REVERBERATION ROOM

Final Report

MAY 11, 2021

CANDIDATE NUMBER: 181334
University of Sussex – Computing Sciences with Artificial Intelligence

1. Statement of Originality

This is to certify the content of this dissertation is, to my best knowledge, a product of my own labour except where indicated clearly. This report is submitted as a requirement for the Computer Sciences & Artificial Intelligence BSc (Hons) degree at the University of Sussex.

This work is based 'Reverb Room Simulator' project first conceptualized by Dr Kingsley Sage and has been continually developed under guidance throughout my third and final year at the university.

2. Acknowledgements

I would like to thank Dr Kingsley Sage for his immeasurable contribution, continued patience and thorough guidance in supervising this project.

3. Table of Contents

4. Summary

This report details the process of designing and implementing an artificial reverberation chamber from the ground up with the goal of emulating a truly immersive binaural listening experience when listening with stereo headphones. The ray tracing approach enables a realistic three-dimensional spatial depiction through development of a realistic physical room via implementation of a reverberation chamber powered by ray tracing concepts enabling effective emulation of the propagation of sound from a point sound source to realistic microphones with a selection of pickup patterns.

To listen to the reverberation effect, please refer to the Audio Samples Folder in the submission file.

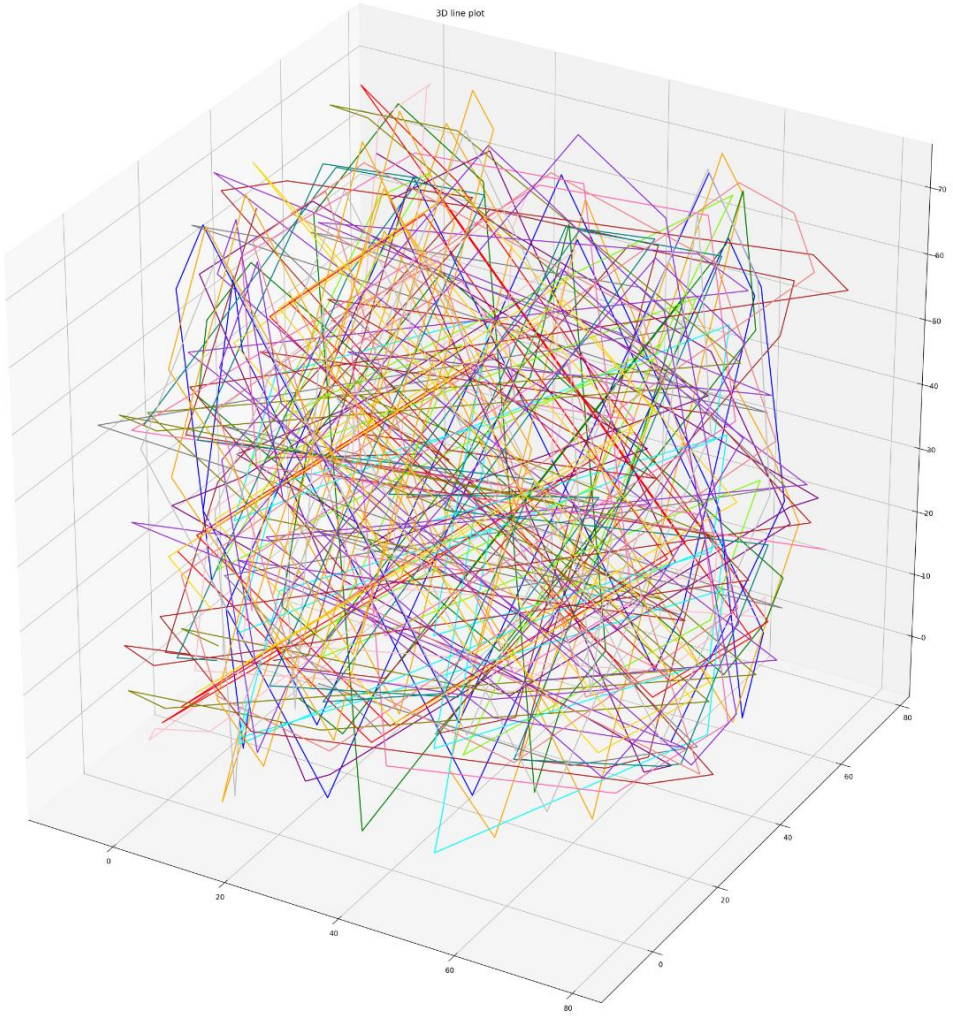


FIGURE 1 RAYS MODELLING PROPAGATION OF SOUND THROUGH SPACE

This project also details building a front-end GUI using the JUCE framework and windows executable application to allow the user reasonable abstraction to apply a reverberation effect, and full control of the algorithm used to construct the reverberation effect.

This report details many of the key features and key design decisions as to the specific technology stack deployed along with key aspects of design, such as the trade-off between runtime complexity when realizing realistic emulation of reverberation.

5. Introduction

Artificial reverberation is widely used in music production. Its purpose is to make sound appear as it was generated in a realistic physical space. This can be achieved through modelling how sound propagates around a room.

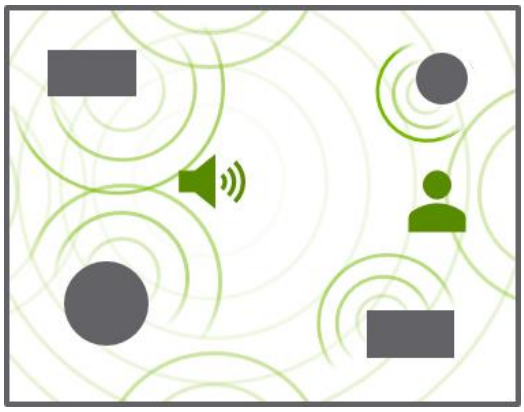


FIGURE 2 PROPAGATION OF SOUND MOVING IN SPACE [1]

5.1. Objectives

The project aims to create a reverberation room software to allow greater flexibility in mixing raw microphone input by emulating a selection of reverberation chambers and microphone placements to emulate realistic sounding reverberation using ray tracing techniques.

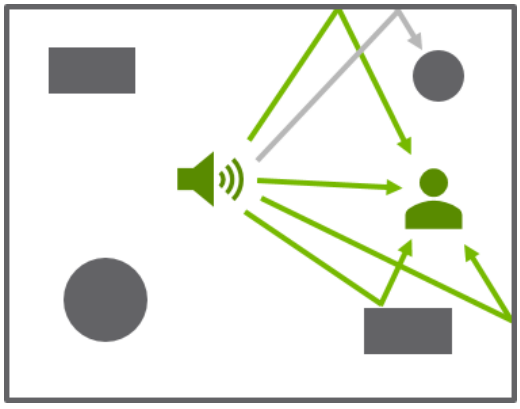


FIGURE 3 RAY TRACED AUDIO BY MODELLING PROPAGATION OF SOUND MOVING IN SPACE [1]

The target use case of this is to enable audio engineers tools to model reverberation through implementation of a customizable ray traced reverberation chamber. We aim to create this with an appropriate level of abstraction to give the users sufficient control to enable manipulation of the properties of reverberation without having any programming knowledge.

A key use case for this is to allow greater flexibility in mixing raw microphone input by emulating a selection of reverberation chambers and microphone placements to emulate realistic sounding reverberation within a 3D space.

One of many examples uses of this could be to correct a balance issue in the mix where the drummer plays one instrument (e.g. kick drum) too quietly, in the room mics and overheads, this cannot be corrected easily. Real room mics could be supplemented with artificial room mics from the reverberation room to boost the sound of the quiet instrument (in this example, kick drum) that will

allow greater flexibility in mixing room microphones by allowing the room mic mix to be corrected and enhanced to a greater extent than was previously possible.

This will also enable a full room microphone mix when recording with limited microphones where no room mics are available. (A standard drum mic set up is 2 overheads, mic on each tom or a single ribbon microphone for each tom drum, a snare mic or 2 (top and bottom of snare), kick drum microphone)

This could simply enable a one microphone setup to have a full room mix. For example, one unidirectional SM57 microphone against a guitar amp, would be able to artificially create a full room mic setup to pickup the reverberation response for user defined room proportions and a selection of microphone types, placement, and 3D orientation of the microphone and relevant pickup patterns.

6. Professional Considerations

This project consists of work produced with, to my best knowledge, strict adherence to the Code of Conduct published by BCS – The Chartered Institute for IT [2]. The code of conduct has been familiarized and strictly adhered to my best knowledge. The following statement has taken from the official BCS website and modified to apply this project to ensure it has been thoroughly understood and applied to thoughtfully to this project.

PUBLIC INTEREST:

> *I shall have due regard for public health, privacy, security and wellbeing of others and the environment. I shall have due regard for the legitimate rights of third parties by using only licence permitting third party material and referencing it clearly using Harvard style referencing. Any professional activities occur without discrimination on the grounds of sex, sexual orientation, marital status, nationality, colour, race, ethnic origin, religion, age or disability, or of any other condition or requirement. This project promotes equal access to the benefits of IT and seek to promote the inclusion of all sectors in society wherever opportunities arise.*

PROFESSIONAL COMPETENCE AND INTEGRITY

> *I shall only undertake to do work or provide a service that is within my professional competence and NOT claim any level of competence that I do not possess. This project will allow me to develop my professional knowledge, skills and competence on a continuing basis whilst maintaining awareness of technological developments, procedures, and standards that are relevant to my field. In doing so, I shall ensure that I have the knowledge and understanding of relevant legislation and I shall comply with such legislation, in carrying out any professional responsibilities. I both respect and value alternative viewpoints and seek, accept and offer honest criticisms of work. I shall avoid injuring others, their property, reputation, or employment by false or malicious or negligent action or inaction and reject and will not make any offer of bribery or unethical inducement.*

DUTY TO RELEVANT AUTHORITY:

> *I shall carry out my professional responsibilities with due care and diligence in accordance with the University of Sussex requirements while exercising my professional judgement at all times. I shall actively seek to avoid any situation that may give rise to a conflict of interest between me and my relevant authority. I accept professional responsibility for my work and this project will be entirely product of both my own work and input from my technical supervisors. I shall NOT disclose or authorise to be disclosed, or use for personal gain or to benefit a third party, confidential information except with the permission of the University of Sussex, or as required by legislation. I will NOT misrepresent or withhold information on the performance of products, systems or services (unless lawfully bound by a duty of confidentiality not to disclose such information) or take advantage of the lack of relevant knowledge or inexperience of others.*

DUTY TO THE PROFESSION:

> *I accept my personal duty to uphold the reputation of the profession and will not take any action which could bring the profession into disrepute. I seek to improve professional standards through participation in their development, use and enforcement. I shall do everything in my power to uphold the reputation and good standing of BCS, The Chartered Institute for IT and act with integrity and respect in my professional relationships with all members of BCS and with members of other professions with whom I work in a professional capacity. I shall notify BCS if*

convicted of a criminal offence or upon becoming bankrupt or disqualified as a company director and in each case give details of the relevant jurisdiction and I shall encourage and support fellow members in their professional development.

7. Reverberation Background Research & Literature Review

7.1. Background Research

Key principals of realistic reverberation:

a. Direct sound
b. Early reflections.
c. Late reflections.
d. Gradual build-up.
e. Reverb Tail.
f. Frequency response.
g. Strict adherence to the law of conservation of energy.

Reverberation is the prolongation of a sound through resonance. Reverberation time is the measure of time for the sound to decay completely in an enclosed area after the source of the sound has stopped.

A formula was first devised and developed by Wallace Clement Sabine[3] allowing the reverberation time for a room to be calculated.

T = reverberation time (s) , c = sound speed, V = volume and S = total interior surface area of the enclosure in question respectively, αa = averaged absorption coefficient of the enclosure,

αi = individual absorption coefficient of each subsurface, Si.

$$T = 13.8\frac{4V}{\mathrm{c\,S}\,\alpha_a}, \text{ with } \alpha_a = \frac{\sum_i \alpha_i S_i}{\sum_i S_i},$$

FIGURE 4 [3]

Henry Eyring pointed out the need for a more general formula for enclosed spaces with highly absorptive conditions.[3]

$$T = 13.8\frac{4V}{-\mathrm{c\,S}\ln(1-\alpha_a)}.$$

FIGURE 5 [3]

This forms the fundamental information required for non-ray tracing powered reverberation programs to model reverberation in each room size.

7.2. Existing Systems

Existing reverberation algorithms examples:

FIGURE 6 ORILRIVER REVERB VST PLUGIN [29]

Pros: - Plenty of controls to adjust characteristics of reverberation.

- Aesthetic GUI.

- Good choice of different reverberation chambers.

- Very low CPU requirements.

Cons: - Limited control of sound source location and listener location so does not create an immersive effect as if you were in the room.

-

FIGURE 7 EVENTIDE VTERB [30]

Pros: - Recreation of a real vocal production techniques using ambient microphones different distances from the sound source.

- Full control of microphone placements

- Very good quality sound.

Cons – CPU intensive.

- Does not allow emulation of different microphone types.

- Does not model propagation of sound with ray tracing techniques.

FIGURE 8 OCEAN WAY STUDIOS PLUG-IN [31]

Pros – Great selection of UI elements.

- Good selection of microphone types.

- 3D positional stereo reverberation.

Cons – very expensive.

- Limited emulation of the propagation of sound.

7.3. Ray Trace Reverberation

We can emulate reverberation accurately by modelling sound propagation using a ray tracing approach. To do this we will first need to calculate the speed of sound in air which can be given by the formula:

V = speed (m/s),Tc = Temperature (°C).

$$V = 331 + 0.59\ Tc$$

Given that a reasonable room temperature is 20 °C. We calculate the speed of sound in air for our model to equal:

$$V = 331 + 0.59 * 20$$

$$V = 342.8\ m/s$$

$$V = 343\ m/s\ (\text{rounded to nearest integer})$$

We can use V to calculate the time taken for the ray to reach the microphone from the source. This is the delay for the one sample to reach the microphone.

We know that a pressure wave otherwise known as a sound wave loses amplitude with distance following an inverse square law, so we can calculate the amplitude of the ray with the distance it has

travelled. We can also define how much the wave will decay upon reflecting off any given surface. This can give us an output amplitude.

Here we can see how sound pressure or volume decays over time.

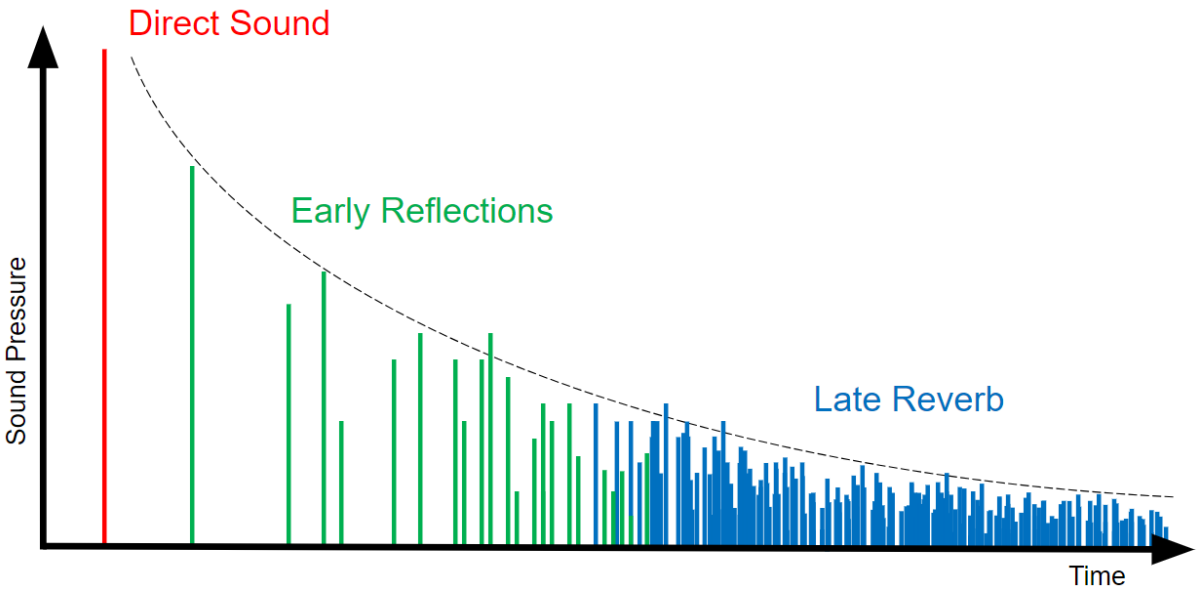


FIGURE 9 GRAPH SHOWING SOUND PRESSURE OVER TIME WITH LABELLED DIRECT SOUND AND EARLY REFLECTIONS [4]

We project rays in every direction from a point sound source. Each ray will be a class object storing information on its current location, sample number of the audio file, distance travelled and current amplitude.

