Real-Time Immersive Sound Scaping

Nikhil Javeri[1†]

December 8, 2016

Abstract

The advent of faster processors and larger stack memories has unleashed a myriad of opportunities for the audio industry. The technological limitations that engineers faced about 10 years ago no longer hamper developments and innovations in the field of Immersive Audio. In this paper, we will take a look at some emerging immersive environment technologies. We will also look at a step through process for creating an immersive audio environment using simple electronic devices and render an aurally immersive environment, real time, on a set of standard stereo headphones.

1 Introduction

In recent years, there has been a lot of emphasis on immersive environments. With faster processors and novel digital processing techniques, it has become possible to compute large amounts of data in the blink of an eye. This project aims at Real Time rendering of natural/immersive sound environments on stereo headphones whilst taking various computationally expensive processes like Head Tracking, Room Modelling, Head Related Transfer Functions(HRTF)/Head Related Impulse Response(HRIR), etc. into consideration.

We will start by looking at some congruent immersive technologies that have been an inspiration for this project. We will then look at the process blocks that characterize this project. We will end with a summary of performance comparisons for this project and also delve into some future aspects and directions that aspirants in this domain can follow in order to expand on this project.

2 Related Work

All of the sections mentioned below describe aurally immersive systems. We will use some of the concepts these systems use and try to apply them to our project.

2.1 3D Audio plugin by Unity

The native DSP plugin in Unity supports Real Time Spatial Audio rendering. The Audio Spatializer SDK[1] is an extension of the native Audio plugin that spatializes sound sources based on their locations with respect to the receiver. It generates spatialization cues by varying the amplitude levels in the left and right channels of the headphones. The Audio Spatializer SDK provides us with a variety of options when it comes to choosing a HRTF database. It provides the KEMAR database from MIT Media Labs[2] as a standard HRTF database along with tools from IRCAM[32] as well. Along with HRTFs, the Audio Spatializer SDK provides us with a Room Reverberation and Occlusion model. The Room Reverberation Model models the reflections that occur because of virtual bounding structures like walls, doors, hills, etc. The Occlusion Model models for specular reflections that translate absorption, reflection and transmission of sound off of structures which help us interpret the type of surroundings we are in. All in all, this is a really good spatialization interface to look at if we are to model an immersive sound scaping system.

[***]This paper reflects upon a project that was developed for the course of Immersive Audio Signal Processing at the University of Southern California.

[†1]N. Javeri is a Graduate Student in the Electrical Engineering Dept. at the University of Southern California, Los Angeles, California, USA. nrjaveri@usc.edu

2.2 Fixing Incus by VisiSonics Corp

While HRTFs are a good way of capturing binaural cues, they are highly individualized. Each of us has different ear topographies and thus have different HRTFs. It is because of this reason that generalized HRTFs don't fit well for everyone. In order to render life-like immersion, we must therefore tailor HRTFs for every user. VisiSonics deals with this very problem. VisiSonics[3] incorporates rapid anthropometry along with Deep Learning to estimate appropriate and personalized HRTF values for a user.

2.3 Nx by Waves Audio

Waves Audio[33] has released a pocket sized wireless clip-on device that can turn any stereo headphones into 3D Audio headphones. This is similar to what we try to achieve in this project. The clip on device tracks the user's head movements and translates them into a perceivable immersive environment generated by the headphones.

2.4 Natural Sound Rendering for Headphones

An informative requirements table has been showcased by Sunder et al. for aurally immersive environments in [18]. The requirements table from [18] is shown below in Figure 1.

