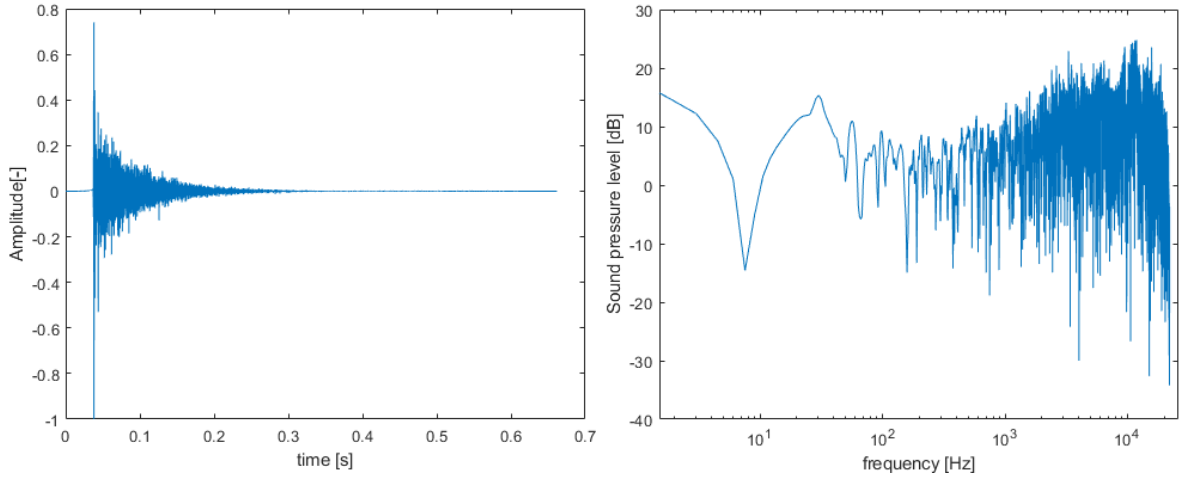

Figure 2.4 - [left] Time domain representation [right] Frequency domain representation of a room impulse response

The Discrete Fourier Transform (DFT) is a Fourier transform used for signals of defined length. This is used commonly in signal processing. An efficient variant of the DFT is the Fast Fourier Transform (FFT). First used as a fast convolution tool by Stockham [19]. Many different implementations of the FFT are available currently. Implementation of the FFT in coding can be done by using libraries that facilitate FFT convolution calculations easily. One of the faster libraries available is the Fastest Fourier Transform in the West (FFTW) [20].

The implementation of the Fourier transform in a convolution method was briefly mentioned in the circular convolution example in section 2.2. C. Muller [17] mentions that the combined complexity of doing a Fourier transform, an inverse Fourier transform and the multiplications in the frequency domain is less than the time domain convolution. This means that this method is faster than a time domain implementation. Using an efficient Fourier transform method like the FFT increases this efficiency. The FFT itself works most efficient when it is applied to discrete signals of with a length that is a power of 2. Of course, the FFT operation will be faster for shorter signals and longer for longer signals. When lower latency, or just shorter processing time is required, partitioned convolution is the next step.

2.4 Partitioned convolution

In a real-time application processing speed is very important. Audio processing cannot lag behind the playback of the results. If the audio processing, using a method mentioned in the previous section, cannot keep up with the play back speed due to the size of the signals, the convolution calculations are divided into parts or blocks to achieve real-time output. This is usually called a partitioned or segmented convolution. A partitioned convolution can be done in several different ways, which are detailed in Figure 2.5.

		Source signal		
		Not partitioned	Uniformly partitioned	Non-uniform partitioned
Filter signal	Not partitioned	X, X	UP, X	NUP, X
	Uniformly partitioned	X, UP	UP, UP	NUP, UP
	Non-uniform partitioned	X, NUP	UP, NUP	NUP, NUP

Figure 2.5 - Partitioned convolution possibilities, reproduced after [16]. *X = non-partitioned. UP = Uniformly partitioned. NUP = Non-uniformly partitioned.*

The figure describes nine possibilities in partitioned convolutions. Partitioning of the source and the filter signals are both available options. Partitioning of the signals can be done uniformly, or non-uniformly. Uniform partitioning indicates that the blocks the signals are divided into are all the same size. Non-uniform partitioning methods use blocks that are not the same size. When working with a short filter, partitioning of the filter might not be necessary to achieve a real-time capable convolution method. On the combinations presented in Figure 2.5, only the orange outlined combinations are real-time capable. A uniform partitioning of the source signal is suitable for a steady stream of audio data. The chosen partitioning method of the filter signal depends on the required latency of the system and on the size of the filter. With filter lengths of 20 – 30 times the partition block size, the universal partition method is faster than the non-uniform method. After that, depending on the partition size and the filter length, the non-uniform method will be faster [16].

For the acoustic VR goal that is set in this thesis, a uniform partitioned convolution is used. The increased complexity of implementing a non-uniform partitioned method [21] does not outweigh the benefits at this time. For the scenarios that are going to be used in this project, the uniform partitioned convolution should be sufficient, because of the relatively short filters. It will be easier to set-up on the different necessary software platforms as well. In the following sections. The UP,X method is discussed first, followed by the UP,UP method.

2.5 Uniform partitioned source signal convolution

For this uniform partitioned convolution, the filter signal is not partitioned, and the source signal is uniformly partitioned. B is the partition size parameter. Figure 2.6 depicts a uniform partition algorithm, with the Overlap Add method included.

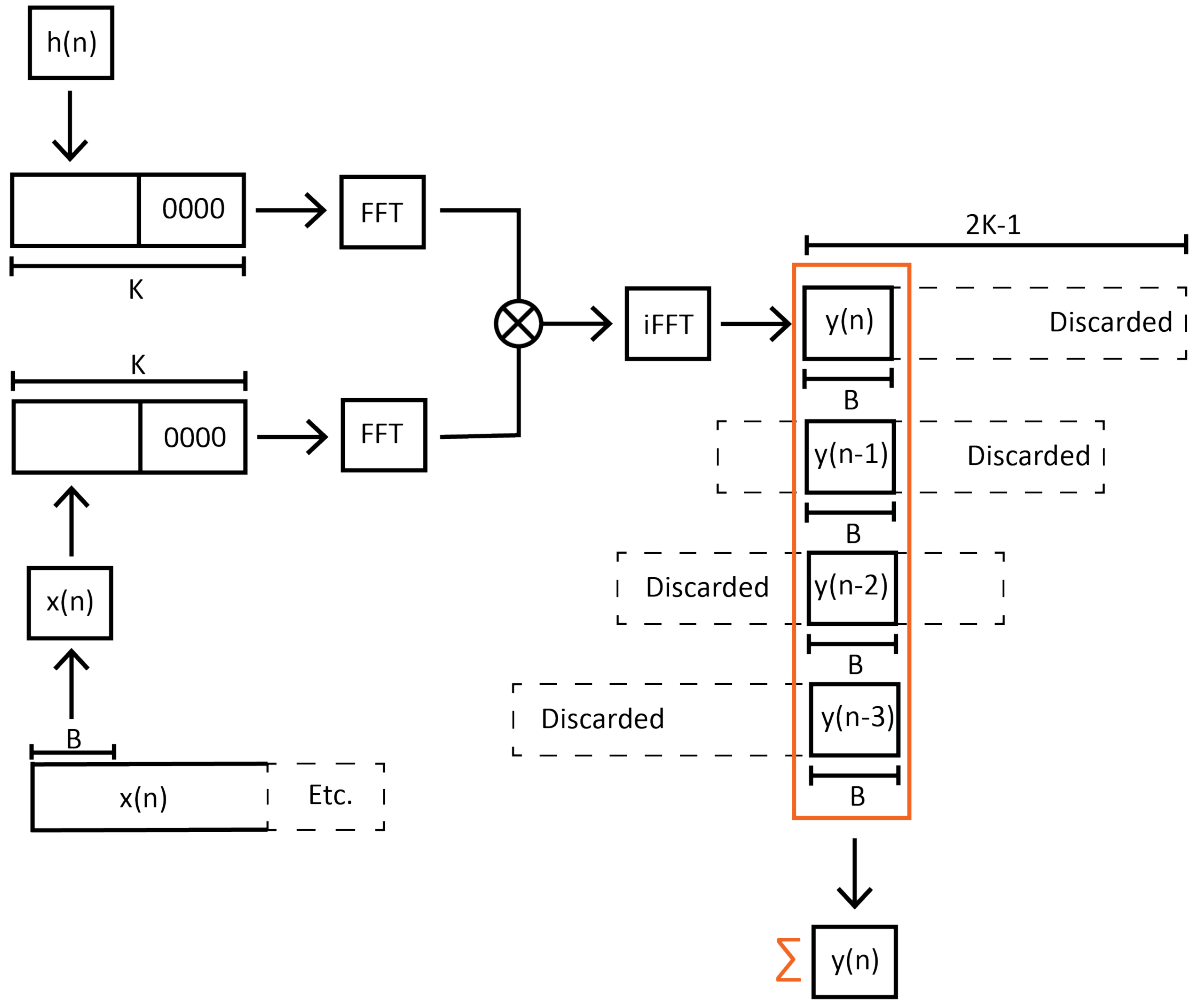

Figure 2.6 - Convolution utilizing the Overlap-Add method

In the depicted example in figure 2.6 x(n) is the source signal of unknown length. The filter signal h(n) is of known length N. The block size (partition size) is length B. Transform buffer size is K. K = 4B in this method as an example. The output signal is y(n). Parameter n is the timestep indicator. The partitioned convolution is more complicated than the regular time domain method explained at the start of this chapter. The example in figure 2.6 might be an example of a running convolution using a short filter. It goes through the following steps:

1. A length B size block of the input signal is zero-padded to length K.
2. The length N size filter will be zero-padded to length K.
3. Both zero-padded signal (blocks) are transformed to the frequency domain using the FFT.
4. Both frequency domain signals are pair-wisely multiplied.
5. The result of that multiplication is transformed back to the time domain using the inverse FFT.
6. The output of this step is a block of length 2K-1.

A problem occurs with this method since the input block is of size B, and the output block is of size 2K-1. A discrepancy in data length is not correct, since no extra sample should be created in the

convolution operation. This issue occurs because of the partitioned source signal. To resolve this, the Overlap-Add and the Overlap-Save methods are available.

For the Overlap-Add method (depicted in Figure 2.6), the size of B and the size of the transform buffer K are important. For a correct output result of size B, the results of multiple convolution steps are added up. For a single size B output block, the results of $\frac{K}{B}$(transform size over partition block size, in this example four) previous convolutions are necessary. The results of the first block B (of total K) of the N^{th} convolution is added up to the second B size block of the $(N-1)^{th}$ convolution and added up to the third B size block of the $(N-2)^{th}$ convolution and added with the 4 (and final) B size block of the $(N-3)^{th}$ convolution. This means that, using the Overlap-Add method, the results of several previous convolution need to be buffered for a correct result. Adding these results at the end also increases the time it takes for the convolution algorithm to complete.

The Overlap-Save method works a little different compared to the Overlap-Add method. Instead of using a buffer at the end of the algorithm, this method implements a way of buffering at the start of the algorithm, using the source signal. An example of a convolution operation with the Overlap-Save method included is depicted in Figure 2.7.

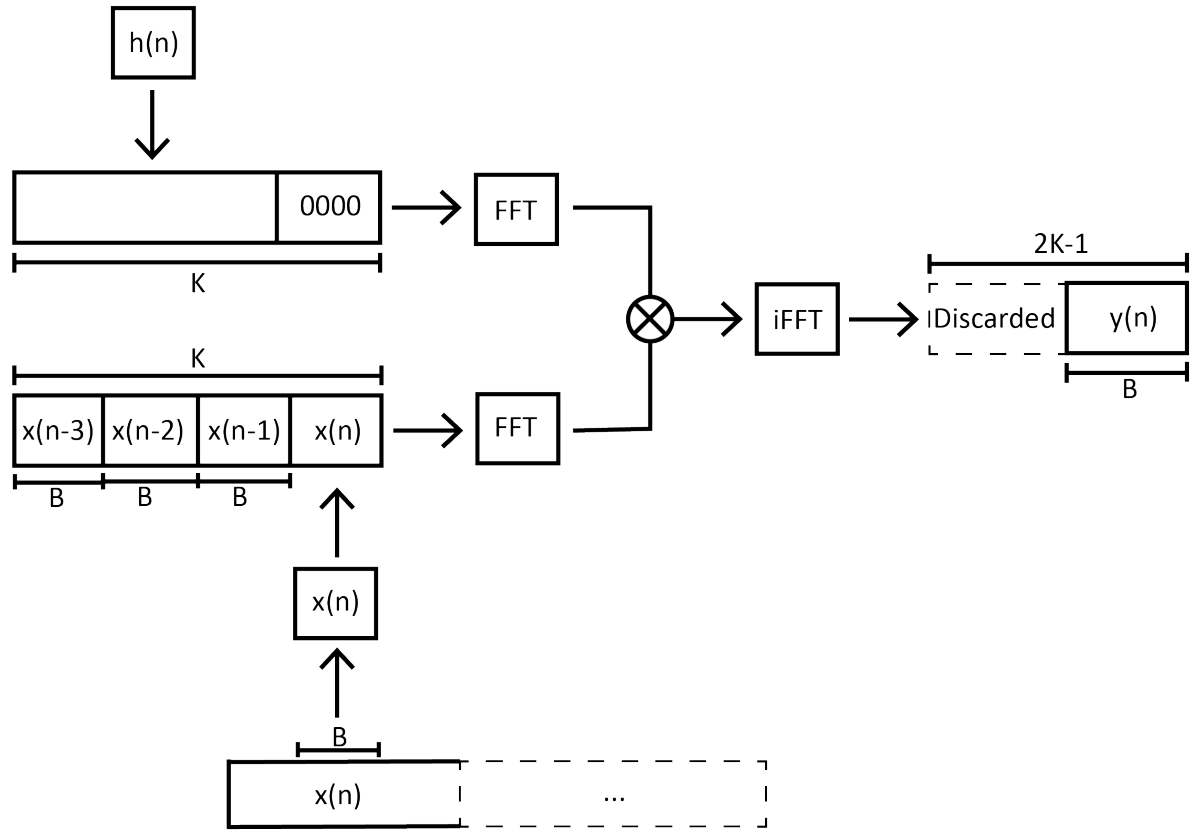

Figure 2.7 - Convolution utilizing the Overlap-Save method

Parameters used in Figure 2.7 are the same as for the previous Overlap-Add example in Figure 2.6. The convolution algorithm with the use of the Overlap Save method lies somewhere between the unpartitioned method and the previously discussed set up for the Overlap Add method. The source signal is still the only signal that is being partitioned, and still in blocks of size B as well. However, this time, the transform size K must be a multiple of B and larger then N (the filter length). The transform block K for the source signal is not filled with a single block of signal, but instead it acts as a sliding buffer that is always filled with blocks of the source signal. For this example, the transform size K equals 4B. This means that for the convolution of the n^{th} convolution the input buffer is filled with the blocks (n-3), (n-2), (n-1) and n (all of size B). The filter is then zero padded to size K as well, and after

that, the FFT enabled convolution works the same as the previous ones. Now for the output, this convolutions algorithm will output a result of length 2K-1. Of this, the first K-B blocks must be discarded to maintain a B size block of data, that is conveniently the correct output as well. No extra additions are necessary; Just the correct block of data needs to be saved. Because of that, the Overlap-Save method is usually preferred over the Overlap-Add method [22].