7.4. Wrapper Formats

As we will be targeting this project for audio engineers and musicians it is important to ensure the target users will be able to utilize the software through use of a graphical user interface(GUI). In order to do so, we need to select a wrapper format or frameworks to allow the same source code to compile and run on as many platforms as possible whilst having a clear graphical user interface for the users to use the program.

FIGURE 10 AUDIO GRAPHICAL USER INTERFACE EXAMPLE [16]

After broad research into DAW plugin wrapper formats, three potential solutions have been overviewed.

JUCE [8] allows the same C++ source code to compile and run identically on Windows, macOS, Linux, and even mobile platforms as a plugin, mobile application and standalone desktop or even web applications. In this project, we are aiming to build a standalone Windows desktop application.

There are free personal and free education licences. We can use the Educational use licence, which has no revenue or funding limit, and it is free to use. On top of this there are many example projects and a multitude of clear documentation and tutorials on the website. The only notable downside discovered at this stage is the Splash Screen watermark displaying 'Made with JUCE' with the free versions.

iPlug 2 [9] allows the same C++ source code to compile and run identically on Windows, Mac and Linux. This includes any platform that can utilize the VST2, VST3, AUv2, AUv3, AAX (Native) and the Web Audio Module (WAM) plug-in APIs. This is very similar to JUCE with marginally less support for the added benefit of not have the watermark on the educational license version. On their official page, they state that they do not consider iPlug 2 to be production ready. This means there are likely some problems with the software. For the only foreseeable benefit of choosing iPlug2 [9] over JUCE [8] is the lack of watermark, at the cost of cross platform support, and an increased risk of encountering an unexpected software problem that could interfere with the project progression, the preference currently remains JUCE [8].

jVSTwRapper [10] is another option to create the plugin however this time using the language of Java. This has the added benefit of allowing relatively easier Graphical User Interface (GUI) programming as C++ is known to be relatively cumbersome when creating GUI windows.

However, the alternative C++ approach, JUCE [8], has a huge plethora of information available on creating GUIs. Overall, this should result in a similar development time for both approaches. As Java is an interpreted language, it will inherently lead to a slower runtime than C++ as it requires a Java virtual machine to interpret the code at runtime whereas C++ is compiled directly into machine code resulting in more efficient runtime. Should the extended objective of optimizing for CUDA acceleration [14] be

fulfilled, we shall need to use the language of C or C++. This is hugely important to optimize for runtime due to the computationally intensive nature of the project. This necessitates C++, thus ruling out jVSTwRapper.

After careful evaluation of the wrapper formats; JUCE [8], iPlug 2 [9] and jVSTwRapper [10], we conclude that JUCE [8] is the best choice of wrapper format as it maximize the chances completing the system objectives to the greatest possible extent.

7.5. User Research

This reverberation software project is designed with the aim of providing a tool for Musicians and or Audio Engineers to allow them to emulate realistic reverberation audio effects within modern digital workstations or as a standalone application.

A common scenario that engineers and musicians alike find themselves in is having to balance the live recording tracks from close microphones with ambient room microphones to create a balanced and great sounding mix capturing the acoustics of the room.

Usually, if one instrument or voice is played too quietly it can easily be partially corrected by adjusting the gain of the close microphone of the instrument that needs correcting. However, this is not easy to correct in the room mix as it picks up the ambience of the entire room. [6]

An existing solution to this, even when a very realistic sounding reverberation plugin such as Quantec [5] and OrilRiver Reverb [29] is deployed for this very purpose, it can be tough to match this reverberation sound to that captured in the room mix. This is because room microphones pick up sound relative to the location of the sound source and this reverberation applied is one dimensional. This is in part caused by them not emulating different microphone types nor placements within the reverberation chamber alongside a realistic propagation of sound emulation.

Another issue is the Hallelujah effect is a problem the traditional non-ray tracing reverberation algorithm first observed by Quantec in 2010 [5]. They solved this by increasing the attack of the note or sound the longer the note plays and then delaying the launch of this sound to produce a realistic reverberation effect. However, there are some inaccuracies in this process. From the 2010, this was the only reverberation software that identified and took steps to solve this issue. By modelling sound propagation via a ray tracing model of sound propagation, along with strict adherence to the physical laws governing reverberation of pressure waves, we can solve the hallelujah effect naturally without excessive corrections needing to be applied to achieve a greater level of realism in the reverberation effect. This is achieved by simply firing rays for every sample of the sound, each ray containing the bit information of the sample which is then reconstructed into audio data at the microphones.

Some solutions for this one-dimensional reverb problem have been solved in applications such as Eventide Vterb [30] and Ocean way studios plug-in [31] which form phenomenal solutions for stereophonic reverberation as they allow for directional reverberation effects with a selection of microphone types.

We look to take this one step further and achieve a similar effect with the key innovation of utilizing ray tracing technology to model to the propagation of sound and generate the stereo audio output.

On top of this, in many budget recording setups, room microphones are not available to the musician or audio engineer due to having access to a limited number of microphones. This software will enable users to emulate additional room microphones and apply a realistic reverberation effect with utilization

of a ray-traced model of the propagation of sound along with a selection of custom microphone types to serve as an alternative to Eventide Vterb[30] and Ocean was studios plug-in[31].

One goal of this is to enable the budget studio the ability to emulate a professional stereo, multi microphone, recording in a variety of acoustically treated rooms with very limited recording equipment.

8. Requirements

8.1. System Functional Requirements

To ensure the user requirements can be met, we give rise to the following system requirements.

8.1.1. Core System Requirements

For this project to be considered successful, the following core system requirements must be met.

1) System allows audio upload and playback to apply ray tracing reverberation to an audio track in an offline manner and then be able to render the audio track with the reverberation effect applied using the CPU.
    a. The user can select from a selection of sample rates and bit depths for the output audio.
    b. The audio effect will monoaural and generated from a single listening point.
    c. Reverberation room shall be 2 dimensional.
    d. User can upload an audio track for processing within the software.
    e. Playback the uploaded track.
    f. The user should be able to render the audio track with the reverb effect applied.
2) System should form a digital audio workstation plugin or standalone application.
3) Create several pre-configured microphone placement and room type configurations.
4) Implement a GUI and code to allow user to adjust:
    a. Sample rate of reverberation audio track forming the number of rays traced per second.
    b. The bit depth of each sample.
    c. Gain.
    d. Volume.
    e. pre-configured microphone placement and room type.

8.1.2. Extended Requirements

Should time permit, and after all the core requirements have been met, the following extended requirements can be met. For these extended system requirements to be met, after all the core system requirements have been met; the system shall:

1) The system allows user to manually add and relocate existing microphones and sound sources within the 3D space.
2) Create a class for each microphone type:
    a. Cardioid
    b. Omnidirectional
    c. Figure-8
3) Enable Directional Reverberation (Stereo) based on where the microphone/microphones are placed within the room relative to the sound source.
4) The reverb algorithm should be improved to emulate the following key features to creating reverberation:
    a. Direct sound
    b. Early reflections.
    c. Late reflections.
    d. Gradual build-up.
    e. Reverb Tail.
    f. Frequency response.

g. Strict adherence to the law of conservation of energy.
h. “Physical feeling of pressure with extremely small rooms.” [5]
i. “The inherent subsonic tremble of huge rooms.” [5]

5) Research optimized runtime utilizing CUDA acceleration technology [14][15] to speed up render times for rendering out the audio track into a selection of audio format with options of both sample and bitrates.

8.1.3. Fallback System Requirements

The system shall:

1) Exist as a Plugin or standalone in any format that allows a user to upload an mp3 audio track and apply a basic reverberation effect with a monophonic output.
2) Basic controls forming a GUI:
    a. Sample rate of reverberation audio track forming the number of rays traced per second.
    b. The bit depth of each sample.
    c. Stereo or mono input audio file and outputting a monophonic audio track.
    d. Gain.
    e. Volume.
    f. pre-configured microphone placement and room type.
3) Select from a limited selection of microphone placements and room types.

9. Project plan

Here is a program evaluation and review technique (PERT) chart forming a project plan. During the implementation stages, an agile approach should be deployed detailing sprint progress along with testing. During the implementation stage, a GitHub Repository shall be utilized to help document the development process

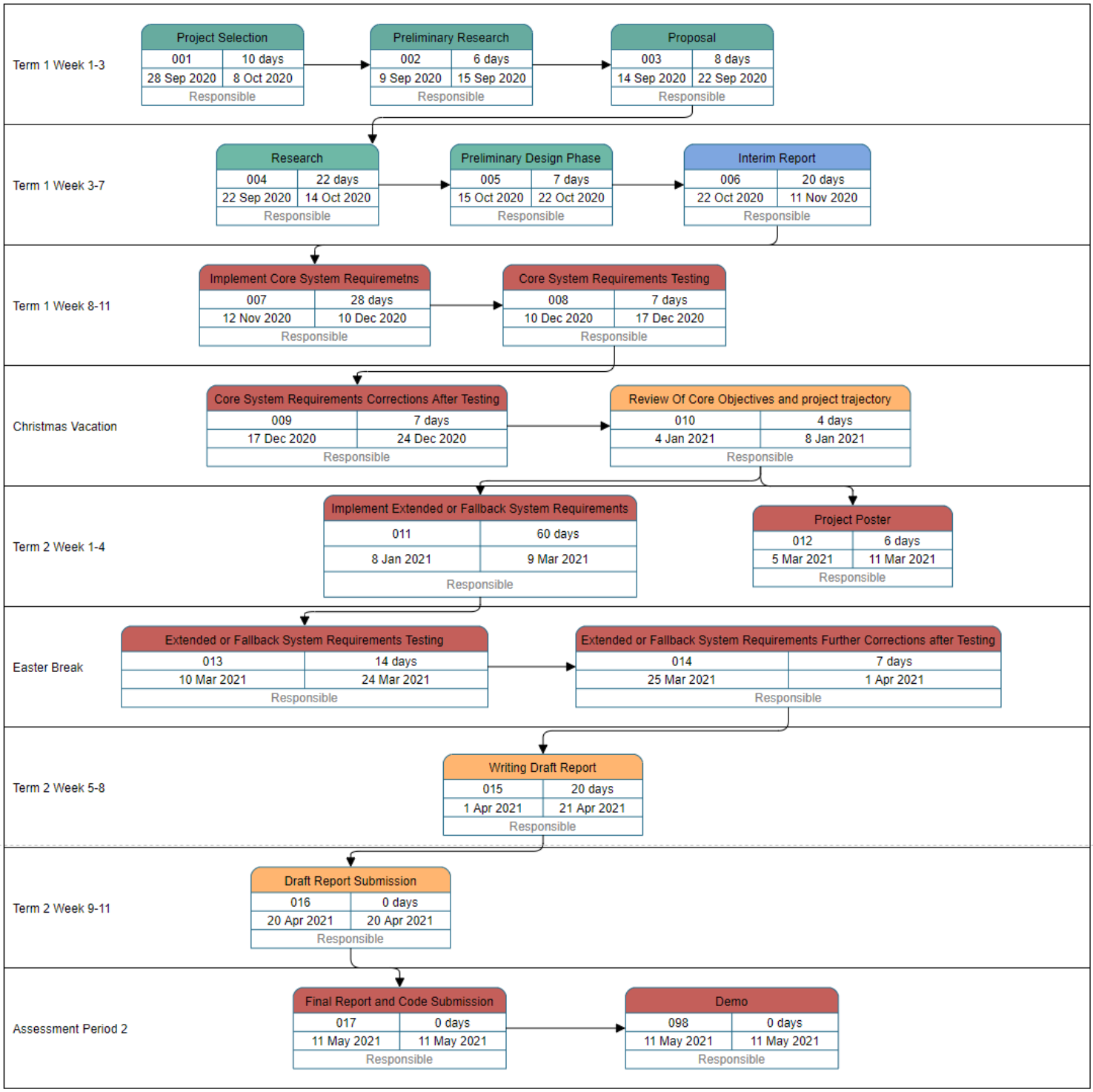


FIGURE 11 PERT CHART DETAILING OVERVIEW OF PROJECT PLAN

10. Design/Implementation

10.1 Technology Stack:

- C++ 14.

- Projucer JUCE [8].

- Adam Stark Audio Library [32].

- Butterworth Filter class [26].

10.2 Design process

The project has been made following an iterative design process focusing on one objective at a time using the compiled language C++ 14, along with external libraries such as AudioFile by Adam Stark [32], the Butterworth Filter Class [26] and the JUCE library [8] to form the wrapper format. We use the compiled language of C++ is well suited as it produces machine language than can directly understood by the system resulting in a highly efficient runtime.

The use of a wrapper format enables multiple data streams to be embedded into an application programming interface such as: VST2, VST3, AUv2, AUv3, AAX for use in any digital audio workstations that supports any of the supported application programming interface. However, in this project, we used the JUCE framework as the wrapper format to create a GUI to be a standalone windows application. We reviewed their initial documentation and loaded up the simple audio player template project. We then manipulated this to our suit our needs via implementation of an external library, AudioFile.h [32], to control the wav/aiff file processing.

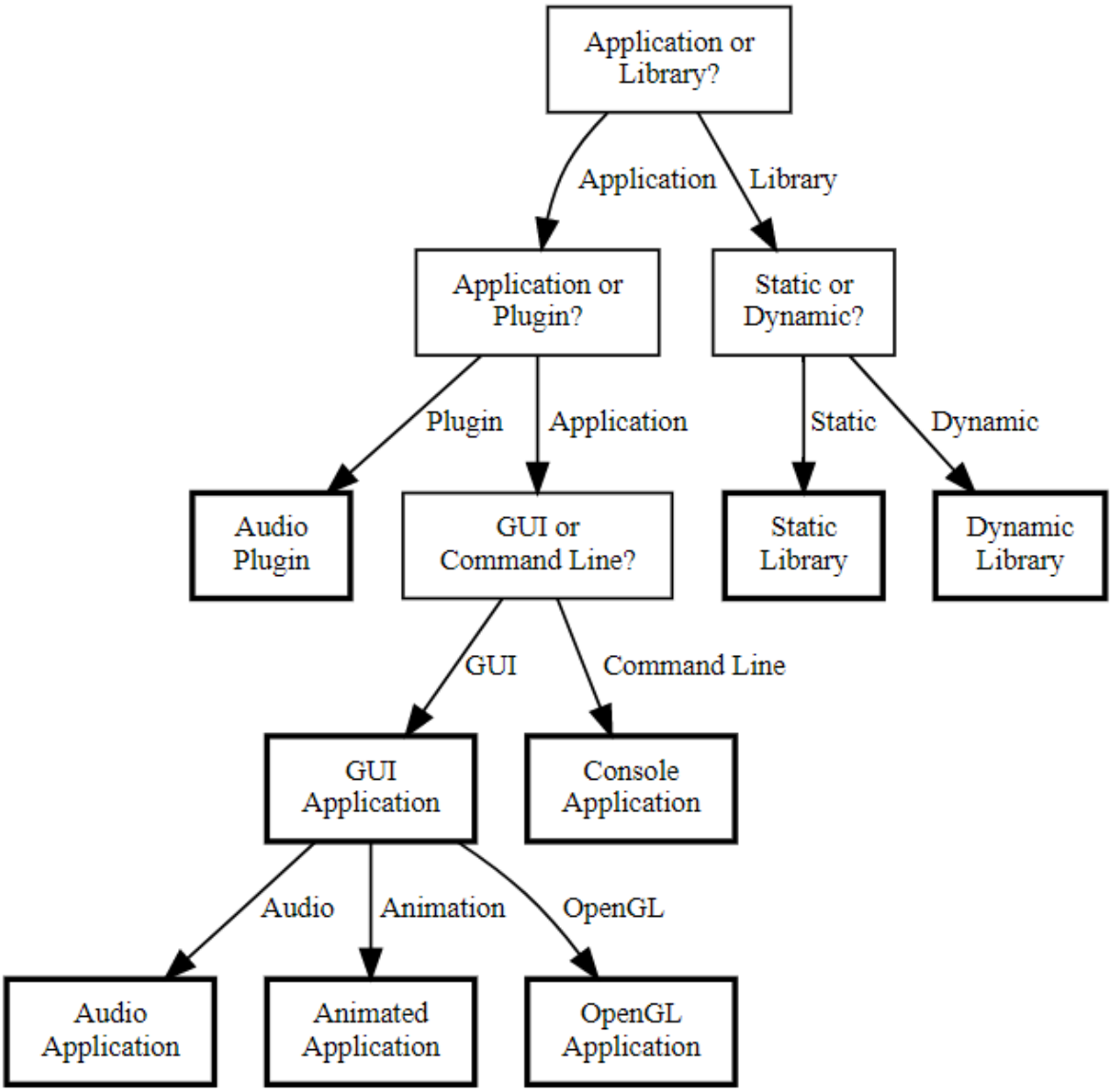


FIGURE 12 JUCE PROJECT TEMPLATE LOGIC [17]

Note: We retrospectively found this documentation highlighting that the Audio Player was not in Audio Plugin format as the official documentation mislead us. As a result, we built our project as a GUI Audio Application. This means the application is a standalone GUI Audio Application built to run on Windows compromising on cross platform support/DAW integration. *Please refer to evaluation and future work for directions and more information.*

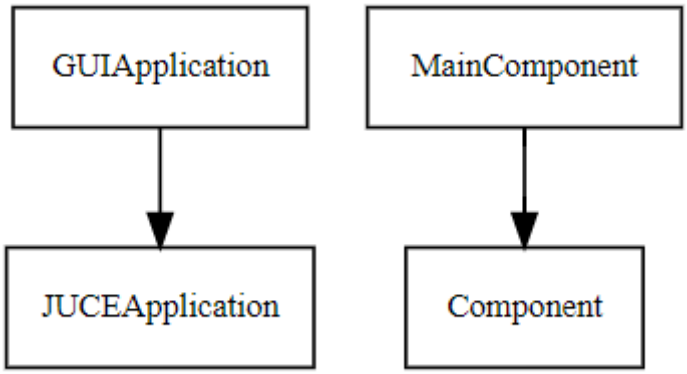


FIGURE 13 THE GUI APPLICATION BASE CLASSES [17]

In this project, users will be able to apply ray tracing powered reverberation to an audio track. The reverberation is achieved by modelling the propagation of sound moving in within a virtual acoustic space or reverberation chamber by modelling propagation of sound moving in space by tracing rays.