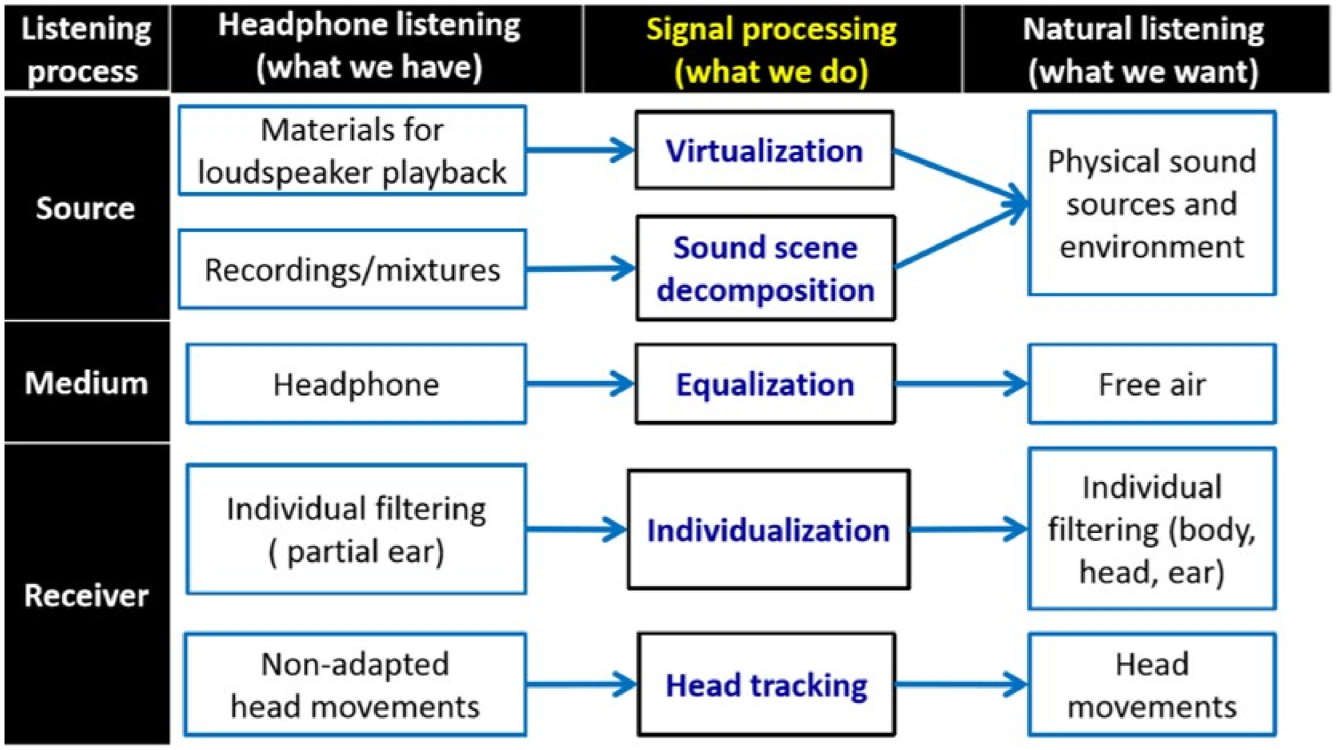

Listening process
Headphone listening (what we have)
Signal processing (what we do)
Natural listening (what we want)
Source
Materials for loudspeaker playback
Virtualization
Physical sound sources and environment
Recordings/mixtures
Sound scene decomposition
Medium
Headphone
Equalization
Free air
Receiver
Individual filtering (partial ear)
Individualization
Individual filtering (body, head, ear)
Non-adapted head movements
Head tracking
Head movements

Figure 1: Requirements Table for Natural Sound Rendering on Headphones. Image obtained from [18]

3 System Design

In the following sections, we will look at the components and the concepts used for creating aurally immersive environments and how they are applied to this project. An aurally immersive system would generally have 2 parts to it:

- The Head Tracking System (HTS).
- The Audio Rendering System (ARS).

Both the sub-systems are computationally expensive on their own and thus need to be processed by individual threads. A system flowchart for this project is given below in Figure 2. As shown in the flowchart, the two sub-systems have been assigned to two different threads. These two individual threads run in a lock-step to generate the required audio output. A list of components and libraries that were required for this project has been given below.

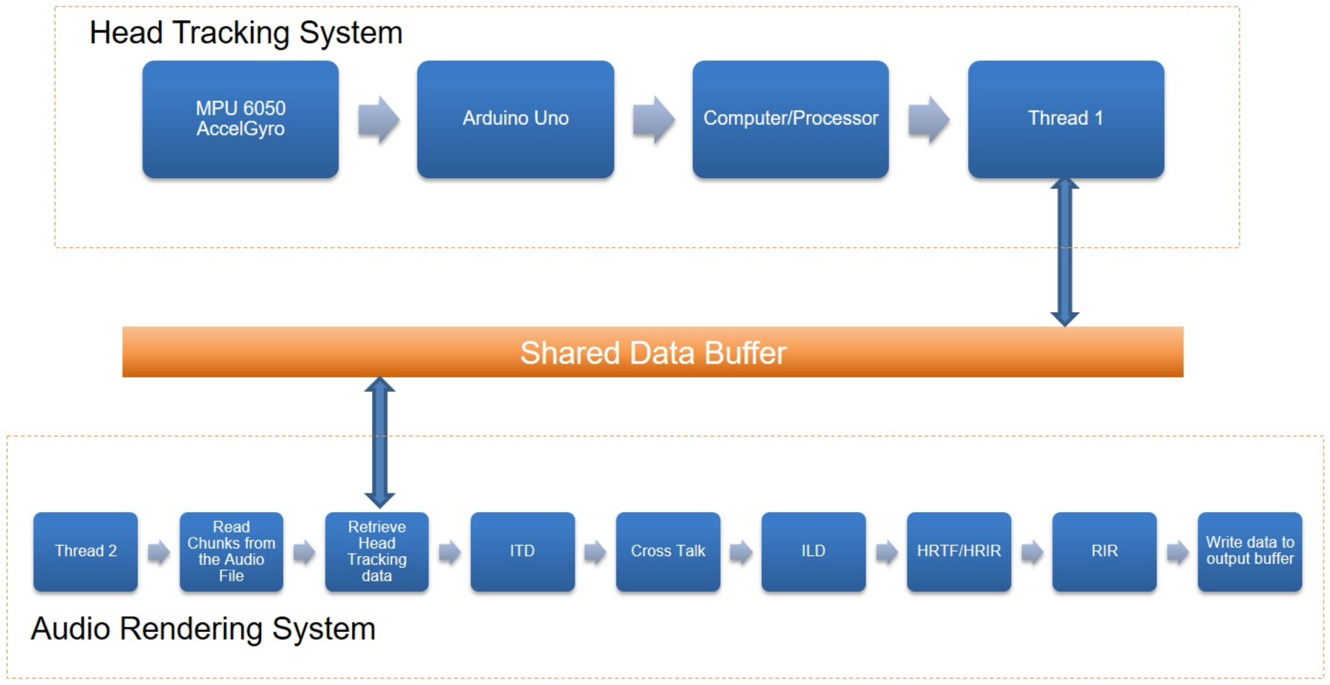


Figure 2: Flowchart of the System

3.1 Hardware Requirements

In this section, we will lay out the hardware specifications and select hardware components accordingly. We will usually try and limit ourselves to choose common electronic items that are readily available in an electronic or an online store. In order to track the user's head movements, we will need a head tracking sensor. An Inertia Measurement (IM) device like an accelerometer or a gyroscope will do the trick for that. We would need an interface for mounting the IM device. For this, we could choose an Arduino development board because it supports a lot of different IM transducers. Finally, we will need a pair of stereo headphones to render the audio data. We could use any standard stereo headphones for this purpose, preferably a pair with a good dynamic range and a flat frequency response. In the following sub-sections, we will look at the actual components we selected for this project and their technical specifications.