Either of the above presented methods can facilitate a real-time running convolution. However, there are some limitations to them. The filter lengths N must be short to maintain an algorithm that is quick enough. Since the transform size K must conform to the filter length, the latency is determined by the filter length. If the filter used for the convolution is longer or the required latency of the system is even shorter, partitioning of the filter signal is necessary. When both the filter and the source signal are partitioned, the transform size K can be reduced in size to accommodate for a shorter latency. This method would fall in the UP, UP category mentioned in Figure 2.5. It is further explained in the following section.

2.6 Uniformly Partitioned Overlap Save Method

Combining the Overlap-Save (OLS) method with a uniformly partitioned source and filter signal results in the uniformly partitioned overlap save method (UPOLS) [16]. The increase in speed gained by partitioning the input signal are lost when the filter size is way bigger than the block size. Therefore, this method is used further in the project to fuel a real-time running convolution in a VR space. Due to the low latency requirements of this running convolution, partitioning of the filter is necessary. An example of the UPOLS method is depicted in Figure 2.8.

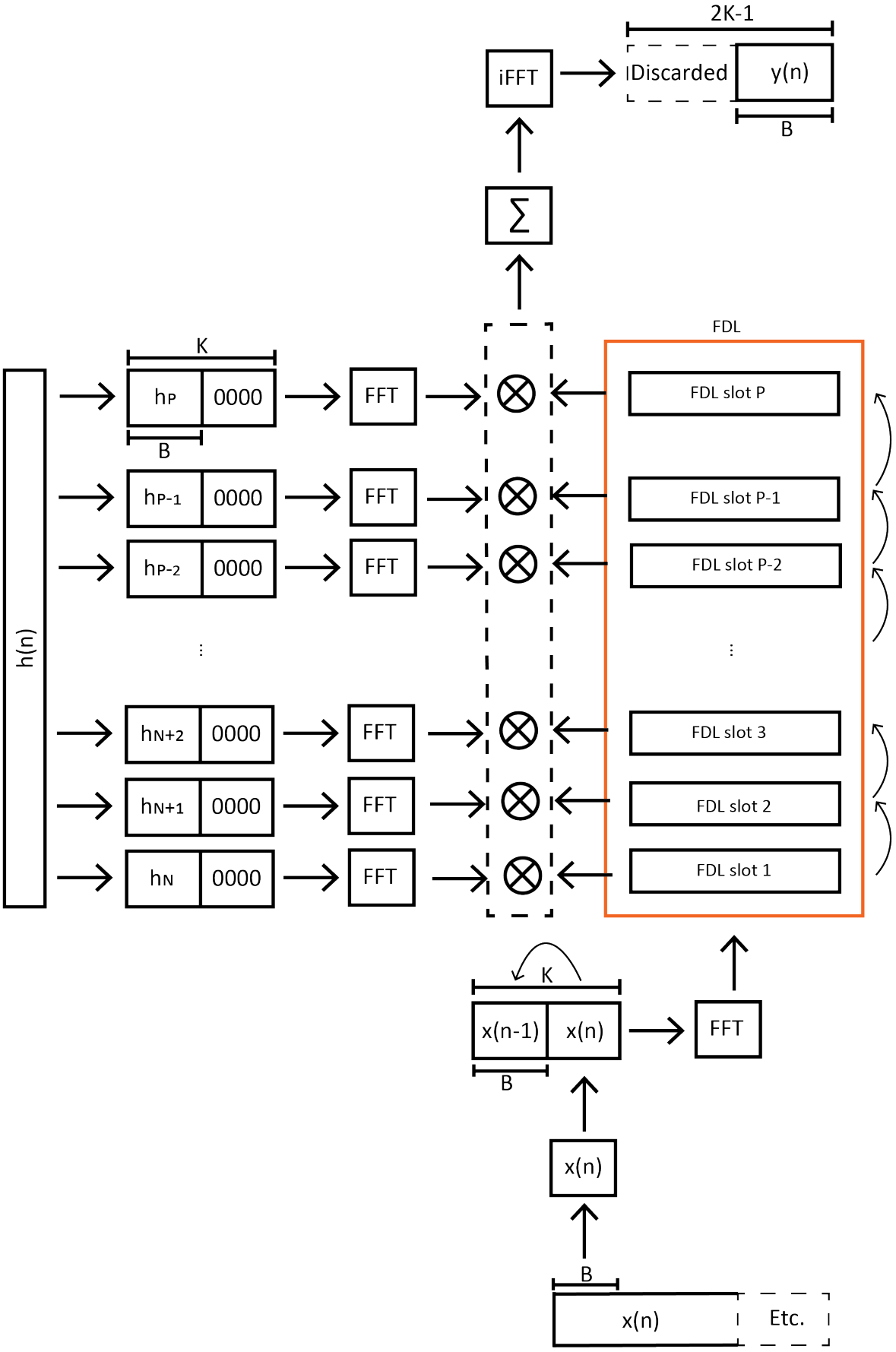

Figure 2.8 - UPOLS example diagram

This convolution uses the same method of input signal buffer as the Overlap-Save convolution. The transform buffer for the source signal always contains two blocks of the input signal. The block of source signal data from the previous convolution step, as well as the block of data for the current convolution step. The size of the transform buffer K is always equal to 2*B. The filter signal requires some extra attention at this point. In all previous editions of convolution that were mentioned in this thesis, the filter was unpartitioned. This changes with this iteration of the convolution algorithm. To assist this step, a new parameter needs to be introduced: P. This is the number of blocks that make up the filter signal in total. For a filter with the length N, and a block size B, the amount of filter blocks the filter signal will be divided into is parameter P = N/B (rounded up to the nearest integer to prevent half blocks).

The partitioning of the filter leads to a new interesting problem. The source signal partition cannot simple be multiplied (after being transformed to the frequency domain) with the transformed filter block. The input signal block needs to be processed by all filter partitions, but the convolution algorithm must output a result after just one run as well. This requires the introduction of a frequency domain delay line (FDL). This is a matrix that will store the transformed input blocks and process them throughout the entire filter. For each step of the convolution algorithm, the FDL will move its content a step further as well. A representation of this step can be found in figure 2.9.

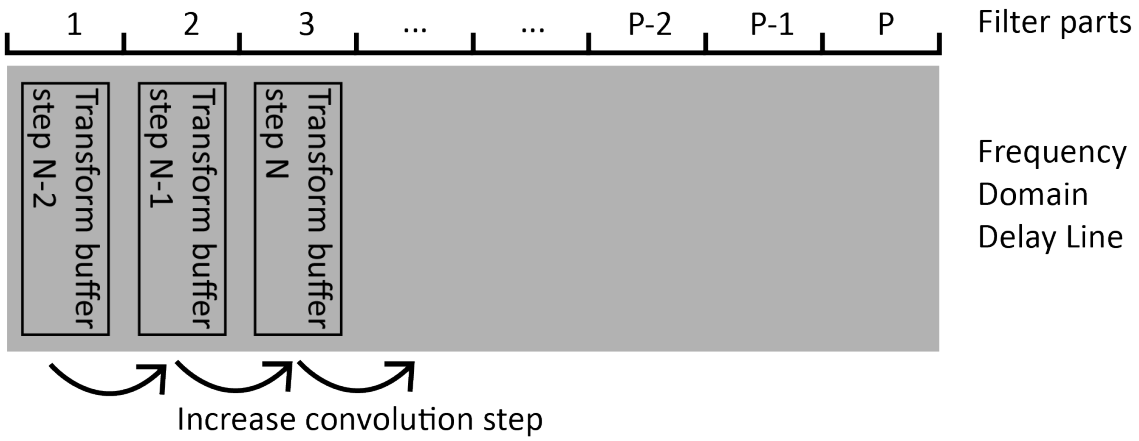

Figure 2.9 - Frequency Domain Delay Line (FDL) representation

This means that the pair-wise multiplication happens not only for the block that is currently being processed and its respective filter partition, but these multiplications are simultaneously done for the previous input blocks that are stored in the FDL. Once an input block has moved throughout the entire FDL and has been affected by every filter part, it is discarded. This means that for P number of steps of the convolution, an input block must be stored. To generate the output, the outcome of the multiplications of all the FDL contents and their respective filter blocks is summed in a so-called frequency domain accumulation buffer. From this buffer, the inverse FFT will be taken, resulting in a time domain result of length K. As we were using an input block of size B (=0.5*K), the first block of data must be discarded, and the results is a convolved block of data.

Concretely, this means that for every step of a UPOLS convolution, several calculations are made as can be seen in Figure 2.8. As a pre-processing step, the filter of length N is partitioned in P blocks of size B. All these individual blocks are then zero padded to size K. Then, all these zero padded blocks are transformed to the frequency domain using FFT.
On the source signal input side, the data from that right block are moved into the left block. Then the right block of the K size buffer is filled with a new B size block of input data. After that, the input block is transformed to the frequency domain using FFT. Then all previous data in the FDL is moved one spot. The newly transformed data is moved to the first slot of the FDL. The transformed filter blocks are pair-wisely multiplied with their respective data in the FDL. The results of all these multiplications

are stored in the frequency domain accumulation buffer. This buffer is then transformed back into the time domain. Finally, the left block is discarded, and the right block is the output block corresponding to the input block from the start of the convolution step.

2.7 Filter exchange

The UPOLS method in Figure 2.8 does not consider dynamic filter exchanges. Dynamic filtering is crucial for the simulation of dynamic VR environments with moving sources and moving receivers. When for example the listener is moving towards the source, the transfer function between the source and the receiver changes, thus, the convolution algorithm needs to update the filter in real-time to simulate this change. Moreover, this filter update needs to be implemented in a smooth way, so it does not sound unnatural. In this project, the listeners were only allowed to rotate their head within the virtual environment (the locations of the source and the receiver remained the same, so did the acoustic environment), thus, the discussions on filter exchange strategies in this section are focused on the update of the different BRIRs based on the angle between the source and the receiver.

A fast way to compute a running convolution is available in the form of the UPOLs method. However, this is limited to a static scenario. To create a realistic user experience, the movement of the user in the VR space should be reflected in the audio experience. Combined with the fact that the filters used are BRIRs, and they are different for each angle of incidence, the filters need to be updated for any degree of movement the user makes in VR. There are a couple of problems that arise with this. The exchange of the filter will have an effect of the results of the convolution algorithm. The discontinuity that is introduced in the output signal will also have to be smoothed to prevent audible artifacts.

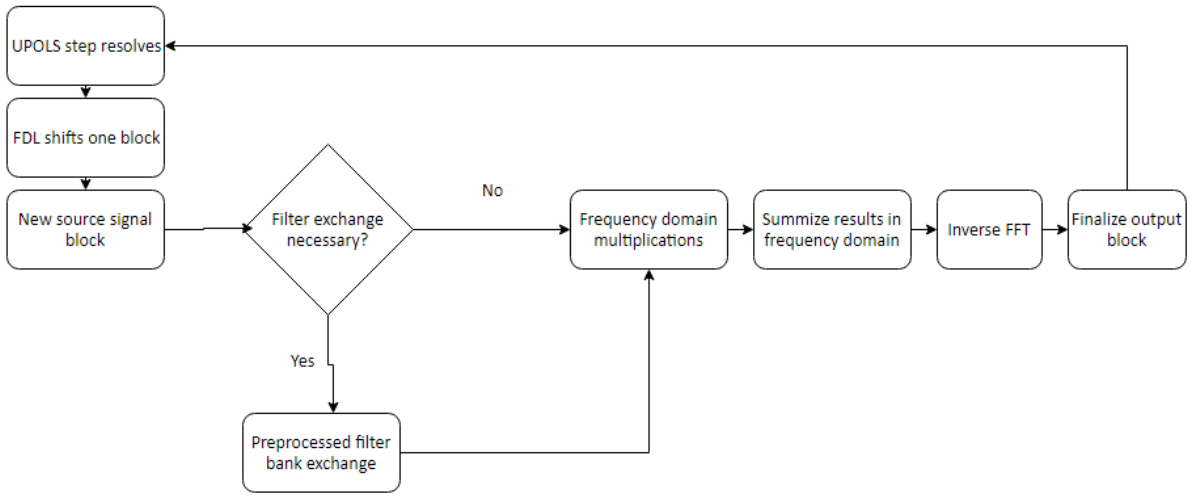

Figure 2.10 - Filter exchange steps

Figure 2.10 depicts the steps in a UPOLS convolution step. The content of the FDL will remain unchanged, but its corresponding filter blocks will have to be updated. With a well-designed convolution algorithm, the amount of filter blocks will be the same as the filter it is replacing. It this is not the case, extra care must be taken to ensure the proper processing of the FDL. The preprocessing steps will have to be done in real-time, or all filters must be pre-processed and then the transformed blocks can be swapped over with their respective filter blocks.

Now, for the output of the convolution, some more things need to be taken into consideration. Just a direct switching between the different BRIRs may result in a not very smooth result. The sound might sound choppy or other artifacts might occur. The different BRIRs can be so close together that the difference between them should not be heard by the users when a switch occurs between them. This angle is called the Minimal Audible Angle (MAA) [23] and is the smallest angular separation that

humans can detect between two sound sources. This MAA is depending on the type of sound source, and the location of the sound source. For a sound source in front of the user the MAA can be as small as 1° [23], while on the horizontal plane the MAA is between 4° and 5°. Even when the model complies with this MAA, the switching between the different BRIRs can still present audible artifacts like clicks and comb filtering effects [24]. This is merely due to the switching operation itself and the discontinuity it brings with it. To contradict these artifacts, crossfading between the result before the switch and after the switch can be a solution. The other solution might be to interpolate between the different HRTFs and thereby reduce the switching artifacts. Several interpolation options are discussed in a paper by Carty and Lazzarini [25].

In crossfading, another choice is available. The crossfading can be done in the time or in the frequency domain. Inherently, crossfading has some downsides, because for a crossfade, two output results are necessary. This means that for an accurate result during a filter switch, two convolutions must run.

Crossfading in the time domain is the simpler method of implementing the crossfade. The output of the convolution of the old source and the output of the convolution of the new source are both required, and then crossfaded. The crossfade window can be a full block, or less than that, but it must be the same for either signals. During this crossfade window, both signals are weighted with an envelope function. These functions must adhere to:

$$f^2(t) + g^2(t) = 1\,. \tag{2.1}$$

This means that the total amplitude of the signal is not changed when the crossfade occurs. Several different envelop function follow this rule. The differences between several of them are researched in a paper by Kudo et al. [15]. Of these methods the constant power cosine windows was deemed to be the best performing. For this method, the following formula describes the weighting factors:

$$f(t) = cos(\pi t) \tag{2.2}$$
$$g(t) = sin(\pi t) \tag{2.3}$$

Where t is the crossfade window and the fade in window is described by f(t) and the fade out windows is described by g(t). The resulting data exchange is visualized in Figure 2.11.