The design of this ray tracer algorithm takes careful account of the balance between max realism and unnecessary computational run-time complexity.

Initial High-Level Conceptualization of Core Ray Tracer Design

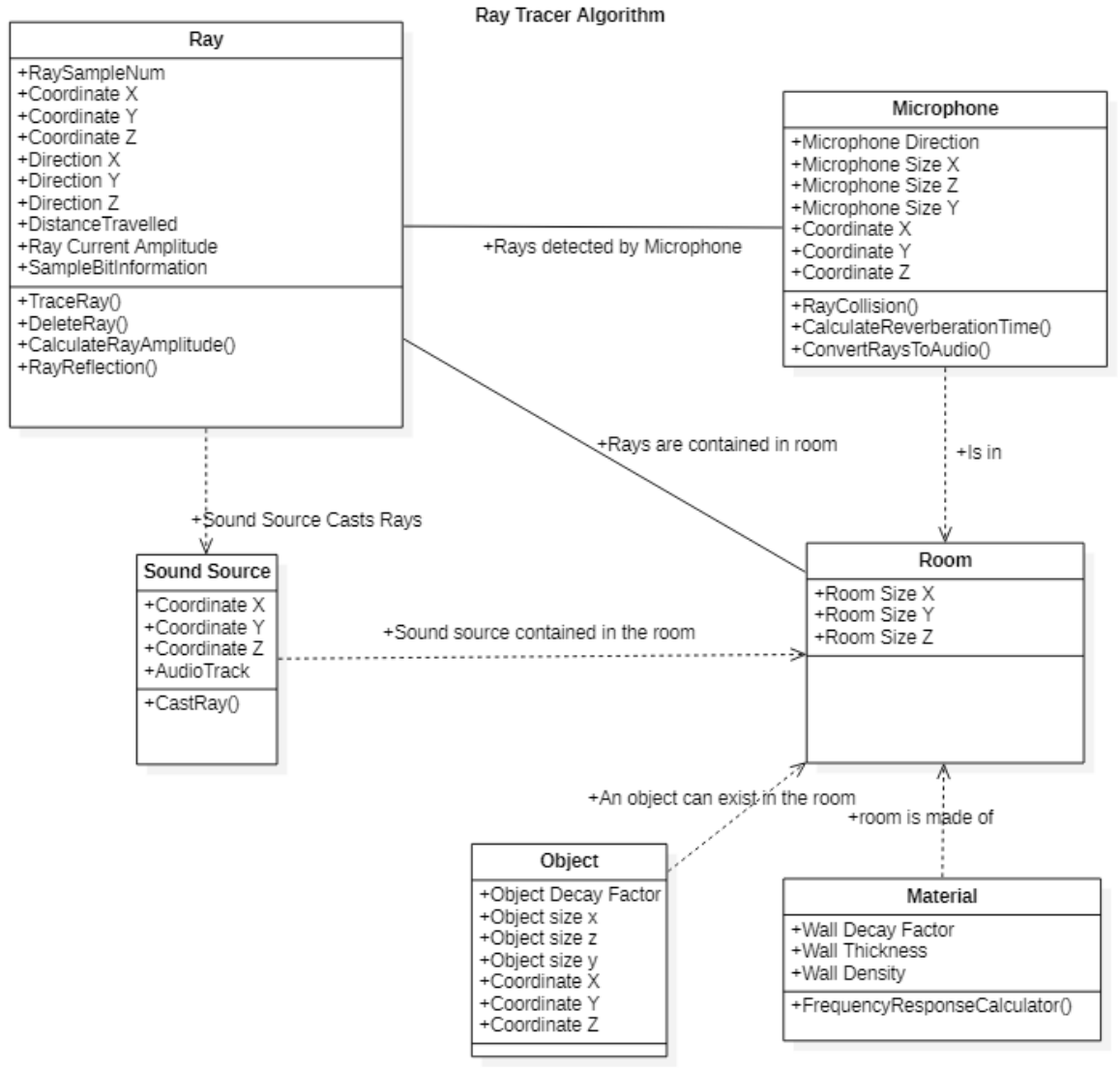


FIGURE 7 PRELIMINARY RAY TRACER DESIGN UML

We define the room dimensions as a 3D matrix, in this example, we will give the room dimensions as label X, Y and Z.

Now the ray will continue to move in the direction in the matrix as specified by the ray direction variable in the ray trace class updating the current coordinates and the Ray Distance Travelled as it goes.

Should the ray current coordinates x, y or z exceed X, Y or Z respectively, the ray should change direction in the axis where coordinate x, y or z exceeded X, Y or Z. This allows the angle of incidence to equal the angle of reflection. When the ray reflects from a wall, a method call takes place in the material object to calculate the frequency response of the wall that will be induced on the ray sample bit information stored in the ray object. The ray amplitude will reduce by the wall decay factor. On top of this, the ray amplitude or sound pressure, also known as the intensity of the sound decays with distance following an inverse square law with distance. This is accounted for while tracing the individual ray, the amplitude will not decay with distance, rather the number of bounces before being pruned from the all rays vector as the number of rays in existence hits the user defined fixed cap of rays in existence to achieve reduced run time complexity.

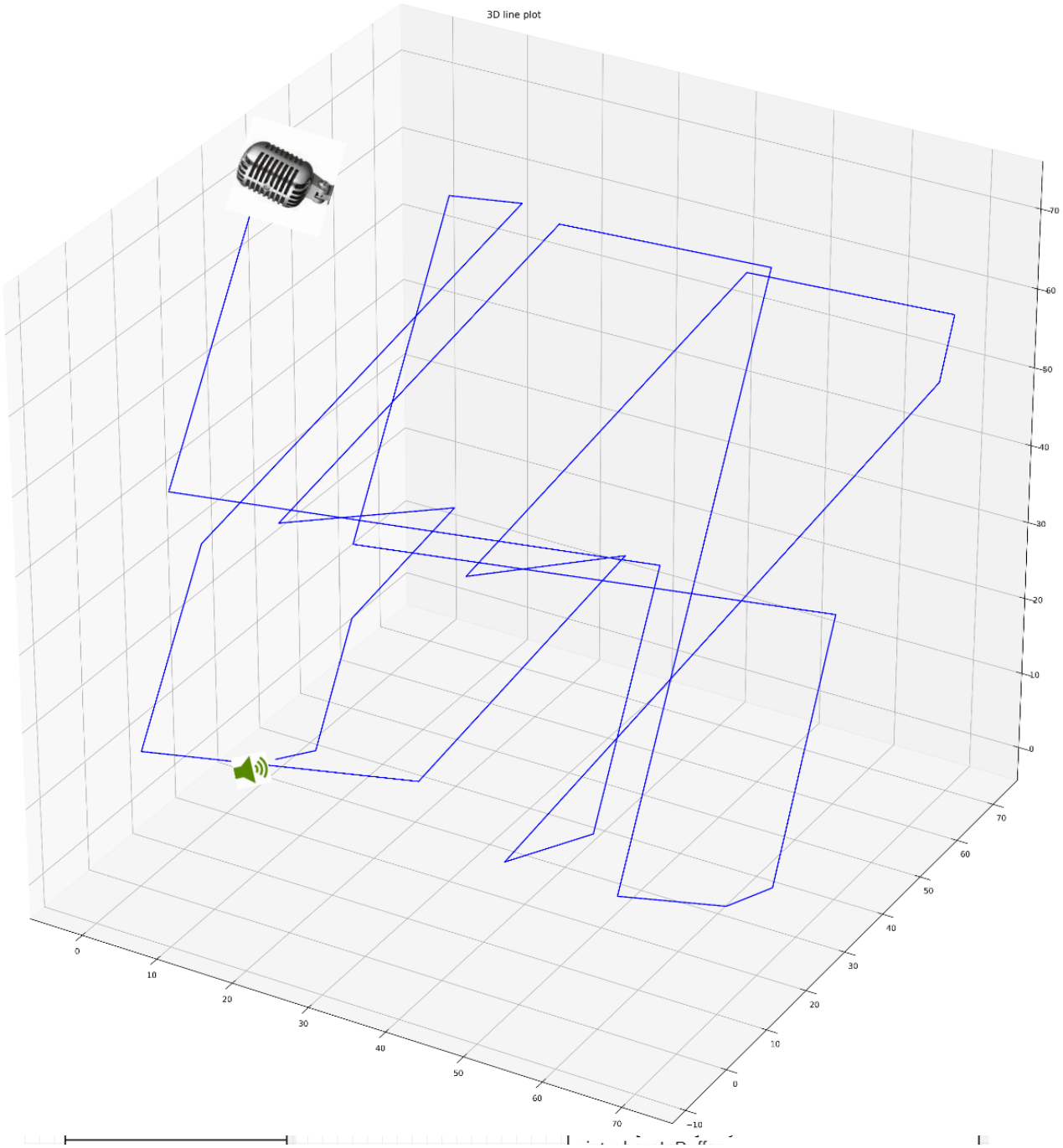


FIGURE 14 VISUALIZATION OF A SINGLE RAY BEING FIRED BY SOUND SOURCE AND DETECTED BY MICROPHONE

FIGURE 15 REVISED UML OF CORE RAY TRACER AS IMPLEMENTED

Key concepts to producing a ray traced reverberation.

1) Convert audio data is dissected into groups of samples.
2) Chunks of samples are cast off in all directions.

3) Cast rays by assigning ray objects to a vector of all active rays via initialization of ray directions. Once new rays are created, iterate through every ray in existence and perform a vector transformation on every ray by the vector stored in each ray object.
4) Rays are detected and reconstructed at microphones.
5) Microphone outputs are written to both left and right audio channels forming a stereo output.

Ray class.

The ray stores the current location of the ray using three Euclidean coordinates, X, Y, and Z, the rays direction X, Y and Z. We have opted to use a Euclidean coordinate system for storing both location and trajectories of the ray. Each ray contains a vector of doubles making up a sample. We also store the number of bounces the ray has made off each wall and the iterations travelled.

ThreeDimensionalRoom class

This is a very simple class containing three dimensions, width, height, and length which are used to generate a 3D Cartesian plane.

Microphone class

We cover the microphone class in more depth in the following section. A brief overview is that the microphone classes job is to detect Rays and reassemble the rays with the samples contained within to form output audio tracks. This class utilises an external audio buffer created using the external library AudioFile.h created by Adam Stark.

Key Methods to the core ray tracer:

raysDetectedByMic

This method passes rays hit and stores them in a vector of rays.

applyAmplitudeCalculationToSample

This method applies a suitable transformation to the detected ray objects to the samples contained within to produce the impulse response as aforementioned in figure 9.

sumOfEveryBitOfDataHittingMicAtThisIteration

This method constructs all ray objects detected to produce a suitable output. This is covered in more depth in the microphone design section.

soundSource class

The sound source class stores:

All ‘ray’ objects in existence in a vector,

All a vector of microphones within the cartesian plane.

generateDirectionsFibonacciSpiralSphere

This method initializes ray directions. We opted to utilize a Cartesian coordinate system to represent points within a Cartesian plane. To effectively emulate a sound source, we must generate ray directions such that rays directions are vectors to form equidistant points around a sphere ensuring a realistic emulation of a sound source. This method assigns each of these vectors(distance) to a C++ vectors;

vector in plane X, vector in plane Y and vector in plane Z as rayDirectionX, rayDirectionY and rayDirectionZ.

One method of generating equidistant points around a sphere is to utilize the Fibonacci Lattice otherwise known as the Golden Spiral. This Fibonacci spiral method, unlike other methods such as iterative, or randomised methods such as simulated annealing works for an arbitrary n where n is the number of points around the sphere. The following formula evenly distributes n points around a unit square [0,1]^2:

$$t_i = (x_i, y_i) = \left(\frac{i}{\phi}\%1, \frac{i}{n},\right) \quad \text{for } 0 \leq i < n$$

where

$$\phi = \frac{1+\sqrt{5}}{2} = \lim_{n\to\infty}\left(\frac{F_{n+1}}{F_n}\right)$$

FIGURE 16 [18]

The **(.)%1** operator denotes the fractional part of the argument. To produce the Fibonacci spiral, this point distribution is mapped to a unit disk via the equal-area transformation.

$$(x, y) \to (\theta, r): \quad (2\pi x, \sqrt{y})$$

FIGURE 17 [18]

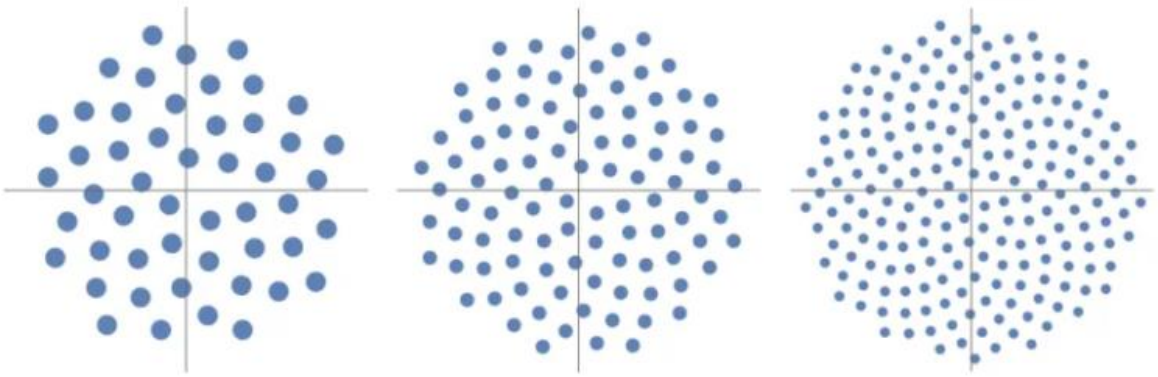


FIGURE 18 [18]

These points can be mapped from the unit square to sphere using cylindrical equal-area projection.

$$(x, y) \to (\theta, \phi): \quad (2\pi x, \arccos(1-2y),)$$

$$(\theta, \phi) \to (x, y, z): \quad (\cos\theta\sin\phi, \sin\theta\sin\phi, \cos\phi)$$

FIGURE 19 [18]

In this example we take “θ” as the longitude and “ Ø” as the angle from the north pole. It is important to note that this is NOT the convention we deployed when microphone modelling in which we use the mathematical convention.