3.1.1 Stereo Headphones

For this project, we use Sony's MDRZX110NC stereo headphones. These are standard stereo headphones with specifications as given in [4]. They have a fairly high dynamic range and a relatively flat response which was good for this project.

3.1.2 Arduino Uno

The Arduino Uno[5] is a general purpose development board that supports embedded C/C++. It is a handy tool because of its simplistic architecture and small size. The Arduino board can easily be integrated with a gyroscope to retrieve head tracking data and relay it to the main processor.

3.1.3 MPU 6050 Inertia Measurement Unit

The MPU 6050 IMU[6] is a 6 axis Accelerometer + Gyroscope. It is used as a peripheral device to the Arduino and communicates with it using the I^2C protocol. The IMU has a special Digital Motion Processor (DMP) that computes motion quaternions which are free from gimbal lock. This helps produce true angular values. The IMU has a resolution of up to 2000°/sec which enables it to capture highly refined movements.

3.2 Software Requirements

While considering the software side of this project, we would have to choose platform that could alleviate the development process. Since we have to integrate hardware in to the project, a language that has a good support for hardware and is efficient in processing should be utlilized. For this project, Python was the best scripting option because of the vast amount of support that it provides for Arduino and Audio Processing. Python, as a language, is very user friendly and doesn't need any compilation; Python code can be run straight out of the script. It has tons of support for Signal Processing and Multi-Threading and thus was an ideal candidate for prototyping. In the sections given below, we will go over the tools and libraries that were used for this project along with their importance.

3.2.1 SciPy & NumPy

SciPy[7] or Scientific-Python is a scientific tool that deals with a lot of areas like Signal Processing, Physics, Optimizations, Linear Algebra, etc. It has its own IO class that connects the Python environment to the outside world. We use this IO class to read in and format our target .wav files.

While designing a real time system, we need to make sure that the total processing time for a frame is less than the run-time of the frame itself. For this to be valid, we must optimize our code so that our tasks get completed with minimal number of operations. Numerical-Python or NumPy[8] is a numerical computation tool kit that specializes in optimizing computations. It has a huge support for Arrays, Linear Algebra, Digital Signal Processing, etc. and thus is an important library for this project.

NumPy boasts MATLAB-like array indexing which alleviates the prototyping task. It also has vectorized implementations for all of its classes and methods on the inside which are written in Cython (C/C++ with Python). This speeds up processing and heavily optimizes operations, giving a near C/C++ like performance efficiency in terms of processing time.

3.2.2 PySerial

PySerial[31] is a library that hosts a serial interface for Python. We use PySerial to communicate with the Arduino board. The Arduino sends the orientation data serially to a USB port on the computer. We can instantiate a serial object of class PySerial, set its attributes to match the serial communication protocol hosted by the Arduino board and establish serial communication with it.

3.2.3 PyAudio & Wave

PyAudio[9] and Wave[10] are PortAudio bindings for Python. We use PyAudio to create objects of class Stream which deal with audio playback over the sound card. Wave objects hold the headers and audio data in member variable buffers and supply it to the stream object for playback. The main reason for the incorporation of PyAudio and Wave libraries is because they allow us to split the entire audio data into small segments called as chunks which we can process individually. We can output the processed chunks to the sound card in time and still maintain the integrity of the audio clip.

3.2.4 OpenGL & PyGame

The Open Graphics Library (OpenGL)[11] and Python Game Library (PyGame)[12] are used to render 3D structures. They are utilized as shown in [13] to render the IMU's position on screen for head tracking purposes. This helps in visualizing and debugging the head tracker for accurate orientation and positioning.

3.2.5 Threading & Queue

As the sound scaping system calls for 2 concurrent processes, an ideal solution would be to allocate each of the processes to a thread. For this very purpose, the Threading library[14] is used. Multi-threaded processes can be dangerous if Resource Acquisitions (RA) are not handled properly. The two concurrent threads in the system need to exchange data in between them but need to do so without any RA errors. For this purpose, the Queue library[15] is used which helps the two threads work in a lock-step. A really good example of implementing Multi-threading in Python has been given in [16].

4 System Implementation

4.1 Head Tracking System

In this section, we will go over the process pipeline that governs the Head Tracking System (HTS). Starting with Data Acquisition to the Communication Protocol utilized and Data Deposition in the shared buffer, we will look at the various steps and techniques this project utilizes. Figure 3 below showcases the Hardware Integration diagram for the HTS. Interfacing the IMU to the Arduino board is relatively simple and the interfacing process can be found in [17].

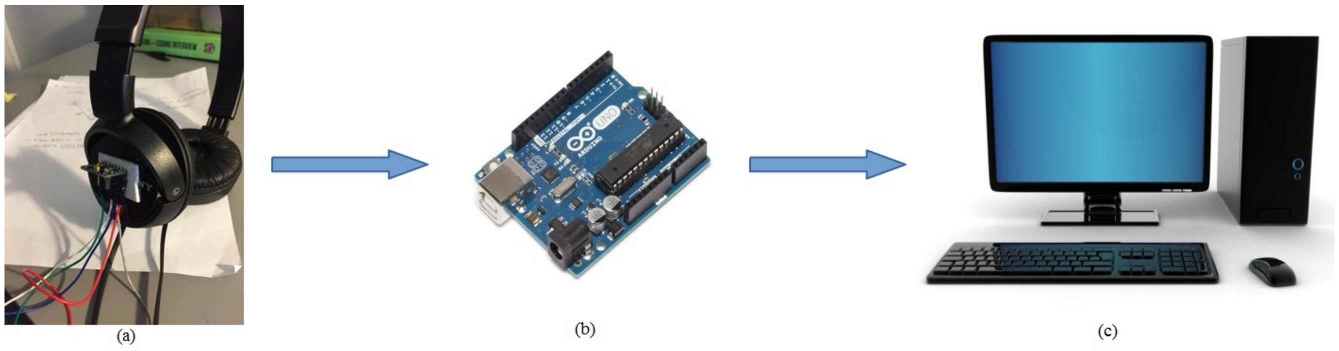


Figure 3: Hardware Integration

4.1.1 Orientation Data Acquisition

The head movements of the user are an integral part of this project since they tell us where the user is going to perceive sound from. In this project, we place the MPU 6050 sensor on the left headphone as shown in Figure 3.a. The Left-to-Right movements are captured by the Yaw of the Gyroscope and the Up-Down movements are captured by the Roll of the Gyroscope. The Shoulder-to-Shoulder movements are captured by the Pitch of the Gyroscope.