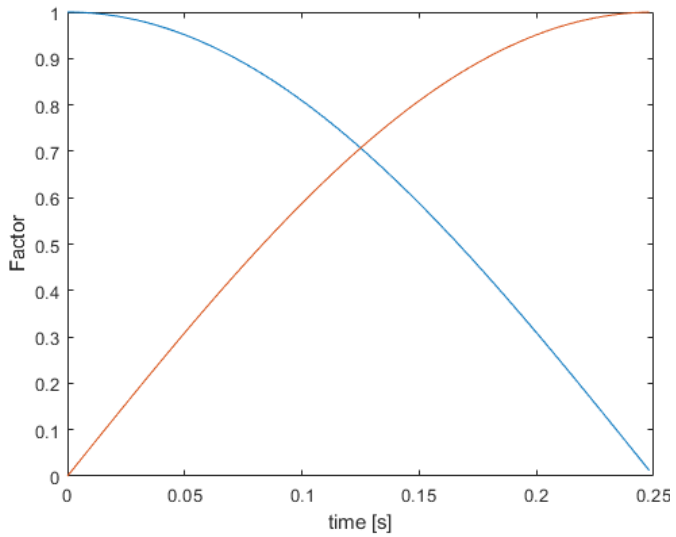

Figure 2.11 - Time domain crossfade exchange

Crossfading in the frequency domain must be implemented as an extra step in the convolution algorithm. Wefers [16] has found this method to be 20% - 30% faster than an crossfading implementation in the time domain. The crossfade method in the frequency domain is shown in Figure 2.12.

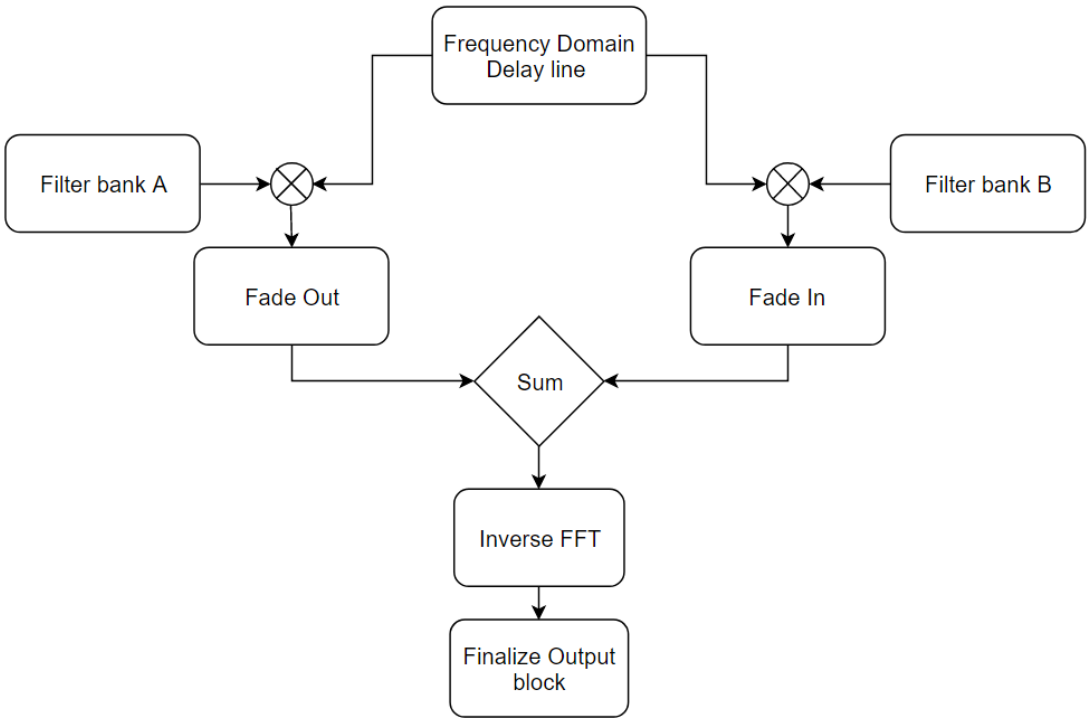

Figure 2.12 - Frequency domain crossfading flowchart

The crossfade method in the frequency domain done by convolving the same FDL content with the old and the new pre-processed filters. However, instead of crossfading the results of both convolutions in the time domain, the multiplications are faded in/out and summed, before it is transformed back to the time domain. Similar fade in and fade out signals are used as depicted in Eq 2.3 and 2.4. However, the crossfade is done in the frequency domain by applying a circular convolution method to the convolved product of the UPOLS method with the transformed fade in and out signals using DFT. Represented in formula form in Eq. 2.5 (time domain crossfade) and 2.6 (frequency domain crossfade) [16].

$$y(n) = y_0(n) \cdot f_{out}(n) + y_1(n) \cdot f_{in}(n) \quad (2.4)$$

$$Y(k) = Y_0(k) * F_{out}(k) + Y_1(k) * F_{in}(k) \quad (2.5)$$

To conclude, the UPOLS method will be used in the modelling for the auralizations in the VR model. To assist with the potential problems that occur with the exchange of filters, a frequency domain crossfading method will be applied. The increase in performance of the frequency domain crossfading method makes it the preferable choice.

Chapter 3: Implementation

3.1 Method outline

The goal of this chapter is to apply the theory from the previous chapter to a real scenario. The use of a virtual reality headset (or a head mounted display) is combined with auralizations in a virtual environment that adapts itself to the user. The adaptations in audio are realized by the real-time adjustments of the auralizations played in the model. The adaptations are dependent on movement of the user in the model, and their orientations towards a virtual sound source. This orientation will be tracked by the headset the user is wearing, in real-time as well.

Initially, the virtual reality (VR) model consist of a 3D model of a relatively simple space. Within this space, a single sound source and a single receiver is placed. The receiver position equals the position of the user in the VR model. Using generated BRIRs from the same 3D model, an auralization is calculated that adapts with the rotations of the user relative to the sound source. To reduce complexity of the model, this rotation is only be measured in the horizontal plane. Rotations in other planes are allowed, but do not affect the sound field. To achieve a certain level of fidelity, in a radial grid of 5° degrees from the receiver, the respective BRIRs will be generated. This leaves a set of 72 filters to be used in the running convolution algorithm.

To create this virtual environment, a suite of software is necessary. Within this suite, each program has a distinct role and acts in a distinct order. A clear overview with some extra information is shown in the figure below. After this figure, each step of the process, and its associated piece(s) of software, will be explained further.

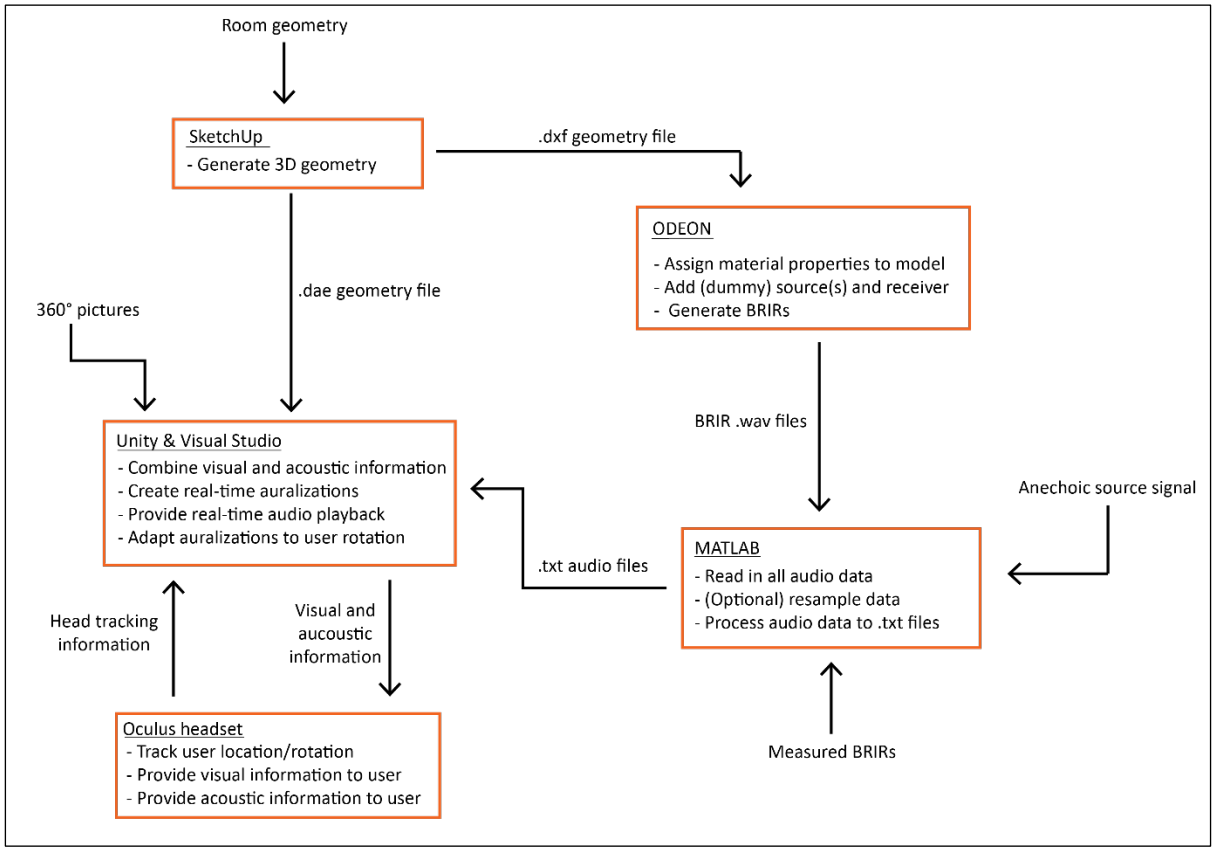

Figure 3.1 – Software framework overview

3.2 Room acoustic modeling

3.2.1 Geometry

The base of the model requires a space in which everything takes place. To create this, SketchUp is chosen. It can export a lot of different file extensions and is compatible with the other pieces of software that use the 3D model, for instance Unity and ODEON. The target of this model is to be relatively simple and small, to prevent a very long impulse response. To ensure the VR experience is not horrible, the model is dressed up with some furniture, and all the surfaces get a relevant texture. The model's dimensions are 3.0m x 5.0m x 2.5m.
To make sure the geometry can be imported correctly into ODEON, the model must be exported as an .dxf file. Please note that the orientation of the x, y, and z-axes are important here. This information is coupled to the model and is carried over to ODEON unless specified otherwise.
For the interaction between Unity and SketchUp, and export to a .dae file is necessary, as prefaced in the software framework in Figure 3.1. This makes sure that the textures and other relevant visual information is transferred with the model. A visual of the model can be found in figure 3.2.

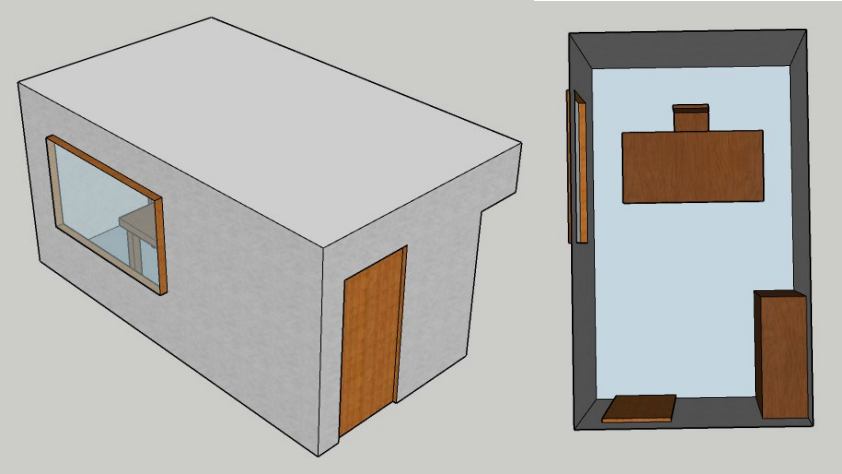

Figure 3.2 - SketchUp 3D geometry [left] Top down view [right]

3.2.2 BRIR generation

After the geometry has been exported from SketchUp, it can be used as a geometrical model in the ODEON. Selecting the correct scale is important here. The standard 'mm' unit of SketchUp must be also correctly adapted to mm in ODEON. Then, to make sure the program can calculate the impulse response of the room, every surface must be assigned a material. ODEON contains a large material library. Their information and absorption coefficients of the material selected in the ODEON model can be found in Table 3.1.

Table 3.1 - ODEON Material characteristics

		Absorption coefficient per octave band							
Material	**Code**	**64**	**125**	**250**	**500**	**1000**	**2000**	**4000**	**8000**
Walls	4049	0,26	0,26	0,2	0,1	0,07	0,04	0,07	0,07
Furniture	3068	0,26	0,26	0,2	0,1	0,07	0,04	0,07	0,07
Glazing	10003	0,10	0,10	0,07	0,05	0,03	0,02	0,02	0,02
Floor	7007	0,09	0,09	0,08	0,21	0,26	0,27	0,37	0,37
Door	10007	0,14	0,14	0,10	0,06	0,08	0,10	0,10	0,10
Ceiling	91	0,45	0,45	0,70	0,80	0,80	0,45	0,45	0,045

Following that, the room is filled with the needed sources and receiver. In this case, only one source and one receiver were added to the model. Their respective locations can be found in the following figure:

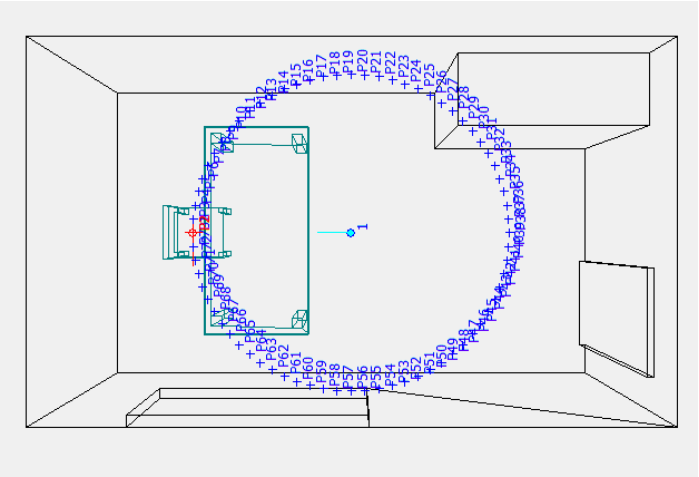

Figure 3.3 - Top down view of the ODEON model

Within Figure 3.3, the sound source is depicted in red, and the receiver is light blue in the middle of the model. Using the ODEON model, BRIRs will be generated on a horizontal plane with 5° of separation between the different BRIRs. These different rotations are marked with the blue crosses on Figure 3.3. Each cross indicates a different source. For each angle of rotation, a new source must be imported into the model, leading to a total of 72 sources in the model. However, for the generation of each separate BRIR, only the red sound source must be active, and all other must be disabled, to ensure the correct orientation of the receiver.
The CIPIC HRTF database[1] of Subject 21 (KEMAR dummy head) were used here to create the BRIRs. The HRTFs are provided with a 5° resolution, which is perfect. ODEON will select the HRTF from the database that is closest to the angle the receiver is facing in case they do not line up perfectly. Finally, the auralization setup in ODEON must be setup as desired. The used setup information can be found in Appendix A.