We take the vector from the centre of the sphere to the equidistant points around the Fibonacci lattice spiral sphere as ray directions to which we cast rays.

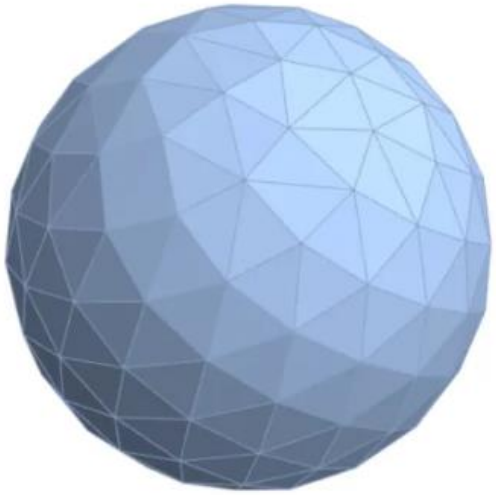


FIGURE 20 FIBONACCI LATTICE [18]

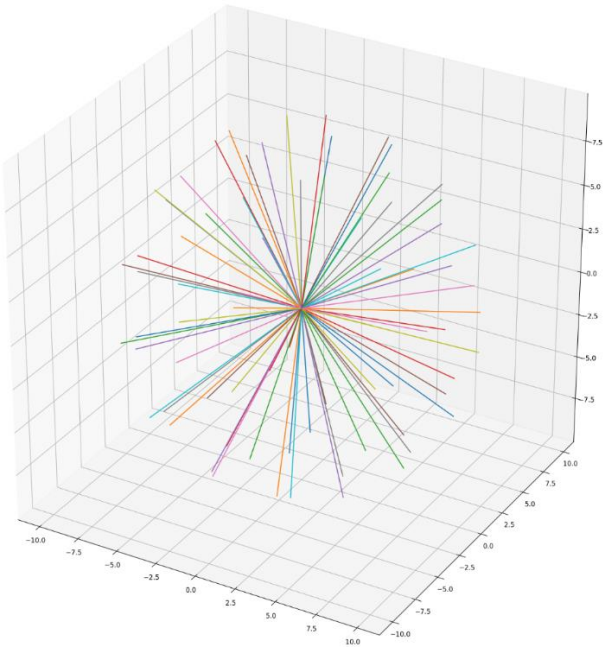


FIGURE 21 RAY GENERATION FOR 75 EQUIDISTANT RAYS AS VECTORS PLOTTED IN PYTHON.

We use this implementation as open source software as implemented in C++ by Bernardt Duvenhage[19]. We then perform a transform to the polar coordinates to create a vector of Cartesian vectors, to store into 3 vectors, DirectionX, DirectionY, DirectionZ. This is then used when casting rays in the castRay method to cast rays in all directions emulating that of a real sound source. We then plot this in python to confirm we are able to this is able to generate our vectors to cast rays equidistantly successfully. It is important to note that the distance each ray travels per iteration (or in every vector) is 10 and this is used to enable multiple sample rates to be used as input audio.

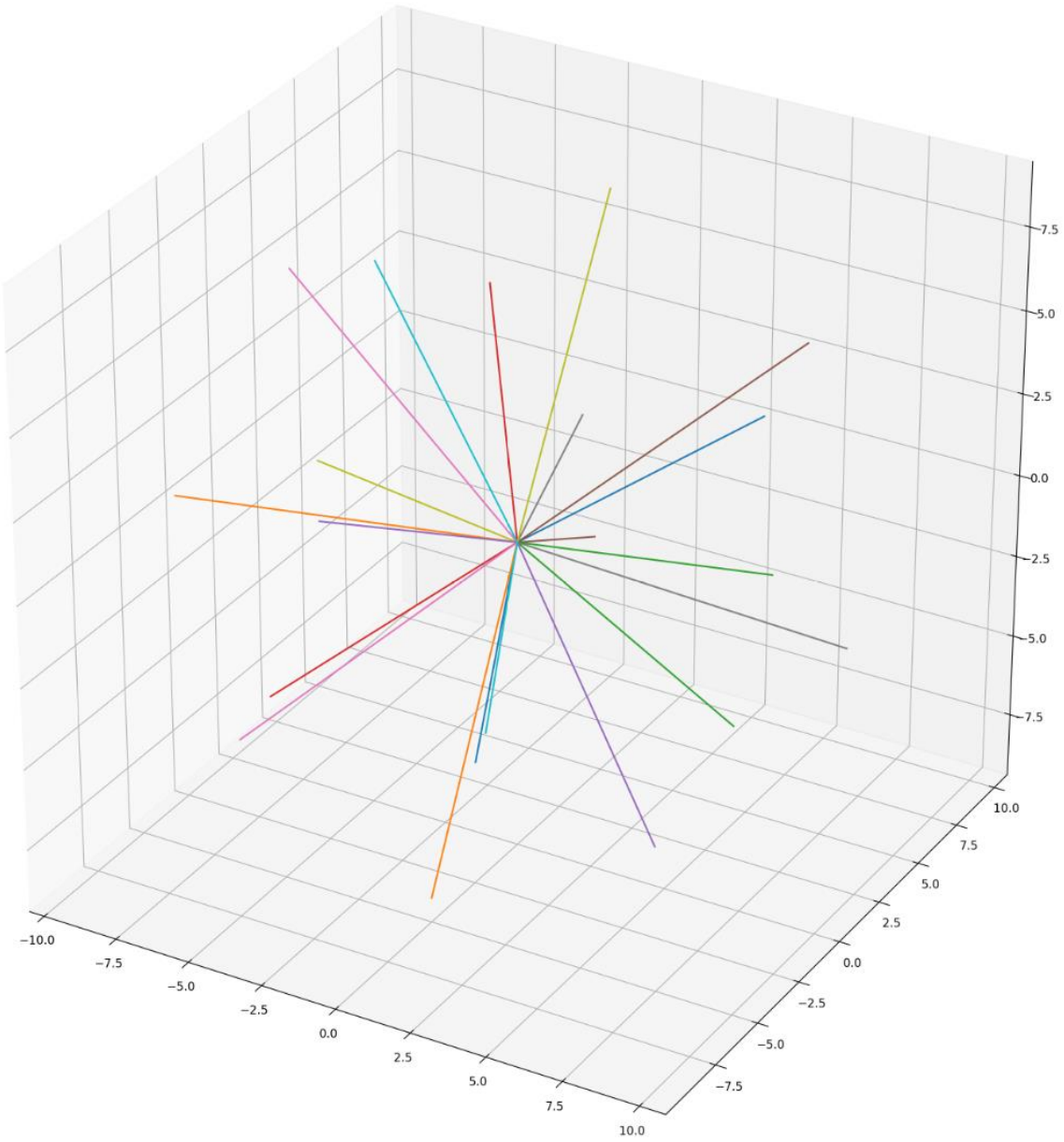


FIGURE 22 RAY GENERATION FOR 20 EQUIDISTANT RAYS TO DEMONSTRATE VARYING NUMBER OF RAYS CAST.

To ensure this is computationally efficient, we must control for the number of ray objects in existence at any given time.

These X number of rays are given a vector of Z samples.

performRayTrace

This is the method called to start the ray trace process. This step performs the following pre-processing steps.

- Generate a set of vectors such that every ray will be fired in directions emulating equidistant points around a sphere.

- Take the input audio data and dissect into groups of samples.

For example, 1.12365 seconds of audio at a sample rate of 44100kHz. 1.12365*44100 = total samples = 49,552.965.

As we are casting 1000 samples per ray, we generate 49,552.965/1000 = 49.552965 Rays in total to cast. As we cannot fire half a ray, we perform a modulo division and append the remainder as empty samples to all the input samples enabling the algorithm to cast 50 rays worth of groups of 1000 samples.

- We then add these 50 groups of 1000 samples to a vector containing 50 each of which contains a vector of 1000 samples.

- We then iterate through each of these groups calling the cast ray function for each group.

castRay

- This method creates ray objects. This generates new rays for the number of directions vectors generated in method “generateDirectionsFibonacciSpiralSphere”. The new ray objects are all assigned the group of Y samples and are added to a vector containing all active rays.

- the vector containing all active rays is pruned to remove all active rays beyond maxRaysInExistence. Then, all active rays that have been removed from the start of the ray of a vector of ray objects.

- the vector of all ray objects is then iterated through and every ray in existence is passed into the function traceOneRay.

- Rays that are detected by microphones are stored in the microphones for processing back into audio output.

traceOneRay

- as we have the bounds of the room in X, Y and Z forming the Cartesian plane we are able to evaluate if the Ray locations coordinates X, Y and Z are outside of the constraints of the room. If it is, we apply any relevant reflections in the form of changing the vector stored as ray direction such that the ray direction is an effective reflection.

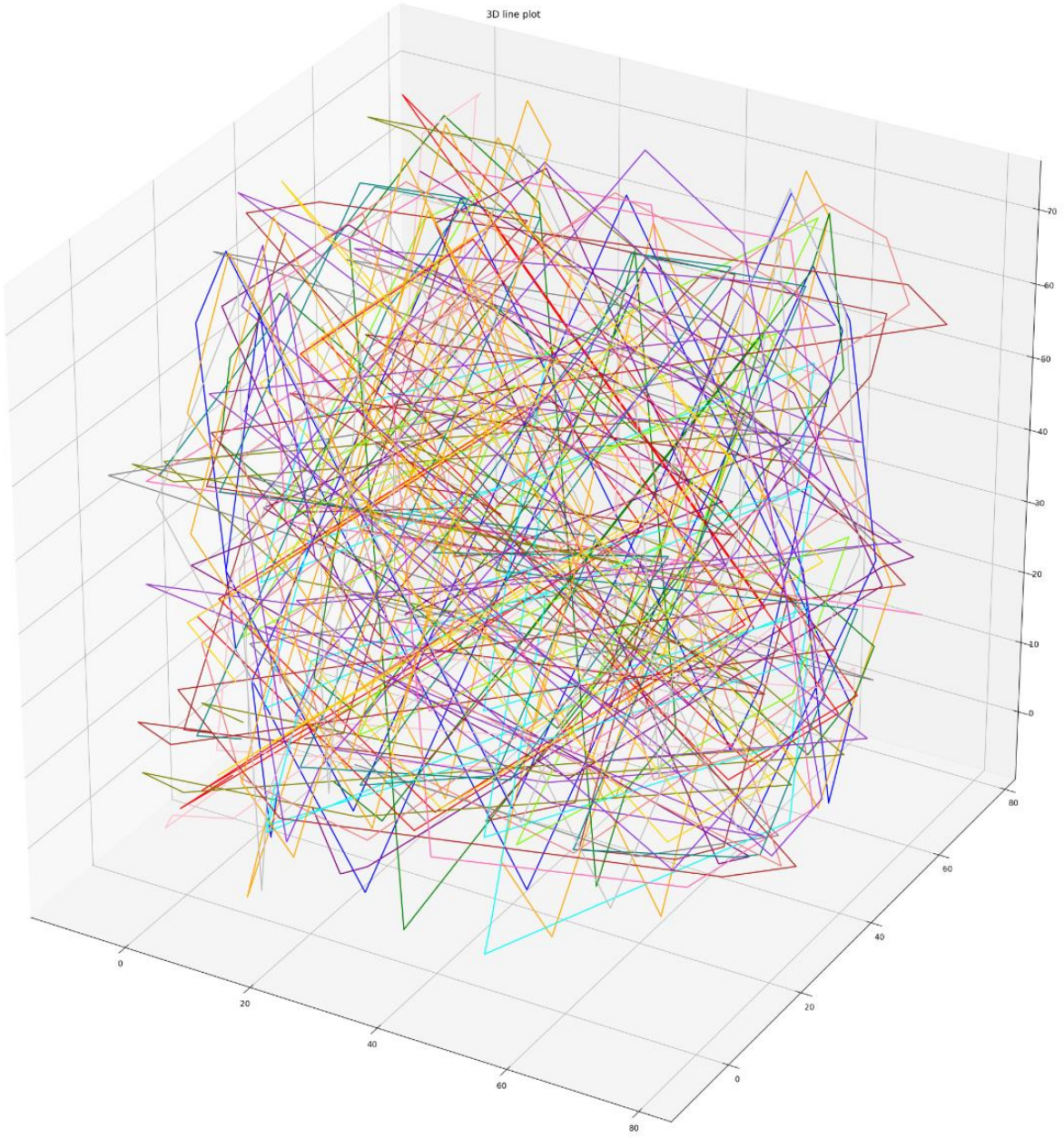


FIGURE 23 THIS VISUALIZATION SHOWS RAYS PROPAGATING THE 3D CARTESIAN PLANE.

Anytime this direction that change happens the ray number of bounces is incremented. This is useful for calculating the amplitude of samples at the microphones as energy is lost upon ray reflection.

As this method is called for every ray in existence during every iteration, we now process whether the Rays have been detected by the microphone. To do this we iterate through all the microphones and we check if the Ray has passed into the coordinates of any of the microphone locations. If the ray is detected, the ray is stored in a vector in the microphone. From here, the microphone class will process and applying the relevant transformation to the sample data of the rays detected to produce output audio sample data.

This process is repeated until all groups of samples have been traced.

Please refer to the code base for exact mechanisms of this process.

10.3 Microphone design:

One key aspect to microphones are their polar patterns. A polar pattern is the 3-dimensional space surrounding the capsule where it is most sensitive to sound. The three basic polar patterns are omnidirectional, figure-8 and cardioid.

We begin by implementing the omni-directional microphone.

A microphone with an omnidirectional polar pattern is equally sensitive to sound from the entire area highlighted in white in the below image.

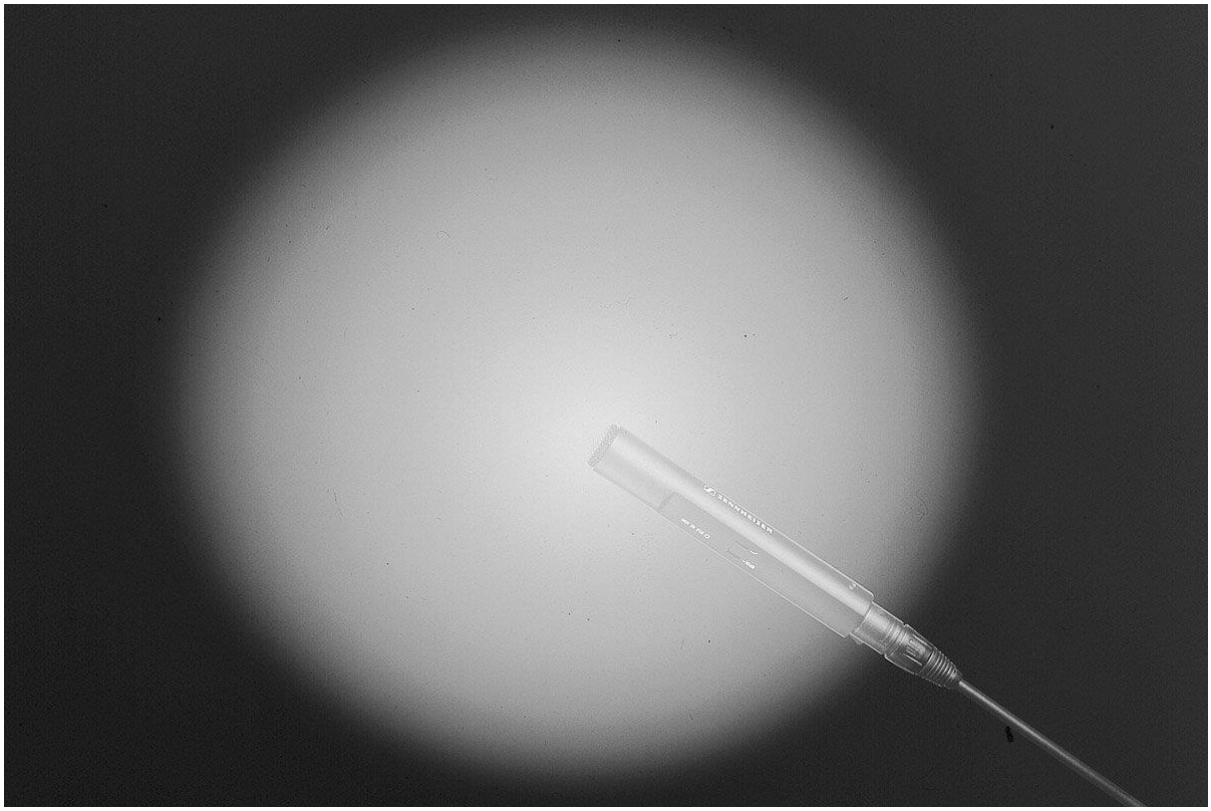


FIGURE 24 OMNIDIRECTIONAL MICROPHONE POLAR PATTERN [21]

Figure-8 Polar Pattern:

A microphone with a figure-8 polar pattern is equally sensitive to sound from the entire area highlighted in white in the below image.