It is important to note the orientation of the sensor with respect to the head of the user since we need to map the rotation axes correctly. Since the HTS values are associated with HRTFs, we follow the orientation convention of the HRTF database itself which in our case is the CIPIC HRTF database[19] provided by UC Davis. The anthropometric and sound source location measurements that are used in this project have been explained in [20].

MPU 6050 provides us with values that range from 0°to 360°in the azimuth plane (Yaw) and -90°to 90°in the elevation plane (Pitch and Roll both). However, for the data to be compliant with the CIPIC standard, we need to constrain the azimuth to range from -80°to 80°and the elevations from -45°to 235°. This conversion can simply be achieved by an if condition to flip the elevation vector if the azimuth goes beyond -80°or 80 °. Figure 4.[20] below showcases the location of the data points that form the CIPIC database.

4.1.2 Communication Protocol

The IMU has a DMP that serially transmits the data from its I2C Port to the Arduino. In order to retrieve that data at the remote end on the computer, we open a serial object in Python with the same baud rate as that hosted by the transmitter (38400 bps for this project). Each line written by the transmitter (Arduino) on its serial port is then read by the receiver's (Computer) serial object from the same serial port. To avoid any data collisions, a lock-step semaphore is introduced in the communication protocol wherein the transmitter transmits its data only

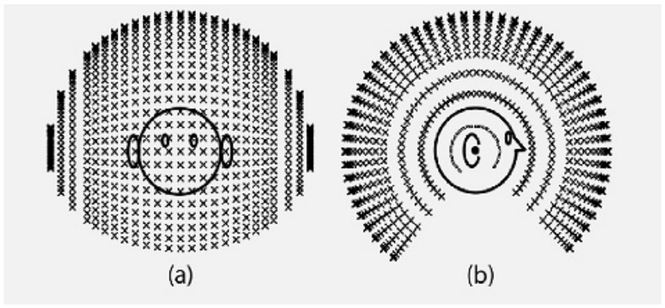


Figure 4: Sound Source Locations a) Front (Azimuthal emphasis) b) Side (Elevation emphasis). Image obtained from [20]

when the receiver is ready. The receiver sends the ready signal in the form of a '.' character which the transmitter acknowledges by flushing out new data to the buffer stream. The receiver waits for the new data and then transmits the '.' character upon data reception. Example code for serial communication between Arduino, MPU 6050 and Python can be obtained in [13].

4.1.3 Orientation Data Deposition

For the system to be able to use the orientation data from the IMU, it must be deposited in a data buffer. We allocate a thread for this very job. Referring to Figure 2, Thread 1 is responsible for the data collection and deposition. Thread 1 thus implements the serial object that reads in the serial data from the serial port and writes it to the shared data buffer. Since the data buffer is accessed by Thread 2 as well, a lock step mechanism for proper RA is implemented. Here, the Queue library comes into the picture which is used to create a semaphore for the access. When the data is written into the buffer, Thread 1 waits for Thread 2 to read it. Once Thread 2 reads it, it sends a 'task_done' signal to Thread 1 which then resumes its data reception and deposition task.

4.2 Audio Rendering System

For processed audio to appear seamless, the processing must be faster than the frame size itself. Referring to Figure 2, we can see that Thread 2 is designated for Audio rendering. The process pipeline for the ARS is huge and cannot be forked because each block relies on the output of the previous one and therefore is cascaded to it. Thus, we need to make changes in the other aspect of the ARS like optimizing the chunk size, array dimensions, operations, etc. to get an optimal performance. In this section, we will go over all the blocks that make up the ARS.

4.2.1 Audio File Access

The audio files that we target for this project are stereo wavefiles with any sample rate and any quantization depth. The complete audio file can be read in by the Stream object as mentioned in section 3.2.2. Small segments of the file can then be individually processed as chunks and then outputted to the sound card for playback. Depending on the sample rate and quantization depth, we can calculate the optimal chunk size which is the amount of data per frame.

The chunk size needs to be calculated judiciously because it governs the performance of the system. A smaller chunk size would mean higher time resolution since the time required to process small chunk would be low. This would thus mean that the orientation data at any instant is associated with a smaller chunk thus increasing time resolution. However, this results in audio stuttering wherein the output audio appears broken or segmented. This also puts a strain on the processor to process that chunk faster which adds to the breaking effect.

A large chunk size solves the problem of breaking because the processor now has ample time to process and write the large chunk to the sound card buffer. However, we lose orientation resolution in time because every reading of the orientation data is associated with a large chunk. Thus, only when the large chunk is completely played out by the sound card, a new orientation reading can be associated to the next chunk.

The trick thus, is to select an optimal sized chunk that properly trades-off the orientation resolution vs the breaking effect and outputs a seamless looking audio chunk. After some tinkering, it was found that a chunk size of 1600 Bytes optimizes the trade-off.

4.2.2 Head Tracking Data

In the previous section we saw the procedure to deposit the orientation/head-tracking data in to the shared buffer. For future references, we will refer to the chunk to be played out or in-process as the current chunk (CC) and the one previous to that as the previous chunk (PC). In order to associate the head tracking data with the CC, we need to retrieve the current/instantaneous head tracking data from the shared data buffer. Thread 1 writes the orientation data to this buffer and waits for Thread 2 to read it. Thread 2 reads the data and stores it in a local buffer to use it later.