3.2.3 ODEON's BRIR drawbacks

The BRIRs generated in ODEON are not suitable for dynamic auralizations. The processing ODEON does to create the BRIRs works well for static auralizations, but is not applicable for dynamic auralizations and comparisons between different BRIRs. This is caused by the processing that ODEON does to the BRIRs. The data is always normalized between the left and right ear. Meaning concretely that for each set of BRIRs, the 'ear' that is closer to the source will be normalized to an amplitude of 1. The normalization algorithm boosts certain frequency bands beyond their already loud volume, creating unnaturally loud volumes. This occurred while using HRTFs for three different KEMAR subjects (subject 12, 21 and 165 are all tested and had the same results). A comparison of the frontal and back facing BRIRs can be seen in the figure below. In the graph depicting the left ear values, the back facing BRIR in blue is louder in the direct sound first peak. For both graphs, found in figure 5 below, the back facing (blue) signal is overall quite a bit louder. However, most noticeable in auralizations is the fact that the auralized signal gets louder when facing away from the source in comparison to facing the source head on.

[1] https://www.ece.ucdavis.edu/cipic/spatial-sound/hrtf-data/

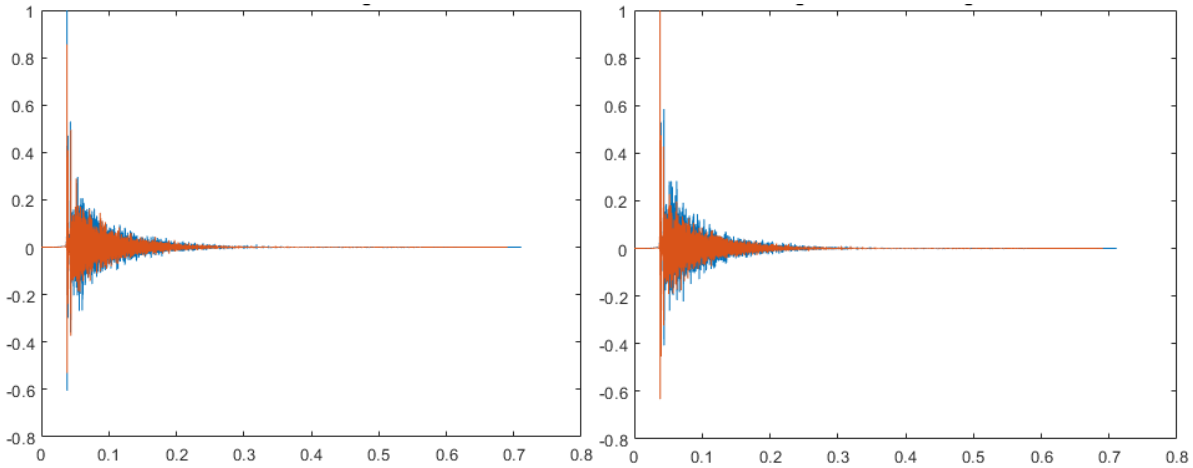

Figure 3.4 - Comparison of 180 degrees rotation source [blue] 0 degrees rotation source [orange]. Plotted signal is the impulse response of the room in Figure 3.2

3.3 MATLAB

3.3.1 Data pre-processing

The BRIR signal that ODEON produces are stored in .wav files. These files must be processed to be made ready for use in Unity. Unity can work with audio files, but to ensure accuracy from Unity with the audio, the wav files will have to be loaded in differently. Unity has the capacity to work with text files as GameObject in the form of TextAssest, but it is even faster to have Visual Studio work with .txt files, because they can be read in, without having to do any processing on the .txt files. This way, the values of the BRIRs are available to Unity quickly and accurately and can be used in the convolution calculations.

The goal in the preprocessing phase is to split up the .wav files into their respective left and right channels and convert them .wav file's content into floating point values that Visual Studio can read. This is done in MATLAB using 'formatSpec' and 'fprintf' functions. An example of this procedure can be found in Appendix B. The results of this script are two .txt files per BRIR. One for the left ear, and one for the right ear. The contents of these txt files are one float value per sample of the signal. These .txt files are used in the convolution algorithm when the filter signals are loaded in. The same process is used for the anechoic source signals. However, they are mono signals, and splitting into a left and a right channel is not necessary. Within the pre-processing stage, it should be ensured that all signals are using the same sampling frequency. All signals used are sampled at 44,1kHz.

3.3.2 Digital signal processing

The UPOLS method presented in Chapter 2 is at first implemented in MATLAB to ensure the output and the performance of the algorithm are correct. Some issues were found on implementing the delay with the use of a frequency domain delay line, so adjustments were made to the method. An updated diagram containing the revisions in the method can be found in Figure 3.4.

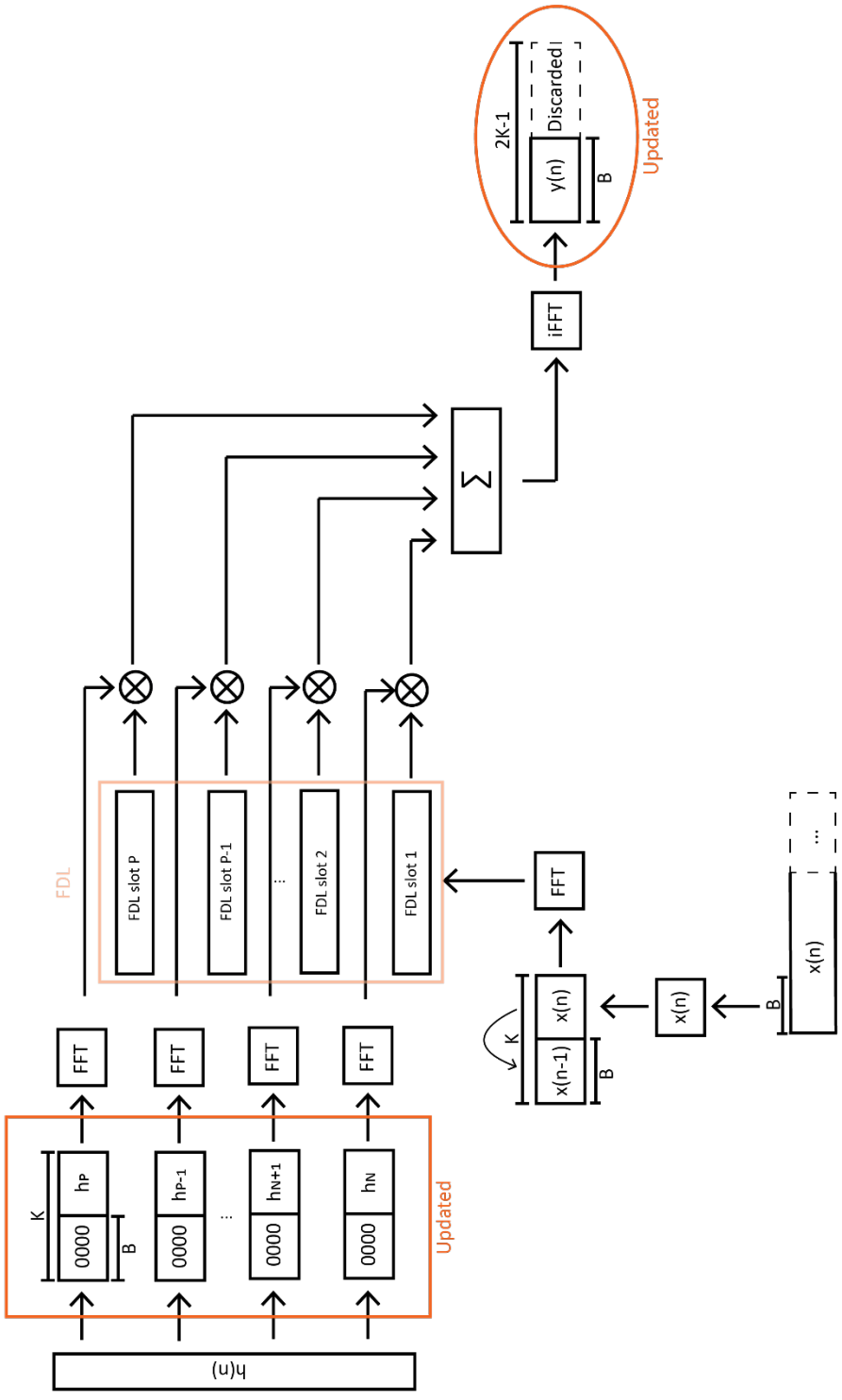

Figure 3.5 - Updated UPOLS algorithm diagram

Delay was not implemented using the FDL method, but instead, the filter blocks are pre-delayed with a B-sized block of zeros, to achieve the desired outcome. Results of the updated UPOLS function can be found in Figure 3.6 and 3.7. The FFT operations are done using the build-in fft.m function in MATLAB. Similar functionality in C# was achieved using the FFT libraries mentioned previously in section 2.3.

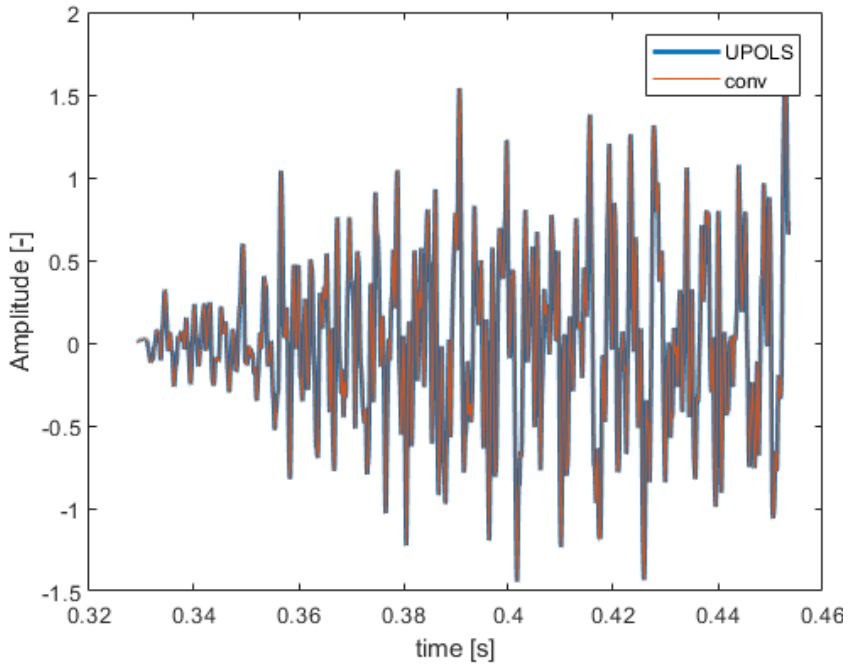

Figure 3.6 - Updated UPOLS algorithm and MATLAB conv function - 5000 sample extract from a convolved piano signal with the IR of the room of Figure 3.2

From a convolved piano sample 5000 samples were extracted to create Figure 3.6. Since the updated version of the UPOLS algorithm applies delay differently, its output signal is pre-delayed as well. To ensure the signals match up perfectly, the pre-delay of a single block of zeros had been removed, and the result is presented in the figure above. A frequency domain representation of the full signal can be found in Figure 3.7.

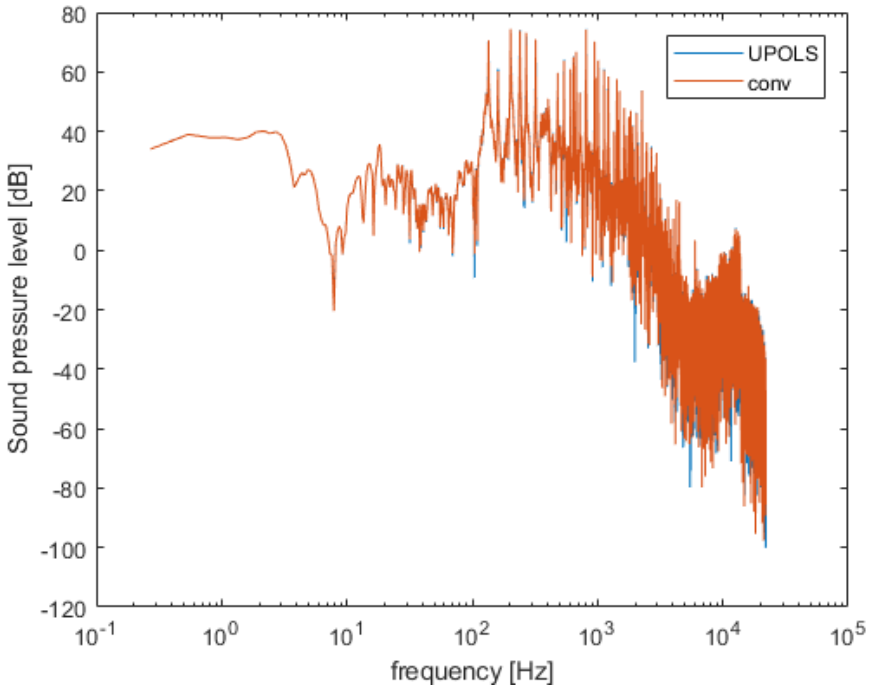

Figure 3.7 - Updated UPOLS algorithm and MATLAB conv function - Frequency domain information of a convolved piano signal with the IR of the room of Figure 3.2

In the frequency domain, the graphs are similar in shape and maintain a similar frequency content. Most importantly, the results sound good, and not distorted at all. This means that the UPOLS algorithm is ready to be implemented in the Unity model.

3.4 Unity

After the validation of the UPOLS method was completed in MATLAB, it was then integrated into a Unity model, together with the 3D geometry. The Unity platform was chosen for its ease of use, and its compatibility with the Oculus VR headset and with SketchUp 3D models and its inherent ability to execute C# code through its integration with Visual Studio. Other platforms or engines like the Unreal Engine could also have been used for a similar purpose. On this Unity platform, several goals must be completed to create the required full VR experience.

- The UPOLS method must be adapted to a dynamic real-time convolution method, working in cohesion with the Unity platform. Convolution and filter updates will have to be updated in real-time and adapt to the head rotations of the user, which is tracked by the Oculus headset.
- A visual representation of the corresponding location of the auralizations must be integrated with the model and be available for all rotation angles.