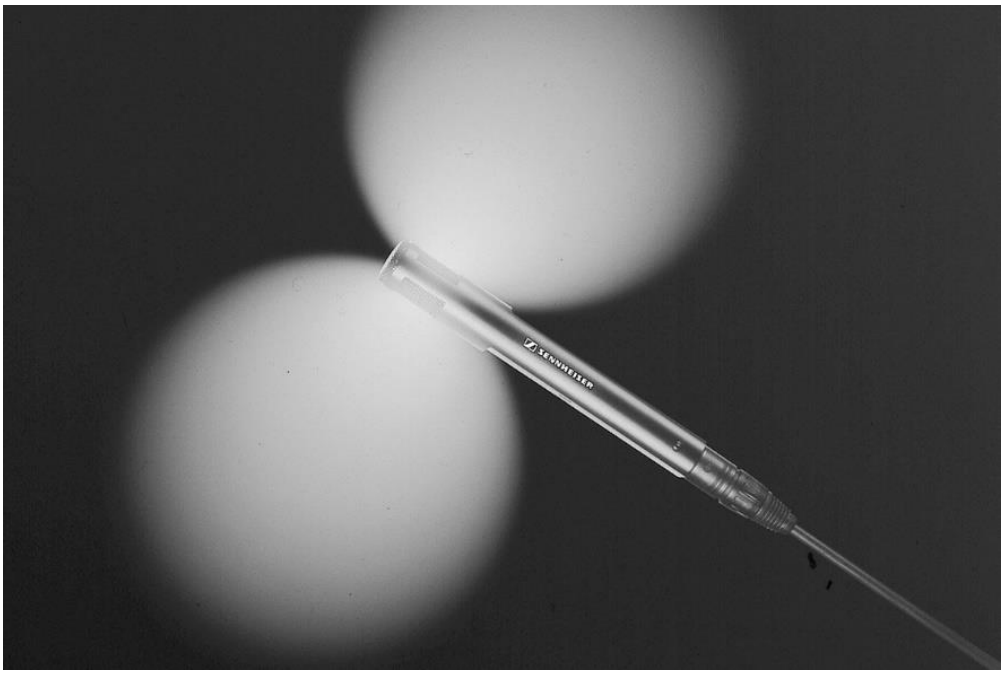


FIGURE 25 FIGURE-8 POLAR PICKUP PATTERN [21]

Cardioid Polar Pattern:

A microphone with a cardioid polar pattern is equally sensitive to sound from the entire area highlighted in white in the below image.

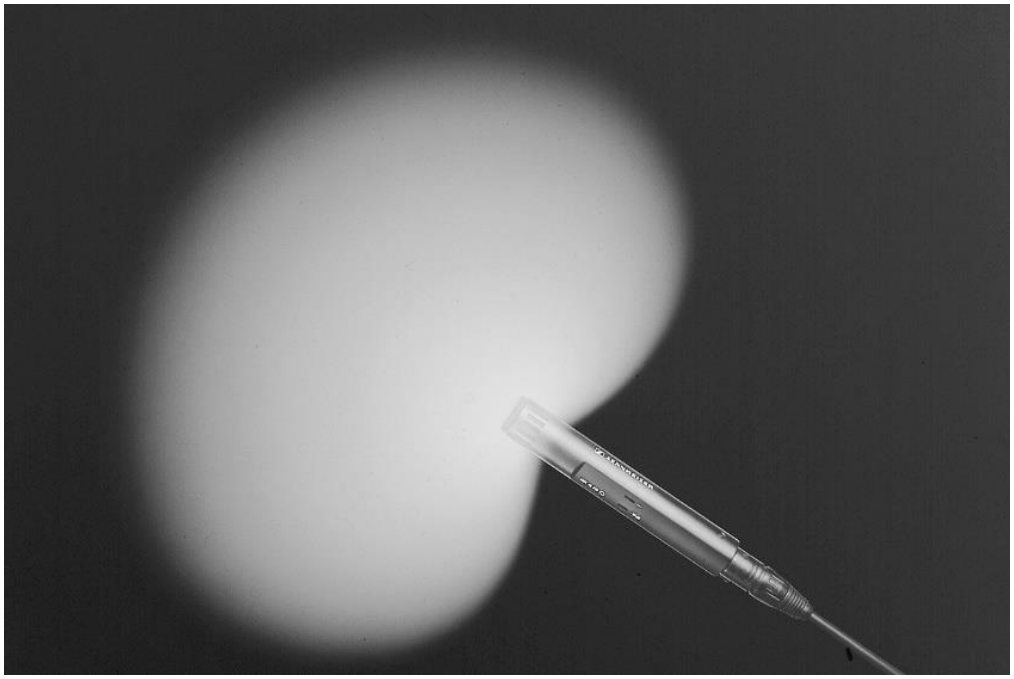


FIGURE 26 CARDIOID POLAR PICKUP PATTERN

We can represent these polar patterns with the following notation. Omnidirection, Figure-8 and Cardioid respectively.

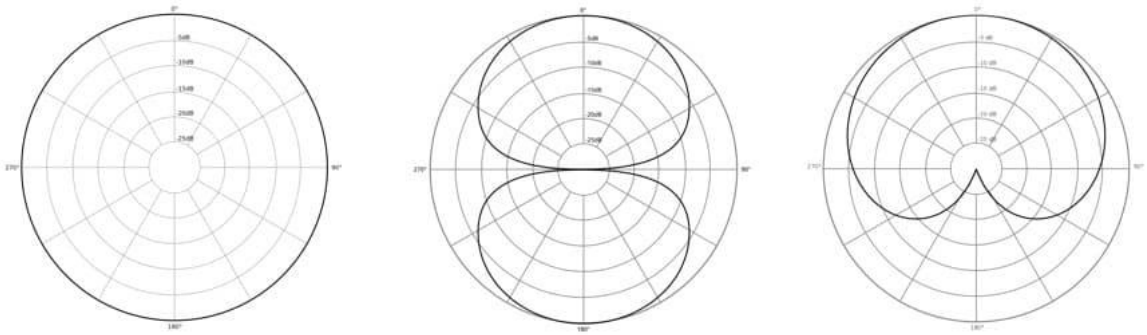


FIGURE 27 OMNIDIRECTION, FIGURE-8 AND CARDIOID POLAR PATTERNS RESPECTIVELY [22]

The Omni-Directional exists as a sphere within cartesian plane. These are commonly known as "Pressure" microphones which measure sound pressure at a given point in space.

Upon the current iteration of all rays, the distance between each ray and the centre of the spherical microphone is calculated. If this distance is less than the radius of the sphere, the ray is stored in the rays detected vector "RaysDetected". It is then deleted from allRays. The Figure-8 and Cardioid polar patterns have been built from the Omnidirectional polar pattern so first we detail key functions of how this works.

Once the algorithm has iterated through every ray in existence and filled the Rays Detected buffer, we then create an output sample for the current iteration of the algorithm. To do this the method, "sumOfEverBitOfDataHittingMicAtThisIteration" is called. For the number of samples each ray contains, a buffer for the current iteration "OutputSamplesToWriteToBuffer" is initialized containing 0s for the number of samples each ray contains.

The algorithm then iterates through all the rays in existence and calls the method "applyAmplitudeCalculationToSample" on every sample of each ray.

This method then inverts to sample data.

```
if ((double(numberOfBounces % 2)) == 1) {
   sample = -sample;
}
```

This accounts for the phase change seen upon every reflection of each ray.

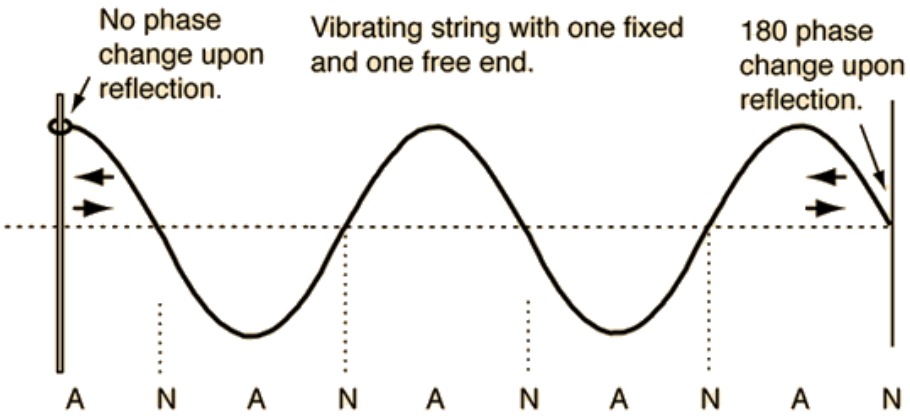


FIGURE 28 PHASE INVERSION UPON RAY REFLECTION [23]

To ensure this is carried out with high computational efficiency is to keep track of the number of bounces each ray has and then to only perform this calculation once as part of the reconstruction of the audio buffer within the microphone class.

On top of this we apply the following transformation to the sample data:

```
if (numberOfBounces >= 1) {
        sample = sample / pow(2, numberOfBounces);
 }
```

This accounts for the loss of energy found in pressure waves upon reflection. This is what creates the impulse response features of Direct sound, Early reflections, Late reflections, Reverb Tail.

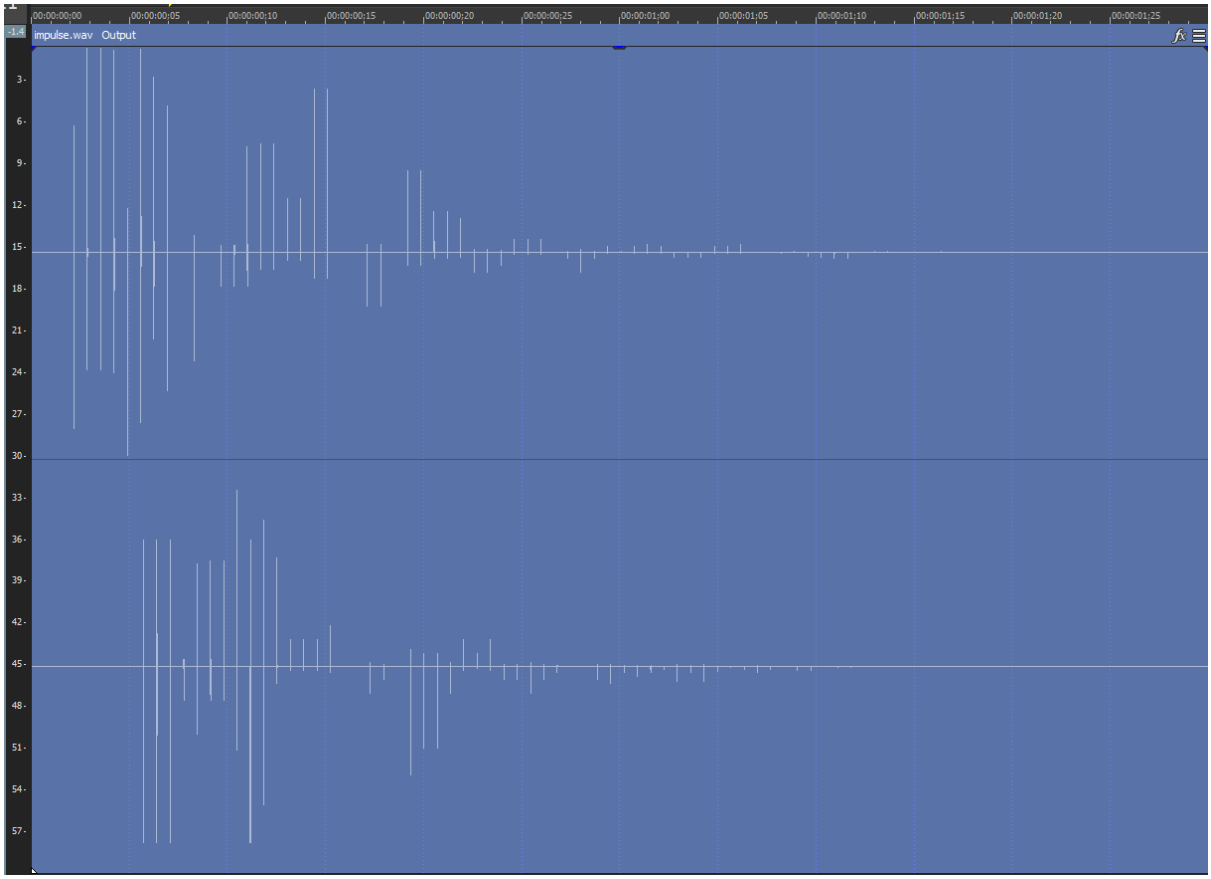


FIGURE 29 WAVEFORM VISUALIZATION OF IMPULSE RESPONSE OF THE ROOM WITH IMPLEMENTATION

As sound waves propagate through the medium of air, energy is in the waves are lost to the molecules in the medium of air as they are forced to vibrate. This is thermal energy. This process consequently means sound propagates limited distances relative to the energy of the sound waves. Lower frequencies travel further distances due to transferring less energy to the medium it the sound waves are propagating in. [24]

To prevent unnecessary increase of runtime complexity, this design has omitted modelling the intensity of sound wave decaying with distance via to an inverse square law relation between sound wave intensity and distance travelled.

Rather, we use the number of bounces to emulate the decay of sound with distance as reflections cause most of the loss of energy found in sound propagation. We also utilize a hard cap for total number of rays in existence as a pruning mechanism to ensure the ray tracing algorithm can produce output in a

reasonable amount of time without requiring excessive hardware resources relative to current CPU hardware standards as of 2021.

The scaling of efficiency with time will be in line with Moores' Law such that the algorithm may improve in efficiency as CPU speeds improve. Although this will lead to incremental improvements to render times as CPU performance improves. This is unlikely to yield any major breakthroughs with a short time horizon. A more realistic and shorter time horizon solution likely to yield much greater results would be to utilize the NVIDIA HPC C++ compiler by OpenACC[28]. This should be deployed to effective GPU hardware acceleration to cause drastically reduced processing times. However, this is beyond the scope of this project and therefore reserved as research and future work directions to ensure highest chances of achieving project extended requirements to the greatest possible extent.

However, due to lack of windows support for this compiler along with this project not requiring real time reverberation, we found optimizing for CPU runtime to be the best choice in ensuring as many of the core and extended objectives are met to the greatest possible extent within the given time frame without sacrificing other aspects of the project.

To enable us to model the propagation of sound realistically, it is important we account for the difference in propagation of sound between low and high frequency waves. The characteristics of high frequency waves having less energy and therefore travel less distance.[25] This means, we need to effectively filter out these frequencies as energy is lost in waves due to the number of bounces each ray has.

We model this effect by applying a low pass filter to rays of varying intensity relative to the number of bounces the ray has had since being fired from the sound source and being detected at the microphone. As most of the energy loss happens at the reflections of sound, this makes for a realistic creation of a reverberation effect without excessive increase in run-time complexity that processing the calculations to model the decay of sound with distance has. However, this does have limitations in terms of the properties of rooms that can be emulated.