4.2.3 Inter-Aural Time Difference Model

Once the orientation data and the audio chunk is in Thread 2, the next step is to start with the actual conditioning of the sound signal. Stereo clips don't inherently express binaural cues (The cues that help us localize and externalize sound sources) and thus, we need to individually model for all the binaural cues. We start off with modeling the Inter-Aural Time Difference (ITD)[21]. ITD represents the perceived time difference in the sound signal due to the physical path difference between ears. This time difference is in the order of milliseconds but is an important cue that helps humans localize sound sources.

The ITD model used for this project has certain assumptions. Firstly, the ITD is modeled for a free-field environment. There is no reflection model that is utilized whilst modeling ITD because that is done separately. The speed of sound is assumed to be 340 m/s. A simple ITD model figure was then constructed to vectorally model differential changes in the azimuth and elevation vectors. This model is shown below in Figure 5.

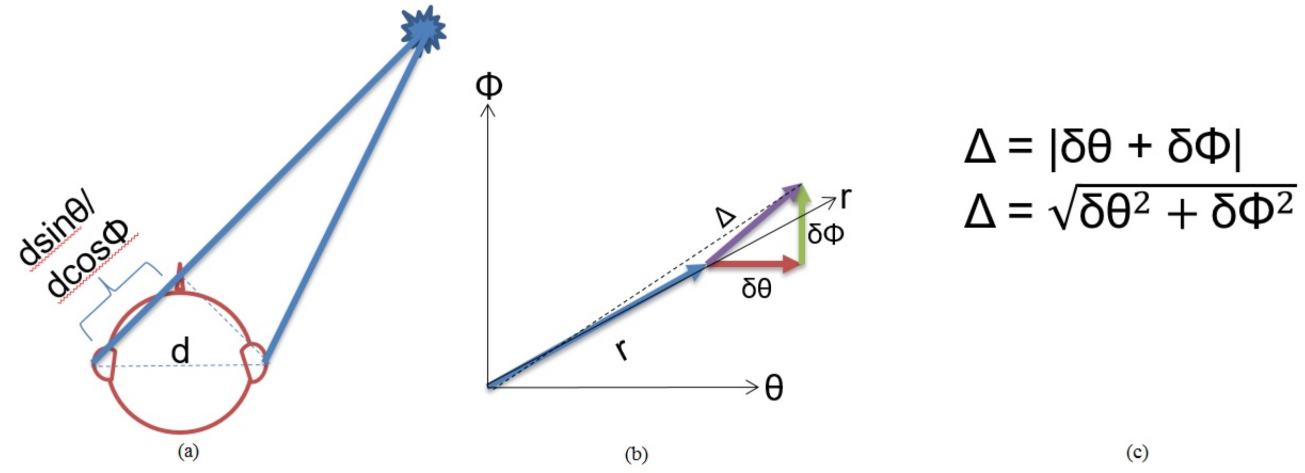


Figure 5: Inter-Aural Time Difference Model

Figure 5.a. showcases a simple model for ITD. As seen from the figure, the left ear receives the same sound after the right ear receives it. We can easily model for the time difference based on the path difference as shown by dsin θ and dcos ϕ where d is the distance between the ears. An average human has a d of about 21-22 cms and a d of 21.5 cms was chosen for this project.

Figure 5.a. doesn't accurately showcase the ITD because it is just a 2D representation. In reality, the ITD would be modeled in a 3D domain which would be done by projecting the actual sound source location onto the azimuth (θ) and the elevation (ϕ) planes and then individually calculating the ITD for each of them. This is OK because the azimuth and elevation basis vectors are orthogonal to each other and thus don't showcase any kind of correlation. The total ITD Δ can then be calculated as the vector addition of the individual azimuth and elevation components as shown in Figure 5.c.

4.2.4 Inter-Aural Level Difference Model

Similar to ITD, the Inter-Aural Level Difference (ILD) helps us localize a sound source. With reference to Figure 5.a., the sound signal that reaches the left ear is not only phase shifted in time relative to the right ear but is also attenuated a little relative to the signal at the right ear because of the path difference. This aids in perceiving sound locations more clearly.

ITD is dependent on the lower spectrum of hearing, whilst ILD depends on the higher spectrum [21]. Zhou X. in [21] has modeled the frequency dependence of ILD and has devised an equation for it.

$$ILD = 1 + (f/1000)^{0.8} \times \sin(\theta) \tag{1}$$

This equation can be used in our model by approximating certain parameters. $(f/1000)^{0.8}$ can be modeled as a straight line with slope 8.703×10^{-4}. A plot of $(f/1000)^{0.8}$ is given below in Figure 6.

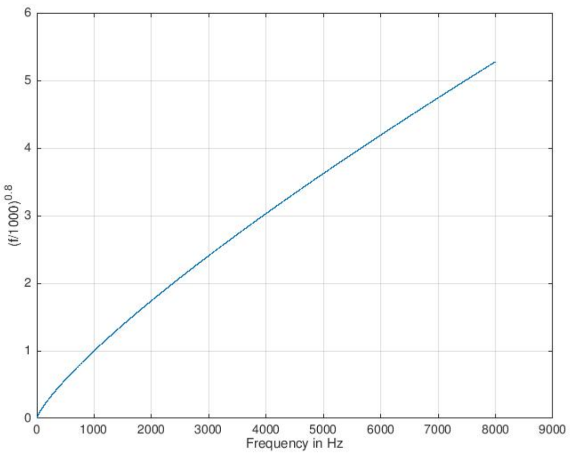


Figure 6: ILD Dependency on Frequency

Each chunk has a dominant frequency associated with it which can be easily extracted using NumPy's fft class. We can then model the multiplier for ILD using the degraded exponential to straight line and $abs(sin(\theta))$. We can then use the result obtained from expression (1) and use it as a multiplier for the calculating relative intensity differences between the right and the left ear. The multiplier can be used according to θ; if θ is greater than 0, the right channel will be multiplied by the ILD multiplier and the left channel will stay the way it was and vice-versa.