Firstly, the 3D geometry is considered. The SketchUp model contains all information necessary to be displayed in the 3D environment, since it has been exported in the .dae file format. When this file is loaded into Unity as a new asset, the geometry can be placed in the environment by dragging it in. However, the scale might not be correct, and can be adjusted by setting a scaling factor in Unity. After that, orientation and location of the model must be set as desired, considering the camera (and therefore VR headset) position in 3D space as well.

Next, the digital signal processing (DSP) part of the model must be addressed. Since Unity allows the execution of code through C# classes (named scripts in Unity) coded in Visual Studio, this option was used to execute the UPOLS convolution algorithm. This allows this project to bypass al the audio processing capabilities of Unity and perform each step of the process exactly as wanted. Audio playback in Unity is discussed in the next section 3.5.2. Output generation precedes output playback.

The C# language is different from the MATLAB coding language. It is however an object-oriented language and therefore understandable with a basic understanding of the MATLAB style of programming. References for methods and functions can be found here[2]. Noticeable differences between MATLAB and C# are that arrays and lists start on zero, instead of one, and once the size of a variable has been declared, it cannot be altered. Of course, exceptions to this rule exist, but most problems can be solved with the Microsoft documentation.

Fourier transform calculations are not natively available in C#. To reintroduce the ability to perform Fast Fourier Transforms in the code, the FFTW library is used. Its functionality allows the usage of the FFT and the IFFT functions in the C# classes within Unity. It can be downloaded here[3]. This library is available free of charge and licensed under the GNU Lesser Public License v2.1, meaning it can be used in any commercial or private project.

[2] https://docs.microsoft.com/en-us/dotnet/csharp/index

[3] http://www.fftw.org/

3.4.2 Running convolution

Audio playback in Unity occurs through game objects called Audio Sources and Audio Listeners. Their respective locations can be used to create spatialization effects in Unity (Bottom halve of Figure 3.7). These effects were disabled in this project, because all spatialization effects are created through the auralizations. Audio Sources will play the content of Audio Clips (Top slot of Figure 3.7) Audio clips are of a finite length and must be filled with audio data before being assigned to an Audio Source. Audio Sources can then play this audio clip once, when they are triggered by a function or code. Looping the content of the audioclip is also an available option. Audio clips can be exchanged with the use of C# scripts.

Figure 3.8 - Audio Source game object in Unity

To ensure smooth playback of the partitioned convolution algorithm, the output of the convolution calculations must be inserted into the Audio Source in the form of a new audio clip for each individual output block. Ensuring no gaps in playback is not possible using this method, since updating the audio clip cannot be done fast enough. Playing the output form the partitioned convolution smoothly is done through the built in Unity method called OnAudioFilterRead[4].

This method still utilized the Audio Source and Audio Listener mentioned previously, but forgoes the usage of audio clips. Instead of clips of finite length, this method is built to play back streaming data. The OnAudioFilterRead function pulls audio data from its own built in float array called 'data'. If this float array contains audio data, the OnAudioFilterRead function will play this content back in increments that are defined by the audio settings in the Unity project. Partition block size of 256, 512 or 1024 samples are available. Figure 3.8 illustrates the steps taken by Unity to perform a single step of the partitioned convolution algorithm.

[4] https://docs.unity3d.com/ScriptReference/MonoBehaviour.OnAudioFilterRead.html

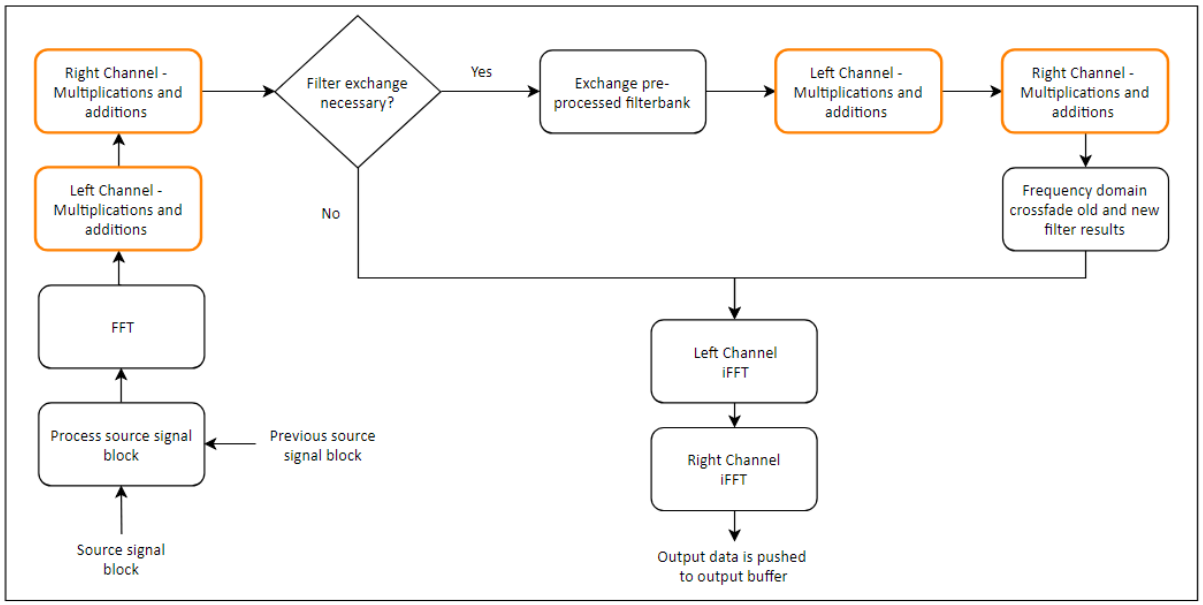

Figure 3.9 - Real-time processing step in Unity

The latency of the algorithm depends on the chosen partition block size. For a block size of 512 samples, this is:

$$t = \frac{B}{fs} = \frac{512}{44100} = 0.0116s \tag{3.1}$$

It takes 0,0116 seconds to play back a section of 512 samples of the auralization. The following section of output data must be calculated within this amount of time to make sure the output data is available for play back in time, ensuring continuous play back. If the OnAudioFilterRead method is included in one of the C# classes active in the Unity project, it will be called automatically. This call happens for each time that a block of time has passed. For a processing time of 512 samples, or 0,0116 seconds, it will automatically be called after every 0,0116 seconds, independent of any other function in the project. Within those 0,0116 second the steps depicted in Figure 3.9 are done every time.

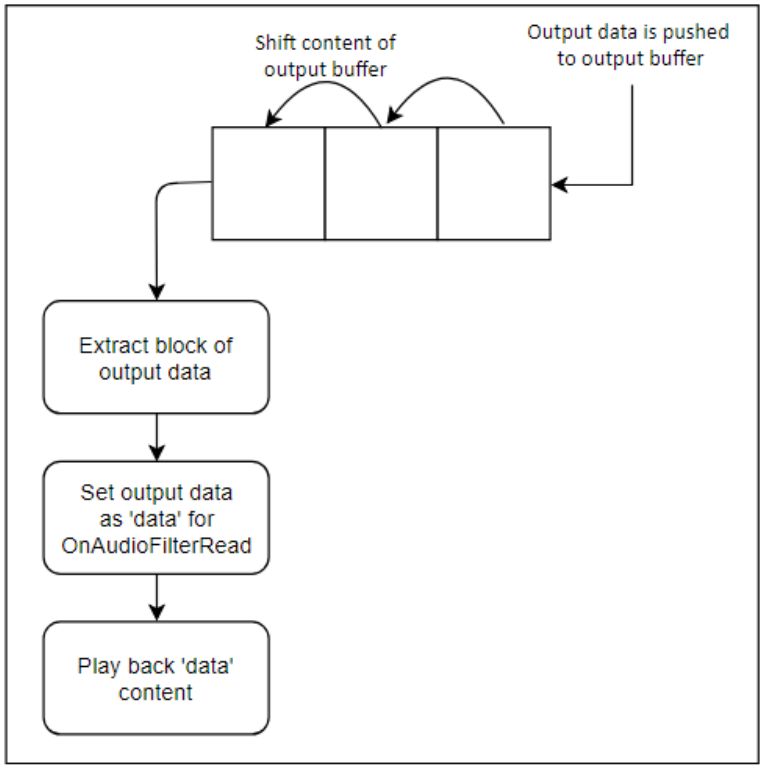

Figure 3.10 - OnAudioFilterRead() steps

During those 0,0116 seconds, a block of data is played back, and the following block of data is being prepared. The output buffer op top of the figure is filled by the steps presented in Figure 3.8. If at any point the processing of the auralizations takes longer than the available time, a small buffer of data is available to ensure continuous playback. If no data is present for the OnAudioFilterRead function to play, Unity will give an error, but the project will not stop running. After another block of time has passed, the function will simply try again to play the audio data in the float 'data' array. This will be noticeable by a gap in the audio playback.

The effect of the user on this process is hinted at in Figure 3.8. The possibility of exchanging the filter bank the auralization is created with is indicated there. This filter exchange is triggered when the head rotations of the user, tracked by the Oculus headset, exceed a rotation of 5°. As mentioned in section 3.2.2, the BRIRs are generated on a horizontal plane for every 5° of rotation, resulting in a set of 72 BRIRs. Switching between these filters occurs in the middle of the generated filters. If a user is facing straight ahead, with a 0° rotation, the auralizations are created with the BRIRs associated with the 0° generation point. For as long as the user stays within a rotation of 357,5° and 2,5°, the same filter will be used. Switching occurs the moment the user rotates past 2,5°, at which time the filter generated at a 5° angle is used.

Whenever a filter exchange is prompted, the next output block requires extra processing steps. With the following source signal block, a convolution calculation is done with the old BRIRs, and with the new BRIRs, as depicted in Figure 3.8. Crossfading in the frequency domain is done between those calculations, with the method descripted in section 2.7. This crossfade operation is done to prevent the clicking artifacts that may occur when switching between different BRIRs [24].

3.4.3 Artifacts

At this point in the development of the calculation algorithm, some problems were still present. The processing speed of the algorithm was not high enough to ensure a latency low enough to provide artifact free continuous playback. Artifacts would occur during heavy workload scenarios. Fast rotation of the camera for instance. Artifacts would reduce when using longer processing blocks (of 1024 samples) and with shorter filters as well, but they were never fully gone. Possible improvements for this algorithm could be by optimizing the code as is, introducing some form of parallelization in the calculations or utilize the GPU, next to the CPU, to do some of the processing.

3.4.4 New development from an external partner

Implementing the mentioned possible solutions for the artifact problem in the previous section is no straightforward task. An external developer from the Building Acoustics group was recruited to provide a solution for this problem. He developed a method to execute the multiplications and summations within the convolution calculation using GPU processing. Effectively creating a parallel process which sped up the total auralization calculation by a factor of 2 – 3 times. Theoretically, this would mean that the audio processing would happen fast enough to keep up with a real-time data stream, using blocks of 512 samples.

Random artifacts could still be heard during tests of this extended algorithm. Especially when creating longer auralizations (around 45 seconds in length during testing) random artifacts could be heard in the form of clicks. This click was never heard at the same time during play back. Sometimes it occurs after 10 seconds, and sometimes it occurs after 40 seconds. A correlation between head rotations and the artifact was not found. Auralizations created with shorter filters present a similar click artifact, but it occurs fewer times. Some runs with shorter BRIRs (with a length of around 1 second) produced no audible artifacts at all. However, they can still be heard in others.

This problem has not been solved yet, and it is something that will be considered in the listening tests that will be performed using this algorithm and this Unity project.

Chapter 4: Listening tests

4.1 Introduction

A listening test was designed to asses the quality of the dynamic convolution algorithm developed with an external partner of the Building Acoustics group. This test was fully executed in virtual reality, with the use of the Oculus headset. The created auralizations adapt to the rotations of the listener in real-time. Therefore, the focus of these listening tests is put on the quality of the real-time dynamic convolution method, and not on the realism of the output signal. First, this chapter discusses the selected sound sources and the used binaural impulse responses. Following up with the graphical user interface designed in VR and the listening test methodology. Finally, the results of the listening tests are shown and conclusions based on these results are drawn.

4.2 Audio characteristics

Three different sound source samples were used in the listening test. A sample of drums playing, a sample of a man speaking and a pink noise sample. Comparable sets of source signals have been used in other listening tests in for instance [26]. Broadband spectrum source signals are recommended [27]. The length of the samples was truncated to 10 seconds. This was done to ensure the total listening test would not be too long and the attention of the listener would be maintained troughout. ITU-R BS.1284-1 [28] recommends the source signals to be no longer then 15 to 20 seconds in length, and even shorter in certain tests.

The following binaural room impulse response (BRIR) sets were used to create the auralizations:
- Anechoic HRTFs of the KEMAR dummy head. (Length: 0,0116 seconds).
- Measured BRIRs of the Maranatha church. (Length: 2,5 seconds).
- Measured BRIRs of the Sint-Catharina church. (Length: 7 seconds).

The used auralizations therefore differ in length from 10 – 17 seconds, depending on which BRIR is used. With that in mind, they do remain within the recommended total length. The HRTFs used are from the CIPIC HRTF database[5] as mentioned in section 3.2.2. They are measured in the horizontal plane, for every 5° degrees rotations. The BRIRs from the Maranatha and the Sint-Catharina church were measured using B&K dummy head. They are measured on the horizontal plane as well, for every 5° of rotations. However, the data was only measured for a half rotation (until 180°) and assumed to be identical for the right and left side, since the measurements are done on an axis of symmetry of the churches. To create a full set of BRIR data, the measurement data were mirrored for the unmeasured side. Measurements and post-processing of this data was done by F. Georgiou and I. Andreazzoli.

4.3 Attributes

Participants were asked to judge the auralizations on three different attributes. These attributes and their descriptions were taken from [29] and [24] and modified where necessary for this test. These attributes together with their definitions, which was provided to the subjects, are given below:

- Smoothness [24]
  - Switching BRIRs can cause the sound images to be perceived disrupted. Fluctuation in amplitude and other artifacts could be perceived when the

[5] https://www.ece.ucdavis.edu/cipic/spatial-sound/hrtf-data/

smoothness in the model is bad. Good smoothness indicates a complete lack of these artifacts, and bad smoothness indicates the presence of these artifacts.

- Responsiveness [29]
  - Responsiveness describes the speed with which the model responds to input of the user. Delay in response of the model is therefore bad responsiveness. This delay could originate from delay in updating the BRIR in response to movement from the user. Good responsiveness is the absence of any delay in the model.
- Externalization [29]
  - Externalization describes whether a sound event is perceived inside or outside the head. Bad externalization would be characterized by a sound event that is perceived inside the head (in-head localization), which is a common problem in headphone listening. Good externalization would be characterized by a sound event experienced outside the head, fully externalized in space (out-of-head localization).

Each attribute was chosen to serve as a qualifier for the quality of the dynamic real-time auralization process. It was not intended for the participants to judge the auralizations on their realism.

Grading of the attributes was done on a direct continuous category scale. This scale is recommended in ITU-R BS.1284 [28], which is based on ITU-R BS.1116 [30]. It is recommended in [27] to use an already established grading scale in listening tests whenever possible. The same scale has been used successfully in [31]. A figure representing the scale can be found in Figure 4.1.