We implemented the low-pass filter using the Butterworth Filter class[26]. The design of this utilizes a Finite Impulse Response Filter(FIR). FIR's are digital structures that enable implementation of almost

any sort of frequency response digitally. The FIR in the library we opted to use it implemented in the following structure:

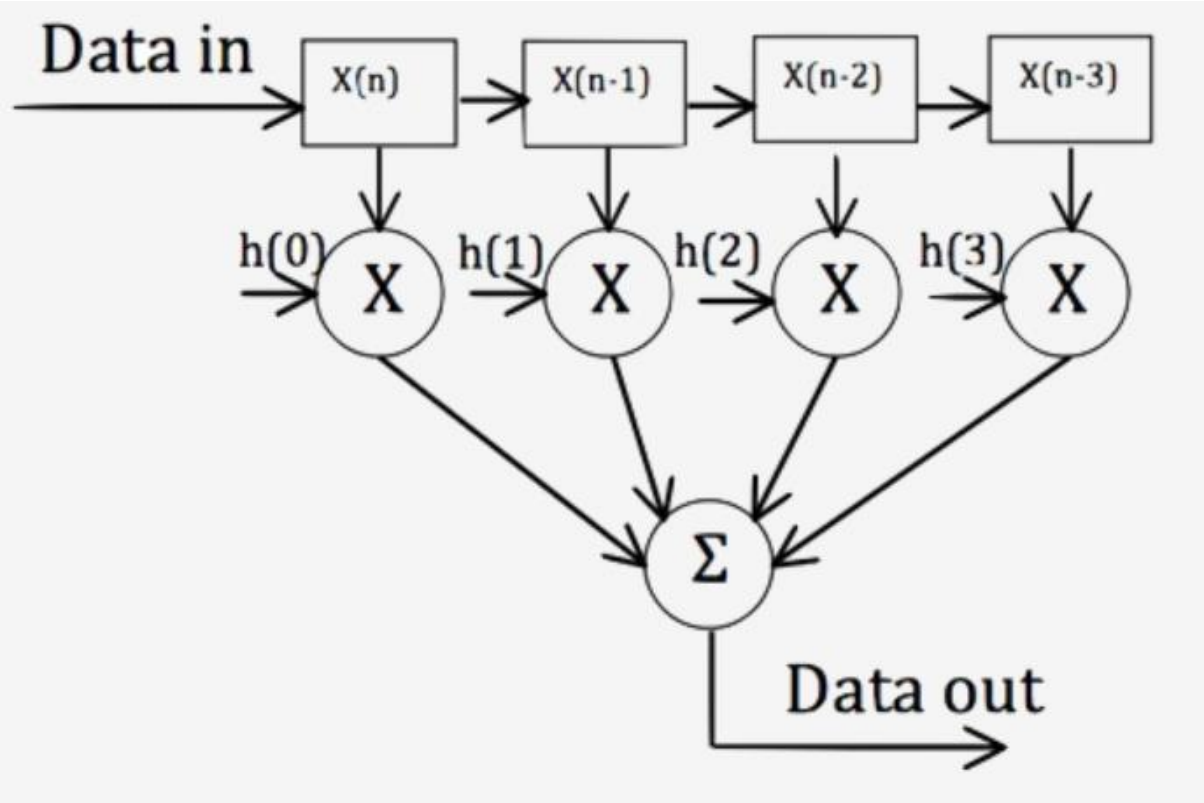


FIGURE 30 STRUCTURE OF FIR FILTER AS IMPLEMENTED [26]

We found this filter subjectively effective for making the reverberation algorithm sound less like a comb filter effect and sound much more like a reverberation effect and a definite improvement to the overall quality of reverberation sound. It is important to note the limitations of this approach are that it effectively emulates the frequency response of a room with thick walls. For a room with thin walls, the low pass could simply be replaced with a high pass filter.

After the appropriate transformations have been applied to all the samples, we then reconstruct the vector of output samples to fill the pre-initialised "OutputSamplesToWriteToBuffer". Then, the sum of all the samples stored in every ray in the following manner:

Ray 1, Ray 2, and Ray 3 all are detected by the microphone in the current iteration.

Ray 1 contains 5 samples [0, -1, 0, 1, 2]

Ray 2 contains 5 samples [0, 1, 0,-1, -2]

Ray 3 contains 5 samples [0, -1, 0, 1, 2]

The output 5 samples detected at the microphone after calling this method would be:

[0 + 0 + 0, -1 + 1 + -1, 0 + 0 + 0, 1 + -1 + 1, 2 + -2 + 2]

Which equates to and output:

[0, -1, 0, 1, 2]

"RaysDetected" and buffers used to create this process are then cleared.

This is then written to the output buffer for the microphone object and whilst concurrently emulating the phenomena of constructive and destructive phase interference upon microphone detection of rays.

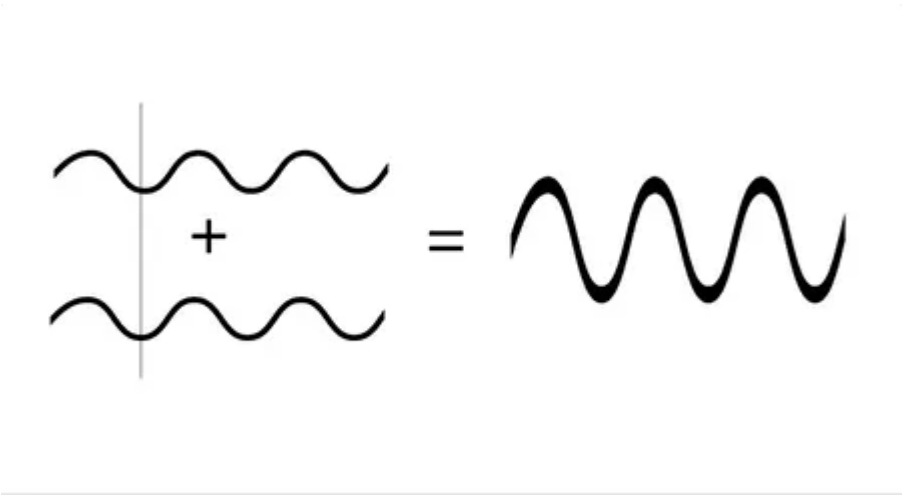


FIGURE 31 CONSTRUCTIVE INTERFERENCE PATTERN [27]

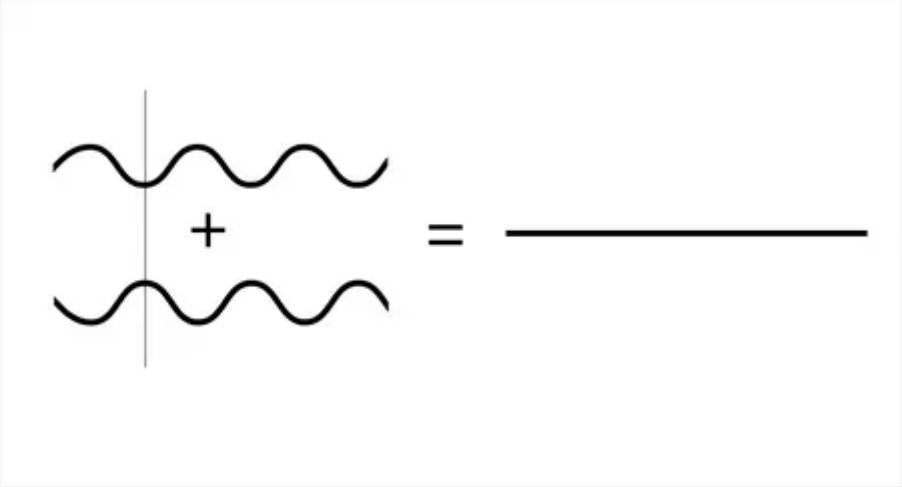


FIGURE 32 DESTRUCTIVE INTERFERENCE PATTERN [27]

This then results in a single output vector of sample data which we have visualized in figure 33.

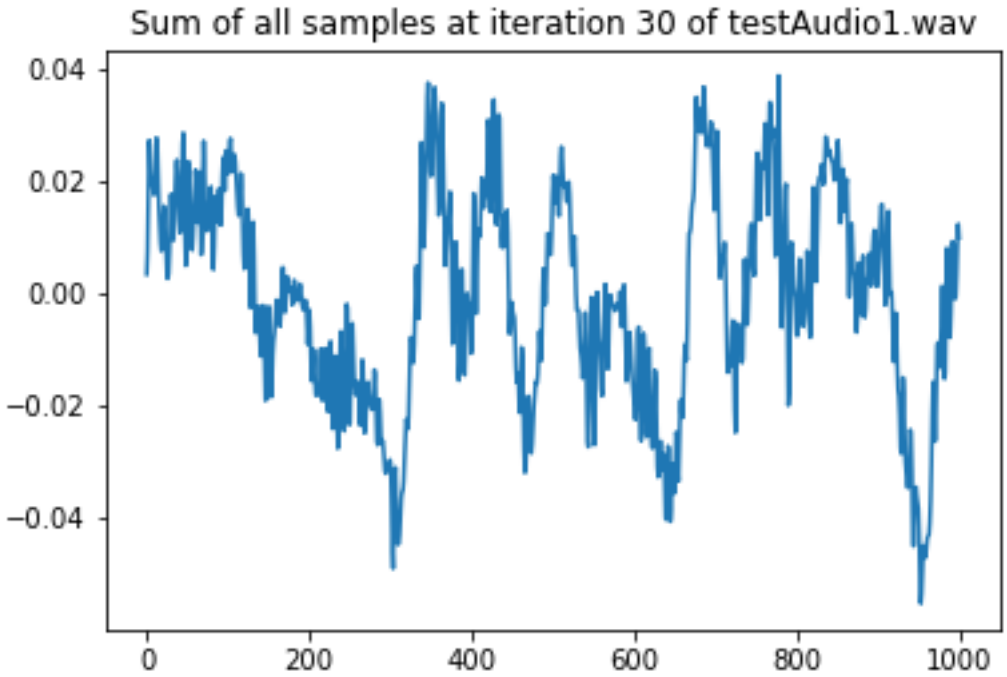


FIGURE 33 GRAPH SHOWING THE OUTPUT SAMPLE DATA FOR ALL RAYS DETECTED BY MICROPHONE AT ITERATION 30. X AXIS = SAMPLE NUMBER. Y AXIS = SAMPLE DATA.

This approach enables a level of accuracy of emulation of the physical laws governing reverberation. This approach is a compromise between max realism and max computational efficiency.

One other key aspect to ensuring laws of conservation of energy are met is the decay of amplitude/sound pressure with distance travelled. As we know the sample rate of the audio, the number of samples per ray, and the iterations travelled we calculate the sound intensity per unit distance travelled with the following formula:

We know the amplitude is inversely proportional to the distance. We have the sample rate of the input audio, and the iterations that the ray has travelled and the distance travelled per iteration of 10. The speed of sound in air of V = 343 m/s (rounded to nearest integer).

I = intensity, r = distance travelled.

$$I \sim 1/r^2$$

However, this has a computational cost and due to design choices capping the number of rays in existence, we were able to achieve a convincing reverberation, provided the user settings are tuned, without increasing the computational runtime complexity through implementation of this. Intuitively, one could presume this would cause a *step-like* or *boxy* reverberation effect due to sound intensity scaling only with the number of bounces. We observe via playing back the output audio this is not the case. We conclude that this *boxy* or *step-like* effect is negated through the emulation of constructive and destructive interference patterns at the microphone resulting in a much higher number of differing amplitudes detected to a point in which this effect is no longer audible. However, one should note that this effect is subjective and should be treated as such until further evidence has been presented.

Cardioid & Figure-8 Pick-up Pattern Implementation.

The cardioid and Figure-8 pickup patterns are implemented with many of the same principals as the omni-directional microphone. As described in detail, the active region from the Omnidirectional microphone is simply a sphere as it detects sound from all angles. As the omni-directional microphone pickup pattern detects rays from all angles, the direction is not needed to be modelled. For the Cardioid and Figure-8 pickup patterns, as they are directional microphones, require use of a spherical coordinate system.

We opted to use the spherical coordinates (r, θ, ϕ) as often used in mathematics to represent both the 3D orientation of our spherical microphone. Where r is the radial distance, azimuthal angle θ, and polar

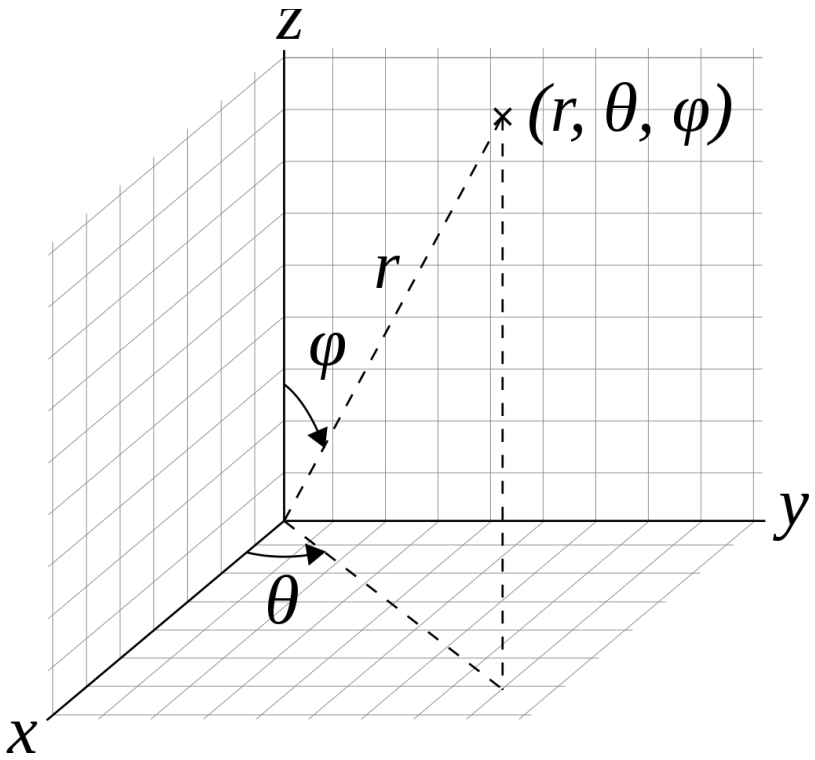


FIGURE 34 SPHERICAL COORDINATE SYSTEM WITH THE MATHEMATICAL CONVENTION [20]

angle ϕ. It is important to note that the meanings of θ and ϕ here are using the mathematical convention.

In the methods for both Cardioid and Figure-8, we must calculate the spherical coordinates of the rays when detected by the microphones to give both Polar and Azimuthal angles. As we have Polar and Azimuthal angles for the microphone and the incoming ray, we calculate the incoming angle relative to that of the microphone orientation in a 3D cartesian plane. As we have quantified this, we are able to detect rays with a selection of sound intensities from various angles.

The Cardioid pickup pattern is a combination of both omnidirectional and figure-8 microphones to give a strong front signal. The read side of a cardioid pickup pattern is a negative input from the figure 8 pattern. This is what gives the reduced input from rear of the microphone.

In our implementation, we simply allow rays to only be detected from the regions of the microphone that are active in the Cardioid pick-up pattern. In this implementation, the cardioid microphone will register rays within +75 and -75 degrees from 3D orientation of the microphone at full sound intensity.

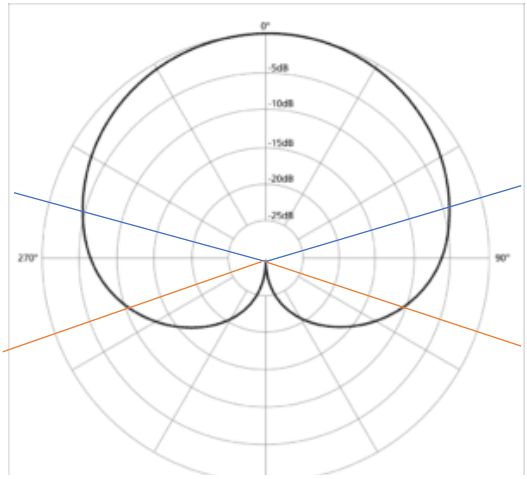


FIGURE 35 CARDIOID PICKUP PATTERN AS IMPLEMENTED.

If the ray is outside of this given window of +75 and -75 (blue line) and inside the given window of +120 and -120 (orange line) in both Polar and Azimuthal orientation of the microphone(blue line on diagram), the individual samples in these rays are multiplied by 1/3 to reduce the intensity of the sound input.

If the ray is outside of this given window of +120 and -120 (between orange line) in both Polar and Azimuthal orientation of the microphone the individual samples in these rays are inverted. to reduce the intensity of the sound input to form the inactive region of the microphone and boost the signal from the positive pickup region of the microphone

The Figure-8 Microphone Pick-up pattern is very similar to that of the cardioid microphone pickup pattern.
Figure-8 microphones are known as pressure gradient microphones. They measure the difference in pressure between sides of the active region.