4.2.5 Cross Talk Model

In reality, when we hear music or audio played off of speakers, the signal from a channel reaches the ipsi-channel as well the contra-channel. However, this is avoided in headphones as the headphones are placed on the ears. If we want to model speakers on headphones then we need to account for crosstalk that happens between the channels. Cross Talk on headphones for immersive environments is necessary if we want to emulate a live concert or a musical where the actual experience that is to be rendered is due to the loud speakers or multiple sound sources themselves. Figure 7. below showcases the cross talk that takes place when you are hearing sounds from a loud speaker.

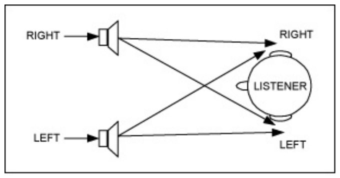


Figure 7: Cross Talk from Speakers. Image obtained from [22]

For this project, cross talk was modeled in a simple way by bleeding the left channel into the right and the right into the left. Model equations are shown below.

$$R = \frac{R + \alpha \times L \times ITD_{left}}{1 + \alpha} \tag{2}$$

$$L = \frac{L + \alpha \times R \times ITD_{right}}{1 + \alpha} \tag{3}$$

Here, we specify a parameter α that governs the percentage of the contra-channel signal to be bled into the ipsi-channel. We also take into account the ITD that the contra-channel signal brings into the picture. Thus, for example, the left channel, post cross talk processing, will have a version of itself plus a ITD delayed version of an α weighted right channel. This is a basic model that takes multiple sources into account and can be extended to accommodate multiple sources.

4.2.6 HRTF/HRIR

The Head Related Transfer Function or Head Related Impulse Response is a function that models the way sound signals scatter off of the human body (torso, face, etc.) and the pinnae to generate individualized binaural cues. Our sense of sound localization is highly dependent on how our body is structured and how our ears are shaped. We grow up getting used to the scattering patterns that our ears produce which highly customizes the way we interpret sounds coming from different directions. Thus, if we listen to sounds that were scattered by someone else's ears and body, we would definitely be confused and disoriented.

HRTFs are those functions that when convolved with dry audio signals can produce a sensation of sound localization. There are many HRTF databases[23] available including the famous KEMAR database from MIT's Media Labs, Sensaura, etc. but these databases individualize the HRTFs too much since they take into account the pinnae of the test subjects. Because of this, when one set of HRTF measurements are used, they don't generalize well to majority of the users and produce front-back and back-front confusions. The CIPIC database[19] by UC Davis however, generalizes well to majority of the population because it excludes the pinnae information from the HRTF model. The CIPIC database captures only the scatter information from the body and the outer ear shape and thus suits better for immersive environment rendering application. For this project, the CIPIC HRTF database is therefore used.

In this project, we aim to localize the sound source at one position (in front of the user) relative to the user's head movement. Thus, we try and negate the user's movement by supplying negatives of the head tracking data (azimuthal and elevation correction) and passing this through the HRTF to generate an output response. The ideal expected output for this is that the sound source should appear localized in the immersive environment relative to the user. A simple example to demonstrate this effect can be showcased in Figure 8.a. given below.

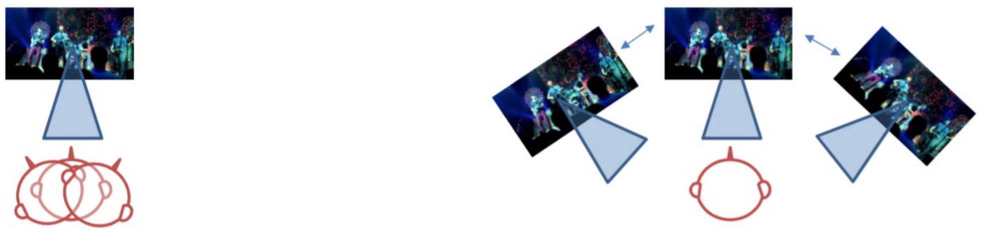


Figure 8: Immersive Environment Rendering Goal a) Localized Source b) Moving Source

4.2.7 Room Impulse Response Model

Room Impulse Response or RIR modeling is very important if we want to simulate the source-receiver combination inside a room. The sound source would emit signals that would reflect off of the boundaries of the room and cause a characteristic reverberation that is easily noticeable. Ideally, we would split any room that we want to model into a grid with a fine resolution and then calculate all the impulse responses of the room for every source vs receiver position combination. This is an exhaustive process to model for room reverberation. Instead of physically collecting measurements, the room dimensions can be simplified to fit into an inscribing rectangle and the Image Method[24] can be used to synthesize the RIR model. [25] showcases a very good example to model for the RIR using the Image Method in MATLAB. Using [25], we can create a data structure to hold the RIR model of a room by containing all the source-receiver combinations in a 3D array in which each slice of the array corresponds to all of the receiver positions for an individual source position as shown in Figure 9. below.