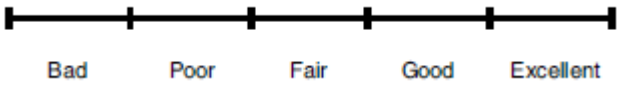

Figure 4.1 - Listening test grading scale

This is a continuous quality scale with values between 0 – 10. The categories assigned are nominal anchor points and are meant as a reference for the listener in assistance to the scoring process. These categories refere to a certain score, which can be seen in table 4.1.

Table 4.1 - Score categories

Category	Score
Bad	0,0 – 1,9
Poor	2,0 – 3,9
Fair	4,0 – 5,9
Good	6,0 – 7,9
Excellent	8,0 – 10,0

4.4 Participants

In total, 19 people of age between 23 and 41 years old (average age 29,1 years) participated in this test. 8 people were female, and 11 people were male. One third of the participants had significant expericience with listening tests, while for other this was not the case. Non of the participants were experienced with virtual reality.

4.5 Methodology

Auralizations are created using all combinations of each of the three source sound samples and the three BRIR sets. This results in a total of 9 different auralizations. Each auralization is tested for all three attributes, resulting in a total of 27 possible scenarios. In the listening test, each of the BRIRs is

combined with their respective room. A 360° picture of each of the rooms is used to add the corresponding visual aspect to the respective auralization. The listening test was split up in three parts, one for each of the rooms. In between these parts, the participants had a minute to take a break. The procedure was the same for each of the three parts of the listening test. The participants were presented with nine (three attributes for each of the three sound source samples) scenarios for each of them. Each of the nine scenarios is assigned a number and each scenario was presented to the participants in random order. In total, all scenarios were presented to the participant twice.

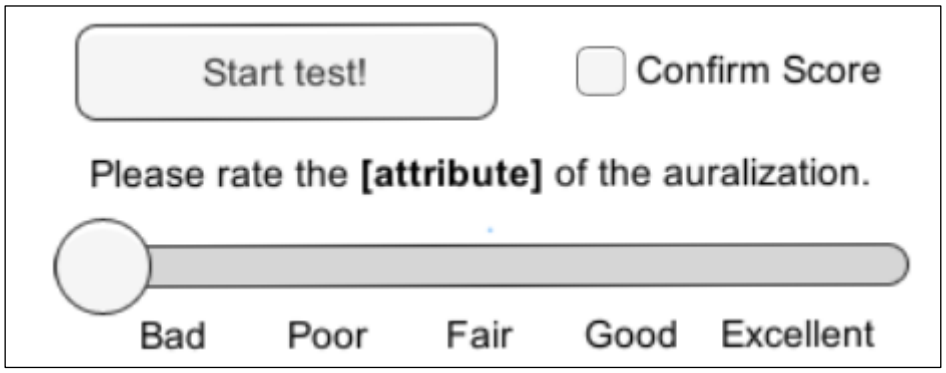

Figure 4.2 – Unity GUI example

After each auralization scenario, the participants were presented with the graphical user interface (GUI) similair to figure 4.2. During the auralization, the attribute the participant needs to focus on is indicated, as indicated on the screenshot in figure 4.3 and this is then repeated in the question in the GUI.

Figure 4.3 - Listening test screenshot

The GUI pops up when the auralization is done playing, and will disappear again when the score is confirmed by the participant. The listening test proceeds along the following steps:

1. Initially, the subjects were presented with the GUI from Figure 4.2, but the button ‘s label was “Start test”.
2. When pressed, the GUI fades away and the first 10 second auralization start playing.
3. When the playback is finished, the GUI pops back up, as indicated in figure 4.2.
4. Score can be indicated by moving the slider in the GUI with the oculus touch controller.
5. Subjects had to select the ‘Confirm score’ tick box in order to activate the ‘Next simulation’ button.
6. Goes back to step 2, until all scores are recorded.

7. When the final score is saved, the GUI will instruct the user to remove the headset and thank them for their participation.

This process was repeated for each of the BRIRs/rooms, resulting in a total of 54 scenarios in the complete test. Preceding this process, the participants take a small training session to familiarize themselves with this process, the VR headset and the GUI they need to use. This training session is 6 auralizations long, after which most participants were comfortable with the procedure. To further ensure that the test procedes as intended, user were given written instructions prior to the start of the test, combined with oral instructions. These written instructions can be found in Appendix C at the end of this raport.

4.6 Results

In total, each of the 19 participants were presented with 54 scenarios. The 54 scores associated with these scenarios are sorted by scenario for each of the participants. Score were saved with a single decimal point of significance, as indicated by [28]. The median values of the test results, for each attributes and each room, are shown in table 4.2. Boxplots were made from the full results, sorting scores based on BRIR/room, and sound source sample. One figure is dedicated to each of the tested attributes. Figure 4.3 depicts the boxplots for the externalization attribute, Figure 4.4 depicts the boxplots for the responsiveness attribute, and finally Figure 4.5 depicts the boxplots for the smoothness attribute.

Table 4.2 – Median values of the listening test scores per room per attribute

	Anechoic room	Maranatha	Catharina
Externalization	5,1	6,8	6,7
Responsiveness	6,0	6,4	6,7
Smoothness	6,2	6,9	6,7

4.7 Comment from test participants

Participants were asked to comment on the test after they were done. Most participants really liked the test and enjoyed the oppurtunity to experience VR for the first time. Some people had some problems with the Unity GUI and interacting with it using the touch controllers. This was mostly solved during the training sessions. The training session merely consisted of six used samples. These samples could have been a bit longer, to give the participants more time to get used to the GUI and the listening test methodology. This would give them more time to familiarize themselves with the model. Since the test did not utilize a reference signal, some people had some issue scoring the samples on the 0 – 10 scale.
Most people were content with the sample length of 10 seconds, especially further on in the test. However, a couple of participants would have liked to have more time to decern the individual samples. This is not a problem to achieve, but would make the test in total longer as well. Depeding on listening test specifics, this might be something to reconsider.

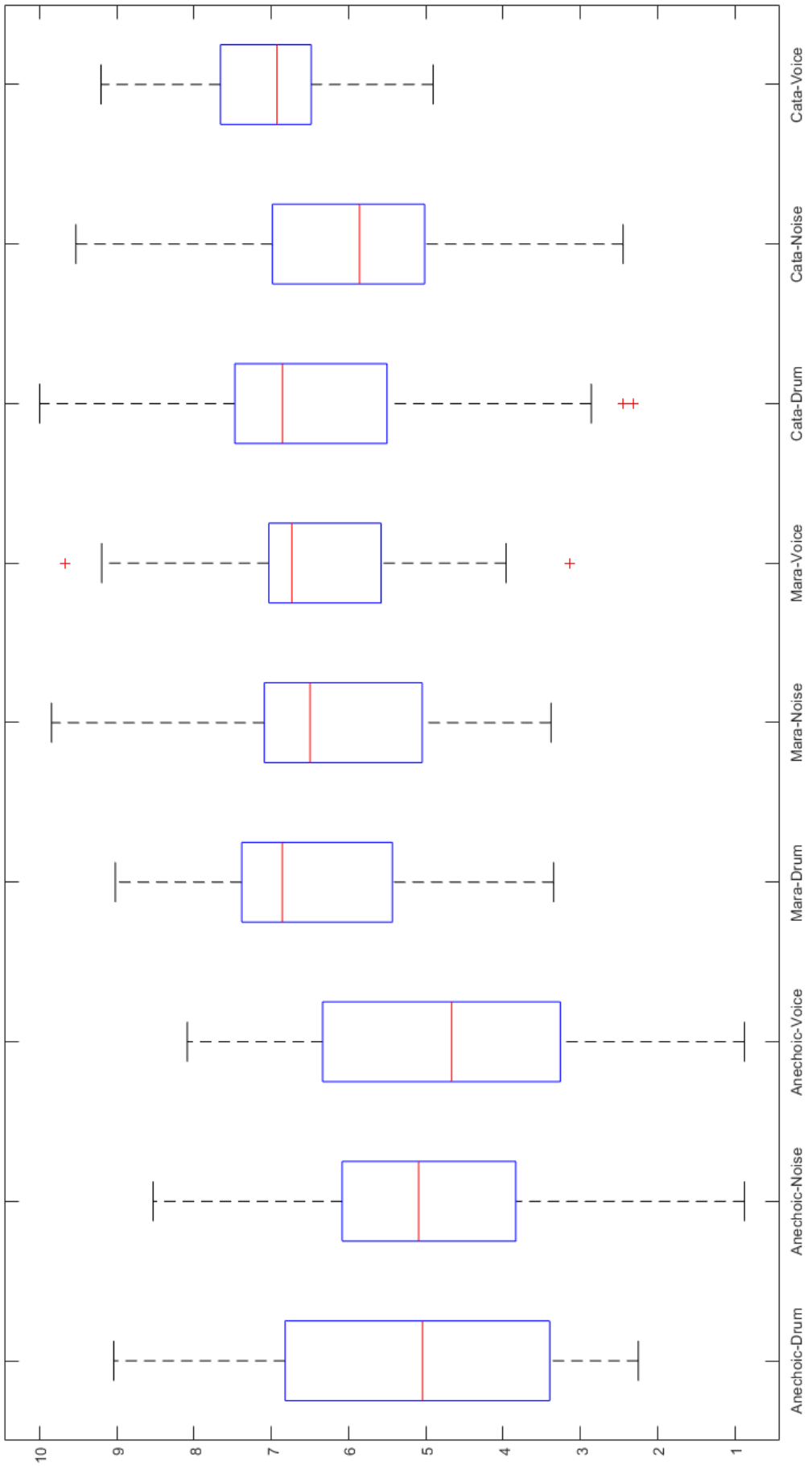

Figure 4.4 – Average scores for the externalization attribute per scenario. Mara = Maranatha church. Cata = Catharina church.

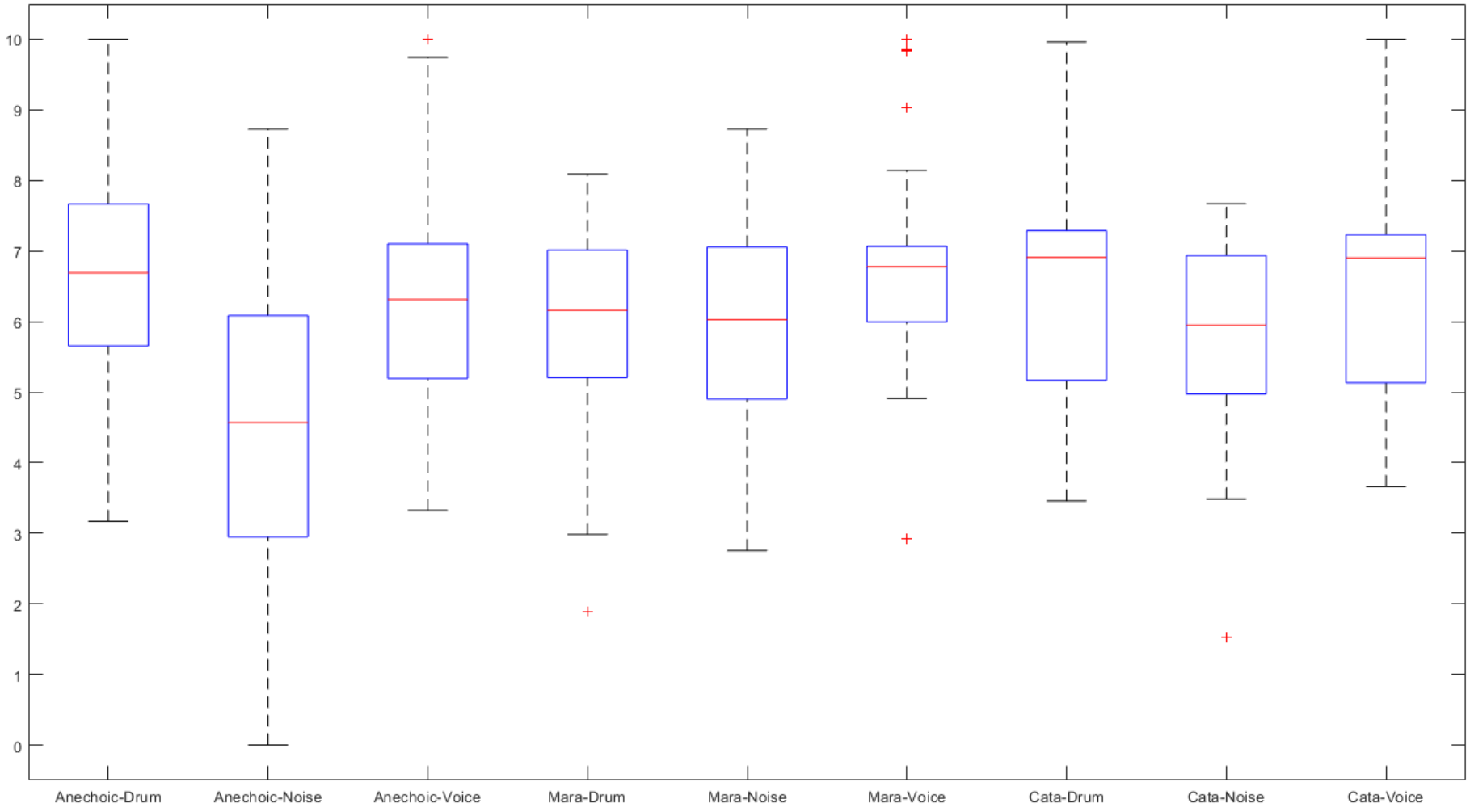

Figure 4.5 - Average scores for the responsiveness attribute per scenario. Mara = Maranatha church. Cata = Catharina church.