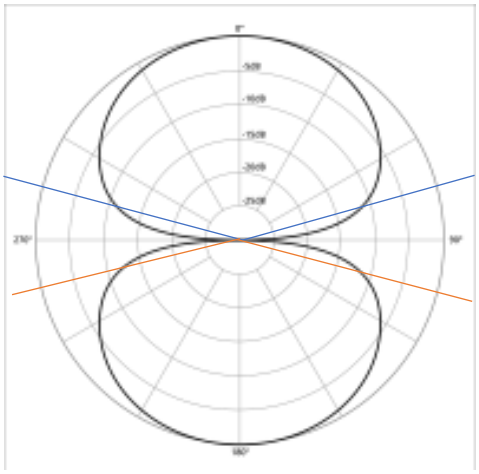


FIGURE 36 FIGURE-8 MICROPHONE PICKUP PATTERN AS IMPLEMENTED

The implementation of figure-8 polar pattern is much the same as cardioid in that it has an active region. However, the signal from the rear of the polar pick up pattern is not inverted. To implement this, we invert the microphone location to rotate it 180 degrees and use this new orientation to apply the following window of -75 and +75. On the polar pattern, this can be considered, blue to blue line, and orange to orange line. The area from blue to orange lines has a 1/5 attenuation so that the samples here in the inactive region are multiplied by 1/5. This is because generally it is near impossible to get a truly dead area within a microphone diaphragm.

Finally, the output buffers from both left and right microphones are written to both channel stereo left and right respectively.

11. Testing/Evaluation

To listen to the Reverberation effect, please refer to the code submission file. Test audio can be found in the folder “Audio Samples”.

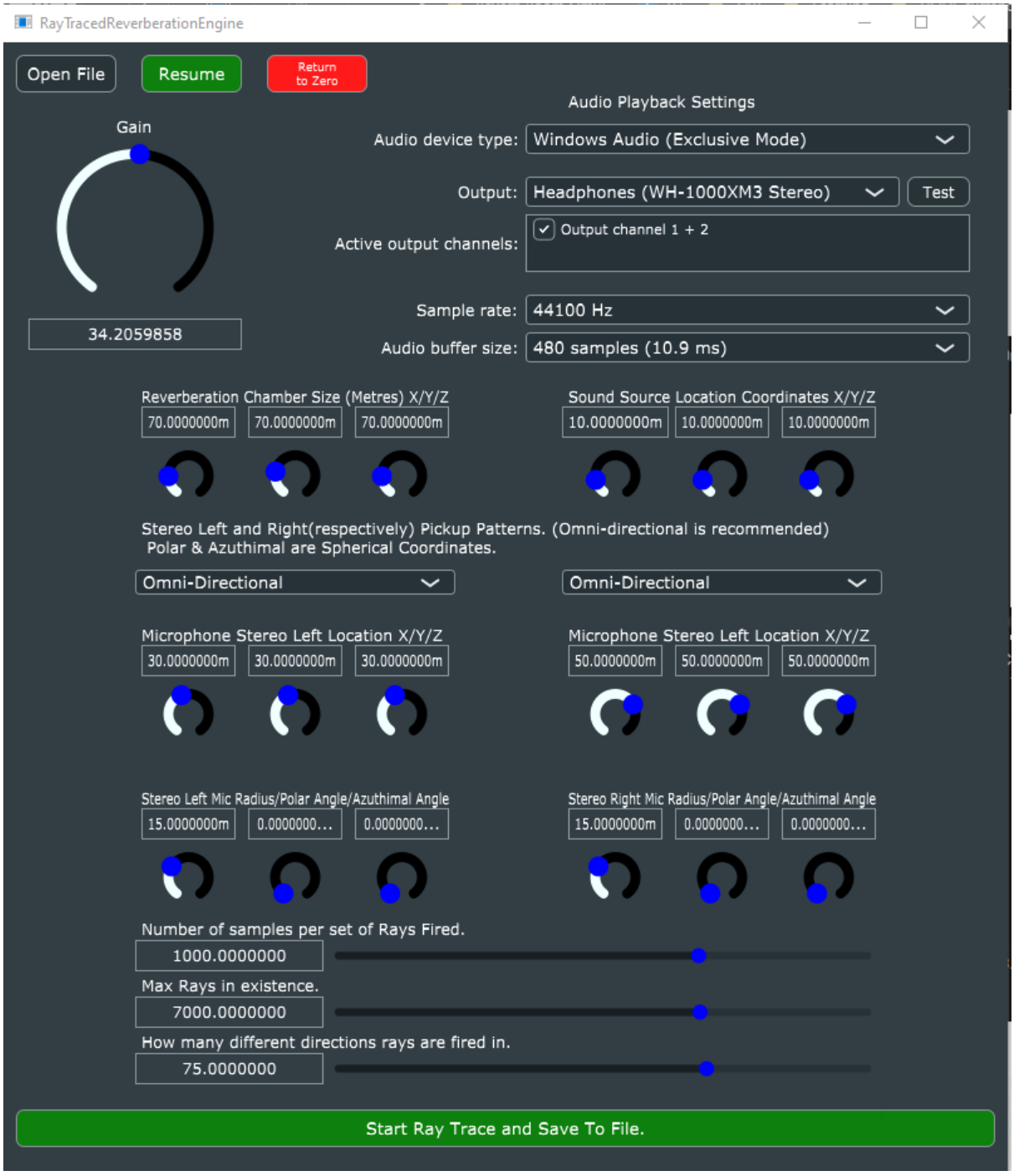


FIGURE 37 THE GUI

To help demonstrate extended requirements are met, we used an impulse of one sample and visualized the impulse response. We created this as a wave (.wav) file and opened it using our reverberation software with the following configuration. Please note that in the waveform for the upper channel is stereo left, and the lower channel is stereo right.

FIGURE 38 SETTINGS USED TO CREATE AN IMPULSE

FIGURE 39 THIS SHOWS THE INPUT IMPULSE BEING FED TO THE ALGORITHM WITH SETTINGS STATED IN FIGURE 8.

This resulted in the following impulse response output.

FIGURE 40 IMPULSE RESPONSE OF THE ROOM WITH NO GAIN ADJUSTMENT

We then used the gain adjustment tool in our software to adjust the gain of the output audio track from the impulse wave file.

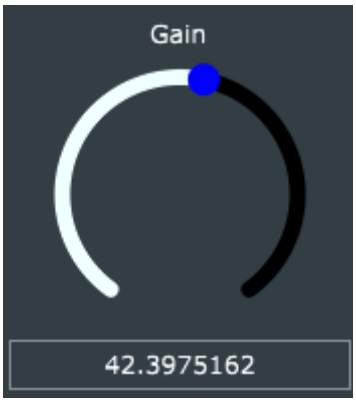


FIGURE 41 GAIN ADJUSTMENT

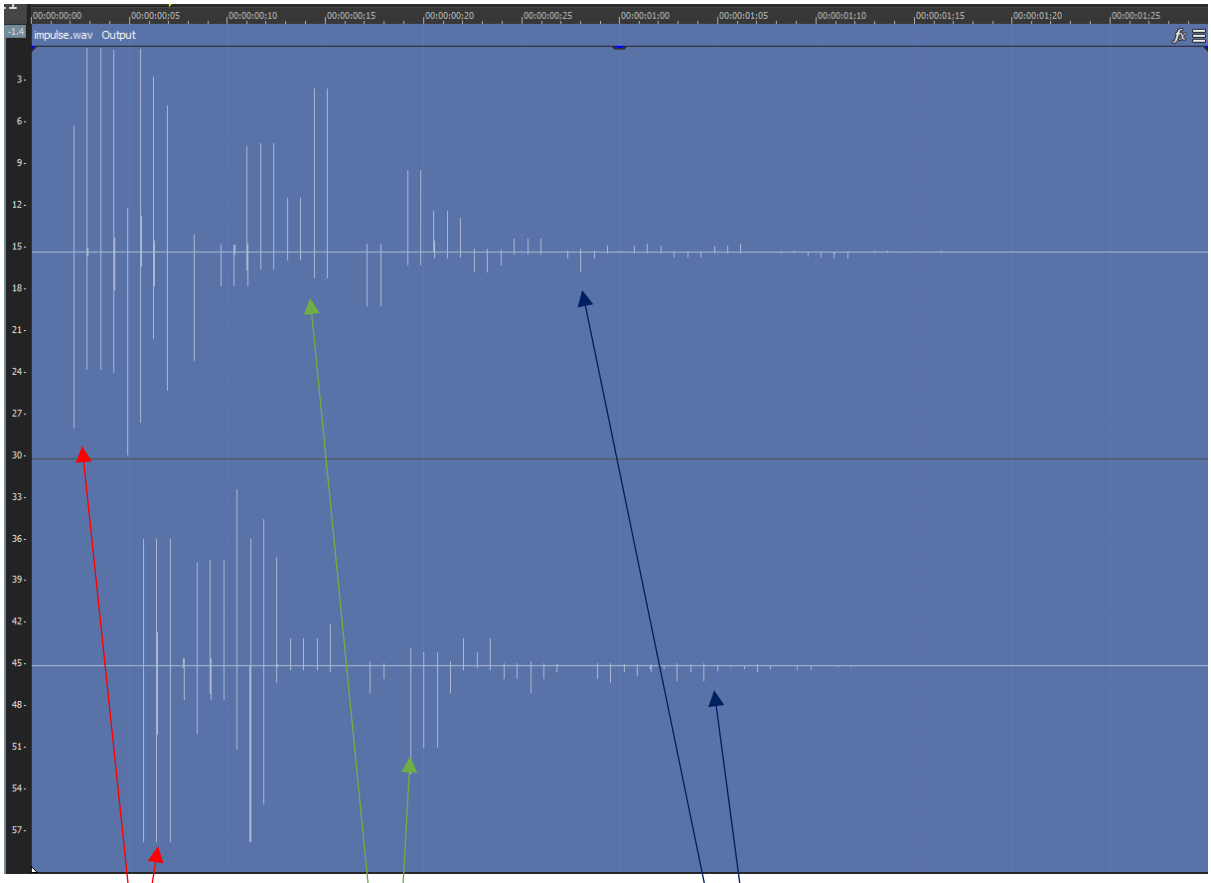


FIGURE 42 ACHIEVED OUTPUT IMPULSE RESPONSE AFTER GAIN ADJUSTMENT IN STEREO LEFT AND RIGHT CHANNELS.

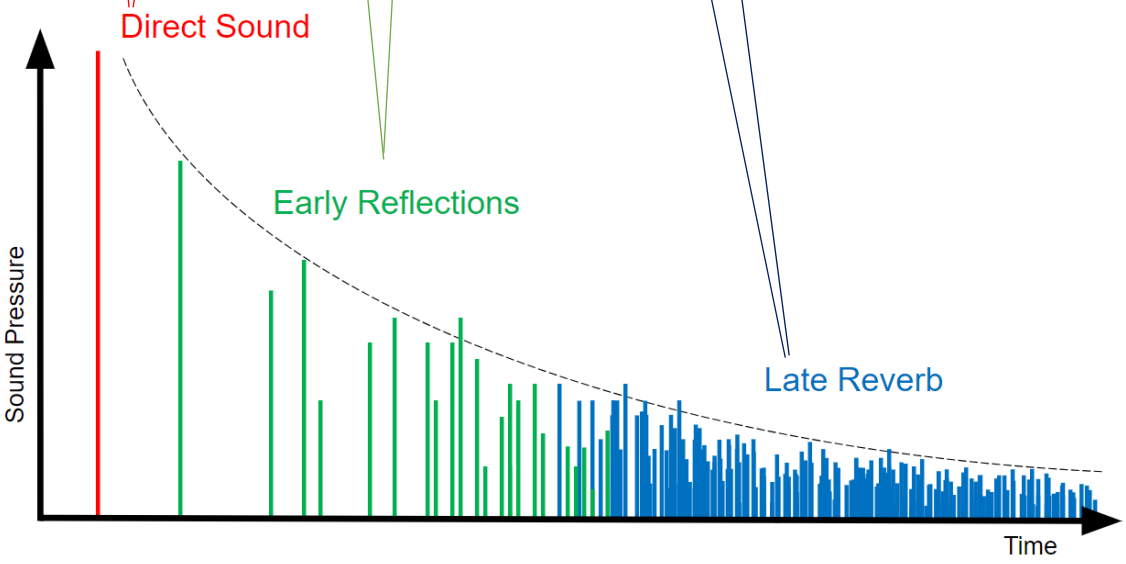


FIGURE 43 GOAL REVERBERATION VISUALIZATION OF IMPULSE RESPONSE [4]

The development process involved checking every component was working as anticipated at every stage of development to ensure steady progress.

Known bugs:

- Gain dial can be computationally heavy as it writes to the file immediately as the gain dial is adjusted rather than waiting for the user to let go of the dial. This can cause a hang up whilst the transformation is applied. This is mainly noticeable with longer files.

- Additional intermediatory audio file that is produced when applying the reverberation effect is not removed once processing is complete. This should be solved by using JUCEs framework to address individual samples rather than an external library.

- Changing the sample rate of playback can adjust the playback speed of the audio track.

- Polar and Azimuthal angles cause the GUI to be very difficult to visualize the microphone orientation within the 3D Cartesian plane.

- Redundant output file/s are generated when processing the output.

- Potential Comb Filtering Effect caused by certain room, sound source, microphone configurations.

- Large room sizes sometimes not produce output if no rays contact the microphone.

- Input sample rates other than 44100HZ can cause issues of pitch shift. Further testing is required.

11.1 System Requirements Evaluation

The System meets the core requirements in that the system allows audio upload and playback to apply ray tracing reverberation to an audio track in an offline manner and then be able to render the audio track with the reverberation effect applied using the CPU. The user can select a selection of sample rates. Instead of the adjusting the output audio bit depth, the user is provided with adjustment of the playback buffer size.

Early versions of the software are as specified in the core requirements, two-dimensional chambers later developed into a three-dimensional reverberation chamber. Please refer to early versions as found in the GitHub Repository to find this functionality as these core objectives were a key development phase to enabling full development and implementation of the extended requirements.

The user is provided a GUI to enable them full control over the algorithm's parameters. Please refer to the screenshot of the GUI.

For this project to be considered successful, the following core system requirements must be met.

1) System allows audio upload and playback to apply ray tracing reverberation to an audio track in an offline manner and then be able to render the audio track with the reverberation effect applied using the CPU.
    a. The user can select from a selection of sample rates and bit depths for the output audio.

    - Users can select from a choice of two sample rates for playback in the application. The audio track will apply reverb to wav files with the same sample rate/bit-depth the input audio is and output.

    b. The audio effect will monoaural and generated from a single listening point.

    - See early versions in GitHub Repository to hear reverberation with a single listening point.

    c. Reverberation room shall be 2 dimensional.

    - See early versions in the GitHub Repository to hear a project version with a 2-dimensional reverberation effect applied.

    d. User can upload an audio track for processing within the software.

    - Users can upload audio tracks for processing within the software

e. Playback the uploaded track.

- Users can play back the audio track.

f. The user should be able to render the audio track with the reverb effect applied.

- Users can render the audio track and listen back to the reverberation effect within the app and saved the reverb effect to file.

2) System should form a digital audio workstation plugin.

- We developed a standalone GUI Audio Application built to run on the Windows operating system rather than the digital audio workstation plugin. The output can be imported to all DAWs supporting .wav format. Please refer to conclusions and future work section for further information.

3) Create several pre-configured microphone placement and room type configurations.

- The room is in a pre-configured microphone and provided controls to adjust the pre-configured microphone placement to allow a huge amount of customizability.

4) Implement a GUI and code to allow user to adjust:
   a. Sample rate of reverberation audio track forming the number of rays traced per second.
   b. The bit depth of each sample.
   c. Gain.
   d. Volume.

   - We have used the same dial to form the gain, and the volume due to the design process of the audio files.

   e. pre-configured microphone placement and room type.

   - The application opens with a pre-configured microphone placement and room shape as a 3D cartesian plane.

11.2 Extended Requirements Evaluation

After meeting the core requirements, we implemented the extended requirements.

1) The system allows user to manually add and relocate existing microphones and sound sources within the 3D space.
   - The user has full control to relocate 2 microphones (stereo left and right respectively) within the 3D cartesian plane. The user can manually adjust the microphone polar pattern and geometry along with the sound source location bounded by the constraints of the user defined 3D cartesian reverberation chamber.
2) Create a class for each microphone type:
   a. Cardioid
   b. Omnidirectional
   c. Figure-8

   - These are implemented and functional.