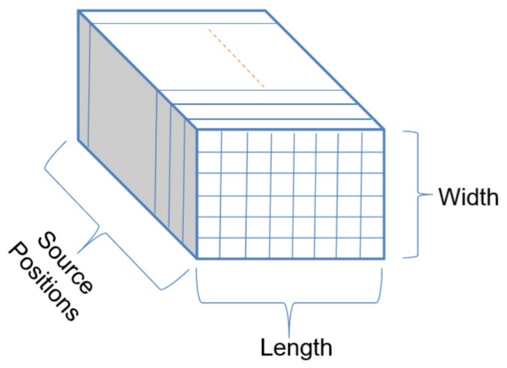


Figure 9: Room Impulse Response Model Data Structure

Depending on the source and receiver positions in the room, appropriate indexes in the RIR array can then be accessed during computation and convolved with the CC. Another thing to notice is that the time-size of the RIR inside each cell in the RIR array must be comparable to the CC size (1600 Bytes). This helps in truncating the convolution output to match the CC size for playback compatibility without losing a lot of the reverberation information. Also, the sample rate of the synthesized RIR Model must match the sample rate of the audio file to avoid any compatibility issues.

4.2.8 Outputting the Audio Chunk

Once all of the cascaded filtering is done, the CC is outputted through the sound card. Before outputting the CC, it is very important to format its dynamic range because multiple convolutions change its dynamic range. The standard format is to scalar divide the samples by $2^{n-1} - 1$ where n is the number of bits per sample. This brings the CC within the standard -1 to 1 range which usually every sound card supports. If not, we can scale it appropriately so that the maximum absolute value in the CC doesn't exceed half of $2^{n-1} - 1$.

5 Performance Evaluation

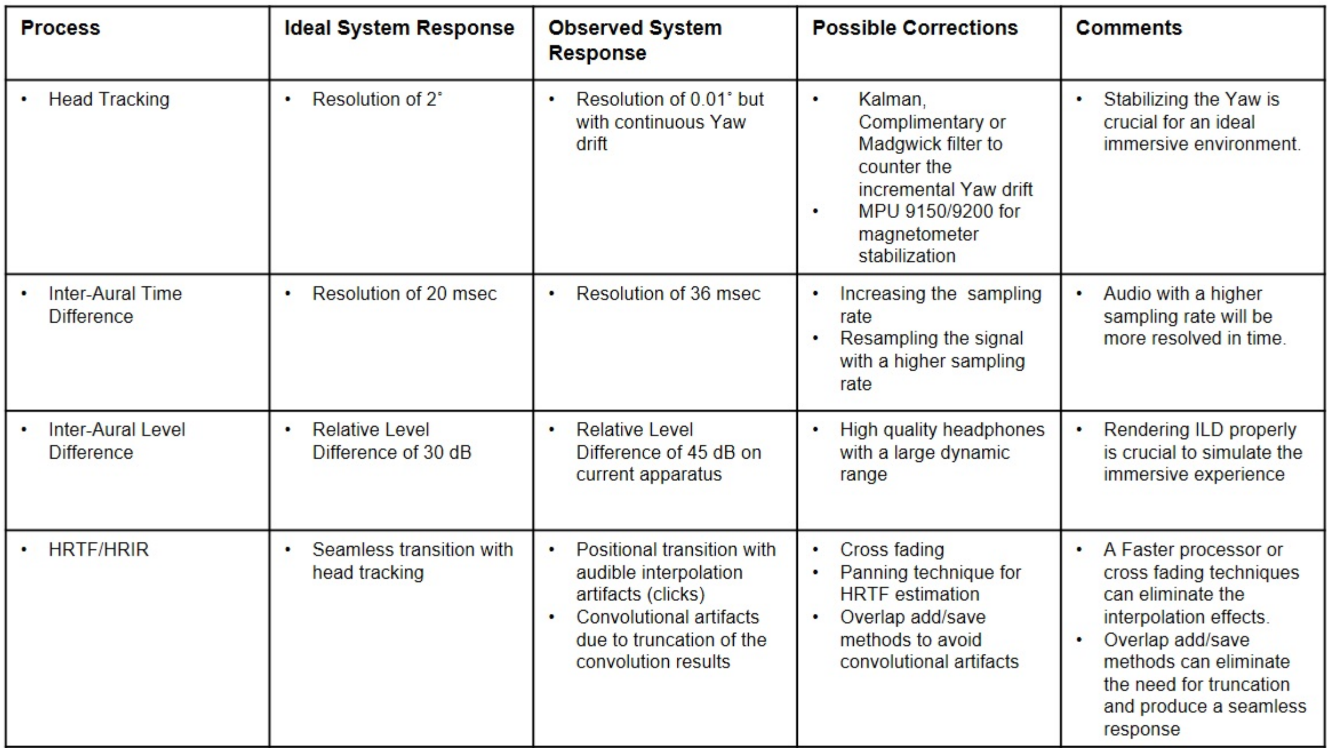
We aim to create a truly immersive environment and thus, it will only be fair if we set our benchmark to be a real life aural experience for performance evaluation purposes. [18] has a nice requirements table that can be used for performance evaluations. We can create a similar table to record our readings and compare to the one given in [18]. Given below is the performance evaluation table for this project in Figure 10.

6 Future Scope & Directions

In order to make the system truly immersive, we need to incorporate more components than what this project deals with. With reference to [18], two major components are missing that are required for a complete immersive experience, viz. Sound Source Decomposition/Virtualization and Air Equalization. Incorporating both of these components is essential because in real-life scenarios, the sounds sources we hear are separated spatially and their responses are transformed because of the medium they broadcast sound in (Air). Thus it is important to consider them if we want to model life-like immersion.

Visual cues drastically affect our sound localization perception as shown in [26]. Thus, having some assertive visual cues from VR/AR technologies or any other synchronized visually active environment can help a lot in rendering a life-like immersive environment.