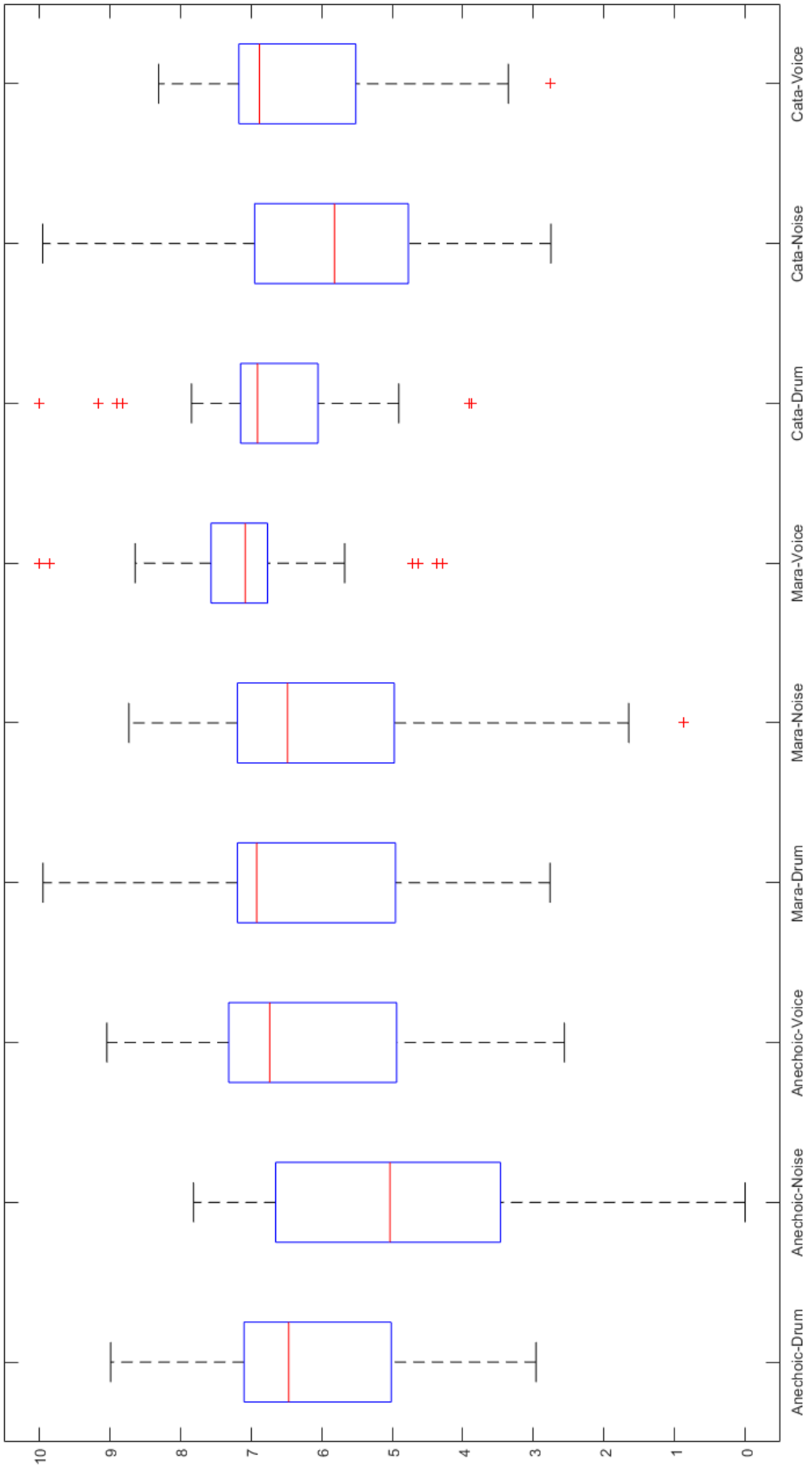

Figure 4.6 - Average scores for the smoothness attribute per scenario Mara = Maranatha church. Cata = Catharina church.

Chapter 5: Discussion

5.1 Result discussion

The main goal of this thesis was to adapt a real-time convolution calculation method to consider head rotations and crossfading between different BRIRs. This method was executed on a virtual reality platform. As mentioned in section 3.4.4, a crucial section of the calculation method was developed by an external Building Acoustics development partner. A listening test was designed and performed to assess the quality of this adapted method. The results of this listening test will be discussed here. The modelling process and associated activities are also discussed.

Results in the previous chapter were sorted by attribute and divided by the difference scenarios presented to the listening test participants. Some indications of performance differences can be seen in these figures. Across most scenarios, the median of the scores is between the 6 or 7 mark, as indicated by table 4.2. This corresponds to a score between 'fair' and 'good' given during the listening test. The minimum scores were the lowest in the anechoic room scenarios, with a median score between 5,2 for externalization and 6,2 for the smoothness scenario. This was expected due to lack of reverberation in these scenarios, but no overall significant differences in results can be found. Considering the smoothness attribute, some high scores can be found for the voice source sample in the Maranatha church and the drum sample in the Catharina church when going over the general listening test scores. The median scores for smoothness are overall the highest, with a 6,9 for the Maranatha church scenario especially. They are however, again not exceptionally different from the rest. Especially when considering the median scores with the 25% – 75% ranges from the other scenarios. A similar conclusion can be drawn when comparing the scores from the noise sample auralizations with the other two samples. No thorough statistical analysis was done on the listening test scores however.

5.2 Audio processing discussion

The convolution calculation method was executed through C# code on the virtual reality platform Unity. This platform provided plenty of challenges and limitations for the digital audio processing that was required for this project. The way Unity executes code through C# classes and the order of operations in code, audio and execution of different tasks was never completely clear. As mentioned before in section 3.4.4, during testing period of the auralization algorithm, small inconsistent artifacts could be heard. Their cause was never found or completely solved. During the listening tests no mention was made of these artifacts in the instructions provided to the listener, but none of the participants specifically mentioned hearing artifacts as well. Audio settings within the Unity platform also limit the available options for the partition size of the blocks in the partitioned convolution algorithm. Perhaps writing an external plugin, instead of using the build in C# class abilities, would be a better option. HRTF sets were used with data available for each 5° degrees of rotation. Measures BRIRs were used with data for half of the rotation (0° - 180°). More accuracy in this data, or a different method in crossfading the output of this data could perhaps have resulted in a more accurate model and higher scores for smoothness.

Chapter 6: Conclusion

6.1 Conclusion

This thesis set out to create a real-time dynamic auralization method, capable of utilizing head rotations of a person using a VR headset, and crossfading between BRIRs in the frequency domain. Several real-time capable convolution calculation methods were evaluated, and the most suitable one was chosen for this goal. The method was successfully adapted to consider head rotation information provided by an Oculus VR headset. Output between different BRIRs is generated with the use of a frequency domain crossfade method. This entire algorithm is successfully implemented in virtual reality on the Unity platform. A listening test GUI design was made that allowed the writer to question participants about several attributes of the auralization, while remaining in virtual space. Median scores of the listening tests mostly fall in between a 6 and a 7, indicating that the method is effective and working well across different sound source signals and within different virtual environments/BRIRs.

6.2 Future work

Several possible improvements could be made to the product that was developed in this thesis. The uniformly partitioned convolution algorithm is not the fastest option available. A conversion to using a non-uniformly partitioned convolution algorithm could provide an increase in processing speed, which could assist in performing better across all attributes.

Using a different method to crossfade between the different BRIRs could increase performance of the algorithm as well. Crossfading in the time domain is still an option, and different windowing functions could prove to be beneficial as well.

Exporting the method to function as an independent plugin would result in more freedom in developing the convolution algorithm. It could provide options that were lacking during the development of the method as it was presented now and could result in a better functioning and faster results.

No statistical analysis was done on the results presented in this thesis. A thorough investigation into the results could provide some insight into the performance of the auralization method. A comparison of this auralization method to the audio spatialization options Unity inherently provides could also be interesting. Focusing the listening test on realism instead of the quality of the process might provide interesting results as well.

Chapter 7: References

[1] M. Imran, A. Heimes, and M. Vorländer, “Auralization of Airborne Sound Transmission for Coupled Rooms in Virtual Reality Auralization of Airborne Sound Transmission for Coupled Rooms in Virtual Reality,” no. March, 2018.

[2] J. Y. Jeon and H. I. Jo, “Three-dimensional virtual reality-based subjective evaluation of road traffic noise heard in urban high-rise residential buildings,” *Build. Environ.*, vol. 148, no. December 2018, pp. 468–477, 2019.

[3] D. Poirier-quinot *et al.*, “EVERTims : Open source framework for real-time auralization in architectural acoustics and virtual reality To cite this version : HAL Id : hal-01712896,” 2018.

[4] S. Serafin, M. Geronazzo, C. Erkut, N. C. Nilsson, and R. Nordahl, “Sonic Interactions in Virtual Reality: State of the Art, Current Challenges, and Future Directions,” *IEEE Comput. Graph. Appl.*, vol. 38, no. 2, pp. 31–43, 2018.

[5] M. Azevedo and J. Sacks, “Auralization As an Architectural Design Tool,” in *EAA Joint Symposium on Auralization and Ambisonics*, 2014, no. April, pp. 3–5.

[6] L. Savioja, J. Huopaniemi, T. Lokki, and R. Väänänen, “Creating Interactive Virtual Acoustic Environments,” *J. Audio Eng. Soc.*, vol. 47, no. 9, pp. 675–705, 1999.

[7] S. Siltanen, T. Lokki, and L. Savioja, “Rays or Waves ? Understanding the Strengths and Weaknesses of Computational Room Acoustics Modeling Techniques,” in *International Symposium on Room Acoustics*, Melbourne, Australia, 2010.

[8] L. Savioja, “Modeling Techniques for Virtual Acoustics,” 2000.

[9] M. Vorländer, D. Schröder, S. Pelzer, and F. Wefers, “Virtual reality for architectural acoustics,” *J. Build. Perform. Simul.*, vol. 1, no. January 2015, pp. 15–25, 2014.

[10] D. Schröder, *Physically based real-time Auralization of interactive Virtual Environments*. 2011.

[11] G. Naylor, “Treatment of early and late reflections in a hybrid computer model for room acoustics,” *Proc. 124th Meet. Acoutical Soc. Am.*, p. 11 p., 1992.

[12] J. H. Rindel, “The use of computer modeling in room acoustics,” *J. Vibroengineering - Pap. Int. Conf. Balt. 2000*, vol. 3, no. 4, pp. 219–224, 2000.

[13] B. Xie, *Head-related transfer function and virtual auditory display*, 2nd Editio. J. Ross Publishing, 2013.

[14] W. O. Brimijoin, A. W. Boyd, and M. A. Akeroyd, “The contribution of head movement to the externalization and internalization of sounds,” *PLoS One*, vol. 8, no. 12, pp. 1–12, 2013.

[15] A. Kudo, H. Hokari, and S. Shimada, “A study on switching of the transfer functions focusing on sound quality,” *Acoust. Sci. Technol.*, vol. 26, no. 3, pp. 267–278, 2005.

[16] F. Wefers, *Partitioned convolution algorithms for real-time auralization*, vol. 20. 2015.

[17] C. Muller-tomfelde, “Time-varying filter in non-uniform block convolution,” *Computer (Long. Beach. Calif).*, pp. 1–5, 2001.

[18] M. Vorländer, *Auralization - Fundamentals of Acoustics, Modelling, Simulation, Algorithms and Acoustic Virtual reality*, First edit. Springer-Verlag Berlin Heidelberg, 2008.

[19] T. G. Stockham Jr., “HIGH-SPEED CONVOLUTION AND CORRELATION,” no. 4, pp. 80–85, 2011.

[20] S. G. Johnson and M. Avenue, “FFTW : AN ADAPTIVE SOFTWARE ARCHITECTURE FOR THE FFT,” *Technology*, pp. 1381–1384, 1998.

[21] F. Wefers and M. Vorländer, “Optimal Filter Partitions for Real-Time Filtering Using Uniformly-Partitioned FFT-Based Convolution in the Frequency-Domain,” *Proc. 14th Int. Conf. Digit. Audio Eff.*, no. 1, pp. 155–161, 2011.

[22] F. Wefers and M. Vorländer, “Using fast convolution for FIR filtering Overview and guidelines for real-time audio rendering,” *Aia-Daga*, pp. 1630–1633, 2013.

[23] P. F. Hoffmann and H. Møller, “Audibility of time switching in dynamic binaural synthesis,” *J. Audio Eng. Soc.*, vol. 53(7/8), pp. 661–662, 2005.

[24] M. Otani and T. Hirahara, “Auditory artifacts due to switching head-related transfer functions

of a dynamic virtual auditory display," *IEICE Trans. Fundam. Electron. Commun. Comput. Sci.*, vol. E91–A, no. 6, pp. 1320–1328, 2008.
[25] B. Carty and V. Lazzarini, "Binaural {HRTF} based spatialisation: new approaches and implementation," *Proc. Digit. Audio Eff.*, no. 1, pp. 1–6, 2009.
[26] A. Lindau and W. Stefan, "on the Spatial Resolution of Virtual Acoustic Environments for Head Movements in Horizontal , Vertical and Lateral Direction," *EAA Symp. Auralization*, pp. 13–18, 2009.
[27] S. Bech and N. Zacharov, *Perceptual Audio Evaluation – Theory, Method and Application*. 2006.
[28] Itu-R Bs, "General methods for the subjective assessment of sound quality Annex 1," *Assessment*, pp. 1–13, 2003.
[29] C. Z. Cavalcante *et al.*, "A Spatial Audio Quality Inventory (SAQI)," *Pesqui. Vet. Bras.*, vol. 33, no. 11, pp. 1364–1370, 2013.
[30] ITU, "Methods for the subjective assessment of small impairments in audio systems BS Series Broadcasting service (sound)," vol. 3, 2015.
[31] T. Hirvonenl, M. Vaalgamaa, J. Backman, and M. Karjalainenl, "Listening Test Methodology for Headphone Evaluation," 2003.

Appendix A: ODEON auralization settings

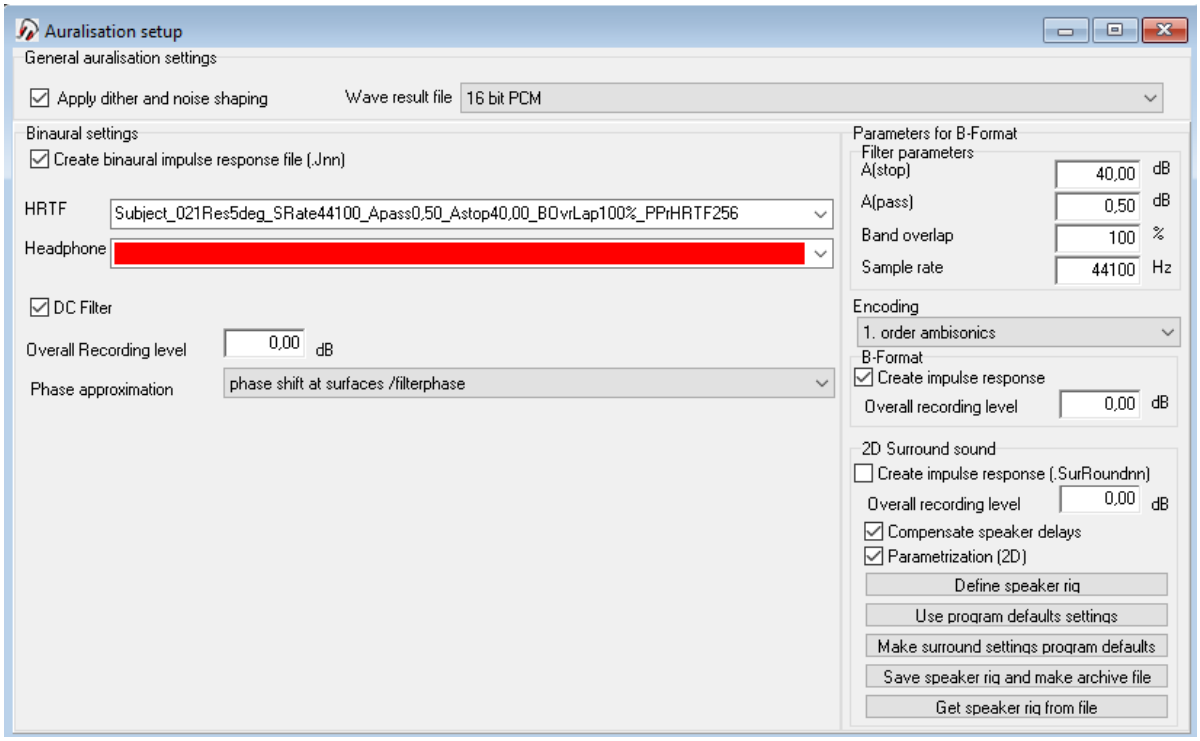

Figure A.1 - Odeon auralization setup

Appendix B: MATLAB code

```
% Create txt files from audio files
clear all; close all; clc;

count = 1;

for i = 0:5:355

    formatSpec =
'Maranathakerk_BRIR_flipped_trun_source_corrected_resampled441kHz_%03d.wav';
    audioreadfile = sprintf(formatSpec,i);

    H = audioread(audioreadfile);
    H_l = H(:,1);
    H_r = H(:,2);

    formatSpecR2 =
'Maranathakerk_BRIR_flipped_trun_source_corrected_resampled441kHz_r_%d.txt';
    textfileR = sprintf(formatSpecR2,count);

    formatSpecL2 =
'Maranathakerk_BRIR_flipped_trun_source_corrected_resampled441kHz_l_%d.txt';
    textfileL = sprintf(formatSpecL2,count);

    count = count + 1;

    fileID = fopen(textfileL,'w');
    fprintf(fileID,'%1.13f\r\n', H_l); %Create file of variable 'H_l', with floats,
formatted with 1 number before the comma, and 13 behind.
    fclose(fileID);

    fileID = fopen(textfileR,'w');
    fprintf(fileID,'%1.13f\r\n', H_r); %Create file of variable 'H_l', with floats,
formatted with 1 number before the comma, and 13 behind.
    fclose(fileID);


end
```

Appendix C: Written instructions for the participants

Dear test participant,

Please fill in your age, whether you have previous experience with listening tests, and whether you are involved in the fields of acoustics in the table below.

Age:	
Involved with acoustics:	Yes / No. (If yes, please elaborate)
Listening test experience: Yes / No.	Number of tests participated in:

This listening test is designed to assess the quality of a dynamic auralization method that is implemented in a virtual reality environment. When 3D models can assist an architect with the representation of a project, auralizations can assist and acoustician in representing how a space will sound. An auralized sound takes the characteristics of the modeled space into account and will sound like it is played back in that room. In this test 3 different rooms are simulated (an anechoic chamber, the Maranatha Church and Catharina church which are located in Eindhoven). In all auralization scenarios within the VR environment the source is fixed on a location right in front of the listener (non-moving source), as represented in Figure 1. The listener however, can rotate his/her head freely in the horizontal plane. The model updates the sound field according to his/her head rotations.

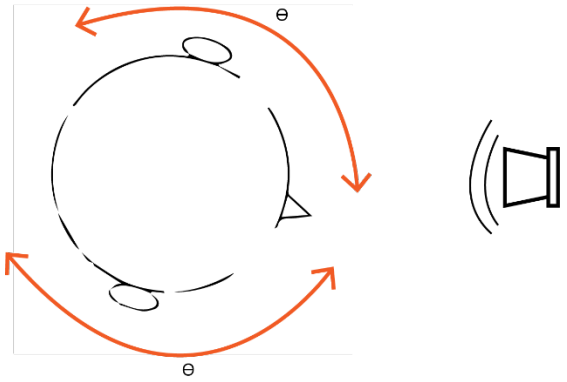

Figure C.1 - Head rotations

During the listening test, you are asked to rate the dynamic auralizations you will hear based on three different attributes. These attributes are:

Externalization

- This attribute describes whether the sound is perceived to be inside or outside the head. Bad externalization is characterized by a sound that is perceived fully inside the head, which is a common problem in headphone listening. Good externalization is characterized by a sound experienced outside the head, fully externalized in space. In the test session preceding the actual test, some examples to assist you with the assessment of this attribute will be played.

Responsiveness

- This attribute describes the speed with which the model responds to the user movement, in this case the head-rotation. Delay in response of the model is therefore bad responsiveness. If this delay occurs in the model, it would be experienced as if the audio information is not updated as

quickly as the visual information. Or a delay between the audio rotation of the actual rotation could be experience.
Good responsiveness is the absence of any delay in the model.

Smoothness.

- This attribute describes the smoothness of the model regarding the movement between the different rotation angles. The switching between angles can cause the sound image to be perceived disturbed. Bad smoothness is indicated by fluctuations in amplitude and other artifacts that occur during that moment. Good smoothness is the complete lack of artifacts during rotations.

To assess these attributes thoroughly, a total of 27 auralizations will be used over this test. Three source signals and three different binaural room impulse responses (BRIRs) will be used (the BRIRs contain the characteristics of the respective rooms), resulting in nine different auralizations, which are to be tested for all three attributes. The test is divided by attribute, and for each attribute, the nine auralizations will be played in random order. The test should not take longer than 30 minutes (including the training session).

The evaluation of the attributes is done within a VR environment. During the test you will be presented with a graphical user interface (GUI) as the one shown in Figure 3, which will be integrated within the VR environment. You can interact with this GUI using the right-hand Oculus touch controller. When the test is started, an auralization will start playing automatically. During playback, the GUI will not be visible. As soon as the sound is done playing, the GUI will pop up, and you will be presented with a question, and an option to answer the question by providing a score using the slider shown in the figure. To ensure no mistakes are made, the score must be confirmed by checking the box below the slider. After that, the “Next simulation” button will be activated and by clicking it you can move to the following auralization. Please make sure to rotate your head during all the auralizations. Each auralization will be played only once (no repetitions are allowed).

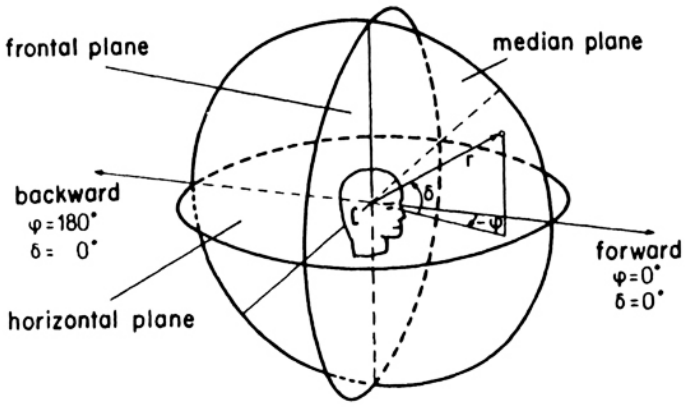

Figure C.2 - Different planes of orientation

It is very important to note that **head movements in the vertical plane are not considered** in the auralization model (visual information is included in the vertical plane but the auralization is not affected with vertical head movements). Thus, please avoid rotating your head in the vertical plane. If you do so, please do not evaluate any of the above attributes based on these head movements. Incorporation of head rotations in the vertical plane has been left for future work. Moreover, you are encouraged to rotate your head at different speeds and angles to get a better insight into the simulations.

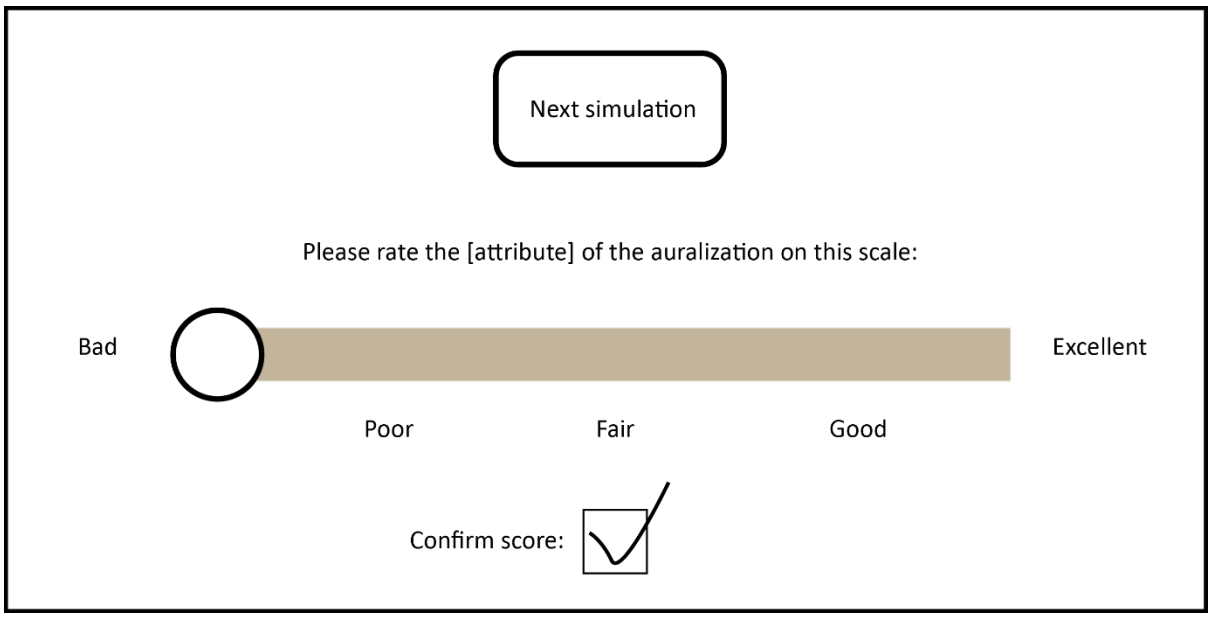

Figure C.3 - Unity GUI

To ensure that the full test will proceed as intended, and the attributes are fully understood, a small training session will be held before the actual test. During this training session, you can familiarize yourself with the virtual environment, the headset, the GUI, and ensure that volume is at a comfortable value for you. After the volume is set, it will remain constant during the entire test. During the training sessions the following auralizations will be played to you:

- Four examples to explain the meaning of 'Externalization'.
- Two 'Responsiveness' examples, in which one of the examples will present poor responsiveness.

After the training session is complete, and you have confirmed that you have fully understood the three attributes and the test process, the actual test will start. As mentioned before, the test will go through a total of 27 auralizations. During any moment when no sound is playing, and the scoring GUI is visible, you can take a break and take off the VR headset. You can continue the test at your own pace by pressing the 'Next simulation' button again. During the whole test duration, I will be outside the listening booth, so if you have any questions or something is wrong with the test interface please inform me right away. The GUI will indicate when the test is done, at which point you can take of the headset.

If you have any extra comments at the end of this test e.g. whether you found the test easy or hard, whether the test interface was good etc., please provide them below:

_____

_____

_____

_____

Appendix D: Using the Unity model

The unity model consists of three main parts, all of which are changeable and adjustable to the needs of the user. These three parts are:

- The visual aspect
- The graphical user interface
- The script logic

Visuals

The visual aspect of the model is mostly defined by the skybox the model is using. To create a sense of immersion into the simulated room, the skybox is replaced by a 3D picture of the simulated location. This picture is assigned to the model as a material component. Solid how-to steps on how to assign a new picture to your skybox can be found here[6].
The only other visual aspects are, in my case, a 3D model of a speaker. This is used to indicate the location of the sound source. This model is loaded in as a SketchUp .dae model, and it inherits all the characteristics assigned to it in SketchUp, like material and color. The material of this object can be changed, but any other changes will have to be made in SketchUp and require loading in the model again.

To ensure the visuals are correctly displayed in the oculus VR headset, the Unity project must be set up correctly. More information on that can be found here[7].

Graphical user interface (GUI)

The user interface that is used for the listening tests and the operation of the model is based on a canvas component[8]. The buttons, text and slider are all combined in a panel component. This makes it easier to hide the complete GUI when necessary. These buttons, text and the slider are basic Unity UI components. The interaction between the UI and the Oculus headset is done through the Oculus touch controller. This setup requires the use of 'Oculus utilities for Unity' which can be imported into the project through the Unity Asset Store. More information of the actual set up of this interaction can be found here[9].

Logic

Any functions and calculations are controlled by scripts and plugins in the Unity project. The project contains the following scripts. Convolution and playback:

- BinauralStudio
- GPUConvolutionApi
- CircularBuffer
- LoadHelper

Other functionality:

- RandomGenerator
- ButtonScript
- PlayerInputs

[6] http://simplifyvr.net/documentation/rendering-a-360-panorama-as-a-skybox-background-in-unity
[7] https://developer.oculus.com/documentation/unity/latest/concepts/book-unity-gsg/
[8] https://docs.unity3d.com/Manual/UICanvas.html
[9] https://developer.oculus.com/blog/unitys-ui-system-in-vr/

As mentioned in the thesis chapters 2 and 3, the auralizations that are created in the Unity project are based on a uniform partitioned convolution algorithm. This algorithm is executed through the BinauralStudio script, in combination with the GPUConvolutionApi. All steps of the convolution algorithm are executed in the BinauralStudio script, but the multiplications and summations in the frequency domain are processed on the GPU with the help of the GPUConvolutionApi. This script allows Unity to use the GPU to execute calculations instead of loading all the calculation on the CPU. The LoadHelper script supplies the BinauralStudio script with the necessary data to perform the convolution calculations. The CircularBuffer script assists the BinauralStudio in the streaming audio playback. Output data from the BinauralStudio script is pushed to a circular buffer, which is then used as the data stream that is played back. This process is assisted by two plugins. The FFTW plugin is used to perform the Fourier Transform operations. The ConvolutionGPU plugin is used to perform the aforementioned calculations using the GPU (instead of the CPU).

The other three scripts perform three different functions. The PlayerImputs scripts allows the user to rotate the camera using keyboard controls. The RandomGenerator script creates an array which contains a random order in which the questions for the listening tests are being asked. The ButtonScript then controls what happens when a certain question is asked.

What to adjust

BinauralStudio
The BrirLength variable. It must match with the length of the binaural room impulse response used. It also must match with the BrirLength variable in the LoadHelper script.

LoadHelper
The string variables at the start of the script indicate what audio samples are being used. These must be changed to use the required samples and filters. The BrirLength must be adapted to the impulse responses as well.

ButtonScript
The script performs different actions using a switch function[10] based on the number of the scenario that is currently chosen. Within a single case of the switch statement, the buttons on the GUI are reset, the skybox can be altered, the text on the menu can be altered, the source signal can be adjusted, and playback of the audio is started. The 'default' case in the switch statement will be used when all the questions are asked.

Audio Settings

Depending on the type of audio samples used in the convolution algorithm the 'system sample rate' might have to be adjusted in the audio settings[11] of the Unity project. The system sample rate should match the sampling frequency of the audio samples used, as well as the sampling frequency of the filters. Within the audio manager, the number of speakers for the streaming output data must be set as well. The DSP buffer size variable determines the block size of the partitioned convolution algorithm. Best latency = 256 samples per block. Good latency = 512 samples per block. Best performance = 1024 samples per block.

[10] https://unity3d.com/learn/tutorials/topics/scripting/switch-statements
[11] https://docs.unity3d.com/Manual/class-AudioManager.html