Immersive Sound Capture has taken off recently due to advancements in the VR/AR/MR industry. Replicating natural sounds is as important as synthesizing them and thus immersive sound capture plays a huge role when it comes to rendering life-like immersive environments. Imagine the endless applications if you could transform your couch into the best seat in the house. If you could capture immersive audio from the acoustic sweet-spot in a concert hall or a stadium along with a 360°video, you could essentially render the AV remotely and make any seat,



Process	Ideal System Response	Observed System Response	Possible Corrections	Comments
• Head Tracking	• Resolution of 2˚	• Resolution of 0.01˚ but with continuous Yaw drift	• Kalman, Complimentary or Madgwick filter to counter the incremental Yaw drift • MPU 9150/9200 for magnetometer stabilization	• Stabilizing the Yaw is crucial for an ideal immersive environment.
• Inter-Aural Time Difference	• Resolution of 20 msec	• Resolution of 36 msec	• Increasing the sampling rate • Resampling the signal with a higher sampling rate	• Audio with a higher sampling rate will be more resolved in time.
• Inter-Aural Level Difference	• Relative Level Difference of 30 dB	• Relative Level Difference of 45 dB on current apparatus	• High quality headphones with a large dynamic range	• Rendering ILD properly is crucial to simulate the immersive experience
• HRTF/HRIR	• Seamless transition with head tracking	• Positional transition with audible interpolation artifacts (clicks) • Convolutional artifacts due to truncation of the convolution results	• Cross fading • Panning technique for HRTF estimation • Overlap add/save methods to avoid convolutional artifacts	• A Faster processor or cross fading techniques can eliminate the interpolation effects. • Overlap add/save methods can eliminate the need for truncation and produce a seamless response

Figure 10: Performance Evaluation Table

anywhere, feel like the best seat in the house. [27] gives an example of a portable 3D sound capturing microphone that uses ambisonics to capture audio all around it.

Haptics is a new area that immersive environments are getting into. Instead of just visualizing or auralizing a rendered environment, we can perceive it better if we could feel the environment. Technologies like SubPak[28] specialize in tactile technologies and wearable jackets that can make you sense the virtual world. 'Its not always about the vocals, sometimes that bass thump in your chest can resonate stronger with your inner self'. Bringing the virtual world closer in every aspect is key in bringing seamless life-like environments to life.

7 Conclusions

In this article we went over the basics of Real-Time Immersive Sound Scaping and the procedure for building an immersive sound scaping system from scratch. Python is a good prototyping language but ideally, a market product must be developed in C/C++. C/C++ has a wonderful 3D audio library called openAL[29] which lets you create, control and render any number of sound sources. It also lets you handle the user orientation, relative velocities of the sources and receivers and simulates the doppler effect for an extra twang.

A good technology that has been emerging now to which everyone is moving to is the Occulus Rift SDK[30] which provides a great platform for integrating immersive 3D worlds including both 3D Audio and Video. The occulus SDK has its own set of proprietary HRTFs that generalizes to a large population thus alleviating the localization problem.

It is easy to model a 3D sound scaping system assuming certain things, but assuming them whilst targeting the global market would be a bad idea. For the market, it is better to either stick with existential SDKs like Occulus or OpenAL. If there is a need to make one from scratch, then all the major parameters must be modeled accurately without any assumptions and must be cross-validated for any discrepancies in generalizations for majority of the population. Psychoacoustics plays a major role in modeling immersive environment parameters and thus should be used wisely, effectively and judiciously.

References

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

[2] [Online] http://sound.media.mit.edu/resources/KEMAR.html

[3] [Online] http://realspace3daudio.com/technology/#techoverview

[4] [Online] https://www.cnet.com/products/sony-mdr-zx110nc-headphones/specs/

[5] [Online] https://www.arduino.cc/en/Main/ArduinoBoardUno

[6] [Online] https://www.invensense.com/wp-content/uploads/2015/02/MPU-6000-Datasheet1.pdf

[7] [Online] https://en.wikipedia.org/wiki/SciPy

[8] [Online] http://www.numpy.org/

[9] [Online] https://people.csail.mit.edu/hubert/pyaudio/docs/

[10] [Online] https://docs.python.org/2/library/wave.html

[11] [Online] http://pyopengl.sourceforge.net/documentation/

[12] [Online] http://www.pygame.org/docs/tut/PygameIntro.html

[13] [Online] https://github.com/mattzzw/Arduino-mpu6050

[14] [Online] https://docs.python.org/2/library/threading.html

[15] [Online] https://docs.python.org/2/library/queue.html

[16] [Online] https://www.youtube.com/watch?v=EvbA3qVMGaw

[17] [Online] https://diyhacking.com/arduino-mpu-6050-imu-sensor-tutorial/

[18] [Online] https://www.sigport.org/sites/default/files/SPM15slides_Natural%20Sound%20Rendering%20for%20Headphones.p

[19] [Online] http://interface.cipic.ucdavis.edu/sound/hrtf.html

[20] [Online] http://interface.cipic.ucdavis.edu/data/doc/CIPIC_HRTF_Database.pdf

[21] [Online] https://en.wikipedia.org/wiki/Sound_localization#ITD_and_IID

[22] [Online] https://www.maximintegrated.com/en/app-notes/index.mvp/id/4632

[23] [Online] https://en.wikipedia.org/wiki/Head-related_transfer_function

[24] J.B. Allen and D.A. Berkley, "Image method for efficiently simulating small-room acoustics," Journal Acoustic Society of America, 65(4), April 1979, p 943.

[25] [Online] https://www.audiolabs-erlangen.de/fau/professor/habets/software/rir-generator

[26] C. Kyriakakis, "Fundamental and technological limitations of immersive audio systems," in Proceedings of the IEEE, vol. 86, no. 5, pp. 941-951, May 1998.doi: 10.1109/5.664281

[27] [Online] http://www.synthtopia.com/content/2016/05/18/zoom-h2n-update-lets-you-record-immersive-audio-for-virtual-reality/

[28] [Online] https://shop.subpac.com/products/subpac-m2-wearable

[29] [Online] https://www.openal.org/

[30] [Online] https://developer.oculus.com/

[31] [Online] http://pythonhosted.org/pyserial/

[32] [Online] http://ircamtools.com/

[33] [Online] http://www.waves.com/nx