3) Enable Directional Reverberation (Stereo) based on where the microphone/microphones are placed within the room relative to the sound source.
   - *Stereophonic sound is produced relative to sound source and microphone location and orientation within 3D Cartesian plane.*

4) The reverb algorithm should be improved to emulate the following key features to creating reverberation:
   a. Direct sound
   b. Early reflections.
   c. Late reflections.
   d. Gradual build-up.
   e. Reverb Tail.
   f. Frequency response.
   g. Strict adherence to the law of conservation of energy.
   h. "Physical feeling of pressure with extremely small rooms." [5]
   i. "The inherent subsonic tremble of huge rooms." [5]

   - *We have demonstrated Direct sound, early reflections, late reflections, reverb tail.*

6d)Gradual build-up. – With design of our algorithm, the number of rays in existence will increase until it hits the upper limit as defined by the users through use of the max rays in existence control. This means that the number of rays in existence will gradually build up until it hits this fixed cap. It will then remove the rays from the front of the vector to ensure this number is not exceeded. It is worth noting that rays can contain samples data of all zeros. As more rays are cast, more rays containing a specific set of samples gradually build up taking up an increasing number of the max rays in existence.

Frequency response.

- We were able to emulate the frequency response of a room with thick walls through implementation of a low pass filter.

Strict adherence to the law of conservation of energy.

"Physical feeling of pressure with extremely small rooms."

We have used a user defined fixed cap of total rays and adjustments of the cartesian plane that forms the size of the room. This means that as the user makes the room smaller using the GUI, they increase the densely of rays per unit area and vice versa. This is how we tackled this retrospectively subjective/ambiguous requirement. We found this to be quantitative approach.

"The inherent subsonic tremble of huge rooms." [5]

We managed to emulate this through use of a low pass filter ensuring more low frequencies occur. However, there is a carry over where this effect can be observed in smaller rooms.

5) Research optimized runtime utilizing CUDA acceleration technology [14][15] to speed up render times for rendering out the audio track into a selection of audio format with options of both sample and bitrates.
   - *Please refer to the Conclusions and Future work for the summary of GPU acceleration research.*

12. Conclusions

We were able to successfully design and implement all core and extended objectives with the exception of DAW integration due to an early misinterpretation of the JUCE tutorial documentation.

After extensive further investigation of the available JUCE documentation, we discovered that the base Audio Player project template was not an Audio Plugin format. It was more suited to a standalone audio application. As a result, the project was built as a GUI standalone Audio Application. Whilst this limitation only became apparent relatively late in the development, it has not prevented us from delivering fully on the core concept of creating reverberation using ray tracing. Users can still apply a reverberation effect to a wav file and then import the wave audio file into the DAW for use.

We considered runtime optimization exploring a range of techniques. We decided to experiment with optimized runtime using CUDA acceleration and, after several weeks of exploring RTX engines such as VR Works Audio and reading several CUDA acceleration documentation manuals, we concluded that this would be out of scope for the project due to the time required to implement the technology likely sacrificing some of the other objectives. We did however, as a result, explore ways of reducing run time complexity through a range of techniques including grouping samples, and removing unnecessary operations where possible. To compensate for not utilizing CUDA acceleration techniques, we offer the user full control of many of the algorithms hyperparameters such as, samples per ray, max rays in existence and more. We felt this approach was more consistent with the original project objectives with less risk. This approach did allow us to discover an alternative hardware acceleration/parallelisation compiler in OpenACC[28]. This is discussed in more detail in the further work section.

13. Further Work

The microphone class could be reworked to be more efficient. Currently the algorithm applies a transformation based on the microphone active region to every ray as it is detected by the ray. This process is inefficient as it applies the transformation to individual rays one at a time upon detection by the microphone. This process could be streamlined by storing multiple buffers for each active region of each microphone to which the rays detected can simply be added together. Once all rays have been detected we now have buffers for each active region of the microphone. From here we can apply the relevant transformations to each buffer. For example, multiple all samples in the inactive region by 1/10. Finally, iterate through to find the sum of all samples at each place in the output buffers to form a final output for the microphone. This approach would drastically reduce computational load. Further work should be carried out first writing pseudocode and then implementing this code in C++.

Furthermore, the GUI interface should be improved to:

- Improve controls for mic orientation within 3D Cartesian plane to have a drag and drop/3d rotation functionality.

- Address gain dial efficiency particularly when working with longer audio files.

The software should have:

- Added functionality to ensure audio clipping does not occur as result of too strong of too many rays at one iteration causing too strong of a signal. This can be done by measuring the input audio and normalizing the output of each microphone. One potential solution for this could be to take the average sample value for all samples above the value of 0 of the input and multiply this value to every individual positive sample in the output. The same should be repeated for all samples below the value of 0. Some experimentation may be required with this approach to find the best and efficient solution. However, it is worth noting that this, in some cases may be caused by a potential comb filtering or excessive constructive interference effect caused by certain room, sound source, microphone configurations which requires a full investigation into the exact cause to resolve this issue. One suspected cause is a Comb Filtering Effect caused by wave interference patterns of various sample data. This may present itself only with certain room, sound source, microphone configurations so may require a thorough investigation.

- GPU acceleration. This can be achieved upon release and use of the OpenACC [28] compiler for windows as it currently is only supported on Linux. An alternative would be to set up the JUCE/VisualStudio 2019 environment on linux to utilize the openACC compiler and follow their documentation to parallelize and hardware accelerate the algorithm. This can be as simple as adding the line “#pragma acc parallel loop” above various for loops within the algorithm for a basic acceleration or this could be complex as developing a real time ray tracing reverberation solution.

- A selection different shaped rooms through 3D rendering. This likely needs a full rework of the ray reflection aspect of the project to enable accurate ray reflection for complex geometry. Potential formats to create the 3D environment geometry could be VRML or X3D created in 3DS max for example.

- Re-wrap the project and UI into an audio plugin structure following the JUCE documentation to correct the misinterpretation of the documentation enabling cross-platform support and DAW plugin capabilities.

14. References

[1] NVIDIA Developer. 2020. *Vrworks - Audio*. [online] Available at: <https://developer.nvidia.com/vrworks/vrworks-audio> [Accessed 12 October 2020].

[2] Bcs.org. n.d. *BCS Code Of Conduct*. [online] Available at: <https://www.bcs.org/membership/become-a-member/bcs-code-of-conduct/> [Accessed 9 November 2020].

[3] Xiang, N., 2020. Generalization of Sabine's reverberation theory. *The Journal of the Acoustical Society of America*, 148(3), pp.R5-R6.

[4] Inc., A., 2020. *Audiokinetic Blog*. [online] Blog.audiokinetic.com. Available at: <https://blog.audiokinetic.com/image-source-approach-to-dynamic-early-reflections/> [Accessed 9 November 2020].

[5] Quantec.com. 2020. *QUANTEC: Room Simulation*. [online] Available at: <http://www.quantec.com/index.php?id=room_simulation> [Accessed 12 October 2020].

[6] 2020. *Mixing Gerald Heyward | Ross Rothero-Bourge*. [image] Available at: <https://www.youtube.com/watch?v=PiXjBvPNCPw&ab_channel=VicFirth> [Accessed 12 October 2020].

[7] Hidetoshi, N., Yoshifumi,, C., Tsuyoshi, U. and Masanao, E., 2003. Frequency domain binaural model based on interaural phase and level differences. *J-STAGE*, [online] Available at:

[8] Juce.com. 2020. *JUCE | JUCE*. [online] Available at: <https://juce.com/> [Accessed 11 November 2020].

[9] GitHub. 2020. *Iplug2/Iplug2*. [online] Available at: <https://github.com/iPlug2/iPlug2> [Accessed 11 November 2020].

[10] Jvstwrapper.sourceforge.net. 2020. *Jvstwrapper - Java-Based Audio Plug-Ins*. [online] Available at: <http://jvstwrapper.sourceforge.net/> [Accessed 11 November 2020].

[11] Doyle, K., 2020. *Education_Reflections*. [online] Kevindoylemusic.com. Available at: <http://www.kevindoylemusic.com/education_reflections/> [Accessed 9 November 2020].

[12] Arons, B., 2020. A Review of The Cocktail Party Effect. *Journal of the American Voice I/O Society, 1992*, p.1.

[13] Hansen, P., 2020. FUNDAMENTALS OF ACOUSTICS. *https://www.who.int/*, [online] pp.5-6. Available at: <https://www.who.int/occupational_health/publications/noise1.pdf> [Accessed 6 November 2020].

[14] Harris, M., 2020. *An Even Easier Introduction To CUDA | NVIDIA Developer Blog*. [online] NVIDIA Developer Blog. Available at: <https://developer.nvidia.com/blog/even-easier-introduction-cuda/> [Accessed 11 November 2020].

[15] Gupta, P. and Perelygin, K., 2020. *CUDA Refresher: The CUDA Programming Model | NVIDIA Developer Blog*. [online] NVIDIA Developer Blog. Available at: <https://developer.nvidia.com/blog/cuda-refresher-cuda-programming-model/> [Accessed 11 November 2020].

[16] AudioThing. 2021. Phase Motion 2 - Advanced Stereo Phaser Plugin (VST, AU, AAX) - AudioThing. [online] Available at: <https://www.audiothing.net/effects/phase-motion/> [Accessed 9 May 2021].

[17] Docs.juce.com. 2021. JUCE documentation. [online] Available at: <HTTPS://DOCS.JUCE.COM/> [Accessed 9 May 2021].

[18] 2021. How to evenly distribute points on a sphere more effectively than the canonical Fibonacci Lattice. [online] Available at: <HTTP://EXTREMELEARNING.COM.AU/HOW-TO-EVENLY-DISTRIBUTE-POINTS-ON-A-SPHERE-MORE-EFFECTIVELY-THAN-THE-CANONICAL-FIBONACCI-LATTICE/#:~:TEXT=THE%20FIBONACCI%20LATTICE%20(AKA%20FIBONACCI,PRESERVING%2C%20NOT%20DISTANCE%2DPRESERVING> [Accessed 9 May 2021].

[19] GitHub. 2021. bduvenhage/Bits-O-Cpp. [online] Available at: <https://github.com/bduvenhage/Bits-O-Cpp> [Accessed 9 May 2021].

[20] En.wikipedia.org. 2021. Spherical coordinate system - Wikipedia. [online] Available at: <https://en.wikipedia.org/wiki/Spherical_coordinate_system#/media/File:3D_Spherical_2.svg> [Accessed 9 May 2021].

[21] Neumann.com. 2021. Why Do Microphones Have Different Pickup Patterns?. [online] Available at: <https://www.neumann.com/homestudio/en/cardioid-omni-figure-8-why-do-microphones-have-different-pickup-patterns> [Accessed 9 May 2021].

[22] My New Microphone. 2021. The Complete Guide To Microphone Polar Patterns | My New Microphone. [online] Available at: <HTTPS://MYNEWMICROPHONE.COM/THE-COMPLETE-GUIDE-TO-MICROPHONE-POLAR-PATTERNS/> [Accessed 9 May 2021].

[23] Hyperphysics.phy-astr.gsu.edu. 2021. Reflection of Waves. [online] Available at: <http://hyperphysics.phy-astr.gsu.edu/hbase/Sound/reflec.html> [Accessed 9 May 2021].

[24] Openstax.org. 2021. 16.4 Energy and Power of a Wave - University Physics Volume 1 | OpenStax. [online] Available at: <https://openstax.org/books/university-physics-volume-1/pages/16-4-energy-and-power-of-a-wave#:~:text=The%20equations%20for%20the%20energy,and%20therefore%20the%20frequency%20squared).&text=I%20%3D%20P%204%20%CF%80%20r%202%20.> [Accessed 9 May 2021].

[25] Dev.physicslab.org. 2021. PhysicsLAB: Introduction to Sound. [online] Available at: <http://dev.physicslab.org/Document.aspx?doctype=3&filename=WavesSound_IntroSound.xml#:~:text=Consequently%2C%20a%20sound%20wave%20can,energy%20transferred%20to%20the%20medium.&text=When%20sounds%20reflect%20off%20of,reflection%20is%20called%20an%20echo.> [Accessed 9 May 2021].

[26] Contract Engineering, Product Design & Development Company - Cardinal Peak. 2021. Using Butterworth Filter C++ Class to Implement a Band-Pass Filter, Low Pass Filter & High Pass Filter in C - Contract Engineering, Product Design & Development Company - Cardinal Peak. [online] Available at: <https://www.cardinalpeak.com/blog/a-c-class-to-implement-low-pass-high-pass-and-band-pass-filters> [Accessed 9 May 2021].

[27] Audiouniversityonline.com. 2021. Comb Filtering Explained: What Does a Comb Filter Sound Like? | Audio University. [online] Available at: <https://audiouniversityonline.com/COMB-FILTERING-EXPLAINED/#:~:TEXT=COMB%20FILTERING%20OCCURS%20WHEN%20TWO,ARE%20OUT%20OF%20PHASE%20CANCEL> [Accessed 9 May 2021].

[28] Openacc.org. 2015. OpenACC Programming and Best Practices Guide. [online] Available at: <https://www.openacc.org/sites/default/files/inline-files/OpenACC_Programming_Guide_0.pdf> [Accessed 9 May 2021].

[29] Zlatic, T. and Zlatic, T., n.d. OrilRiver Updated To v2.0 (New GUI + Mac OS Support). [online] Bedroom Producers Blog. Available at: <https://bedroomproducersblog.com/2017/06/13/denis-tihanov-orilriver-2/> [Accessed 10 May 2021].

[30] Eventideaudio.com. 2021. Tverb | Eventide & Tony Visconti Triple Reverb Plugin. [online] Available at: <https://www.eventideaudio.com/products/reverb/visconti-reverb/tverb> [Accessed 10 May 2021].

[31] Uaudio.com. n.d. Ocean Way Studios | UAD Audio Plugins | Universal Audio. [online] Available at: <https://www.uaudio.com/uad-plugins/reverbs/ocean-way-studios.html> [Accessed 10 May 2021].

[32] GitHub. 2021. adamstark/AudioFile. [online] Available at: <https://github.com/adamstark/AudioFile/blob/master/AudioFile.h> [Accessed 9 May 2021].

15. Weekly Log

This section contains rough notes of the accomplishments and some notes key pieces of understanding in each week that has not been previously mentioned..

WEEK 1

Preliminary Research around wrapper formats and ray tracing algorithms.

Drafted Project Proposal.

Other Rough Notes:

Further research is required in selecting suitable engines/implementations of the reverberation room. Further research is required in selecting C++ programming environments and choice of compiler.

This project aims to create a VST 3 plugin using JUCE or similar VST 3 software and create a reverberation room to simulate a variety of acoustic options and microphone placements.

Takes either MIDI audio input or RAW audio input (mp3 etc).

Should have a plethora of acoustic environments/mic placements. For example: simulating stereo overheads when using a single overhead mic, additional room mics and/or hall microphones.

Environments could include but not limited to: Cathedral, sound treated studio, bedroom, outdoors etc.

May be used to simulate a singular room mic to help create a thicker sound.

Research will be carried out in terms of various professional mic placements in a variety of reverberation rooms (acoustic environments (see earlier)).

WEEK 2

Project meeting with supervisor.

Preliminary research into wrapper format fundamentals.

Researched language options. Concluded that C++ is the best language for audio programming as it is what the majority of wrapper formats use, and it is most efficient for these purposes.

A traditional reverberation plugin does not model propagation of sound rather it relies on formulas to calculate reverberation time and utilizes this to model reverberation

Began planning reverberation model.

Written Proposal.

WEEK 3

Practiced going through example projects in JUCE and compiled a VST3 printing hello world.

Examined a project with a gain dial and one more project that allowed the user to upload an audio track.

Researched fundamental principles of reverberation. Summary can be seen above in the background research section.

Examined several rough reverb algorithms.

Continued research into what makes up realistic reverberation and results have been summarized in the requirements.

Noted a reverberation property: “physical feeling of pressure with extremely small rooms.” [5]

This is Inherently created from the near field.

“The near field of a source is the region close to a source where the sound pressure and acoustic particle velocity are not in phase. In this region the sound field does not decrease by 6 dB each time the distance from the source is increased (as it does in the far field). The near field is limited to a distance from the source equal to about a wavelength of sound or equal to three times the largest dimension of the sound source (whichever is the larger).” [13]

In practice this can be implemented by allowing ray amplitude to not decay and no effect will be applied to the sample until the ray has travelled a certain distance and having no frequency response applied to the bit data in the ray sample.

WEEK 4

JUCE code to upload one audio track from example project loaded.

Project Meeting with supervisor

Early design conceptualization: Simple matrix, one microphone, cast 1 ray.

Written basic 2D ray tracer.

For a realistic sounding reverberation effect to be created, Quantec [5] had to deal with the flat tops of the diffuse energy over time causing unwanted reverberation effects. One way they dealt with this was to increase the attack of the emulated reverb the longer each note or sound is played.

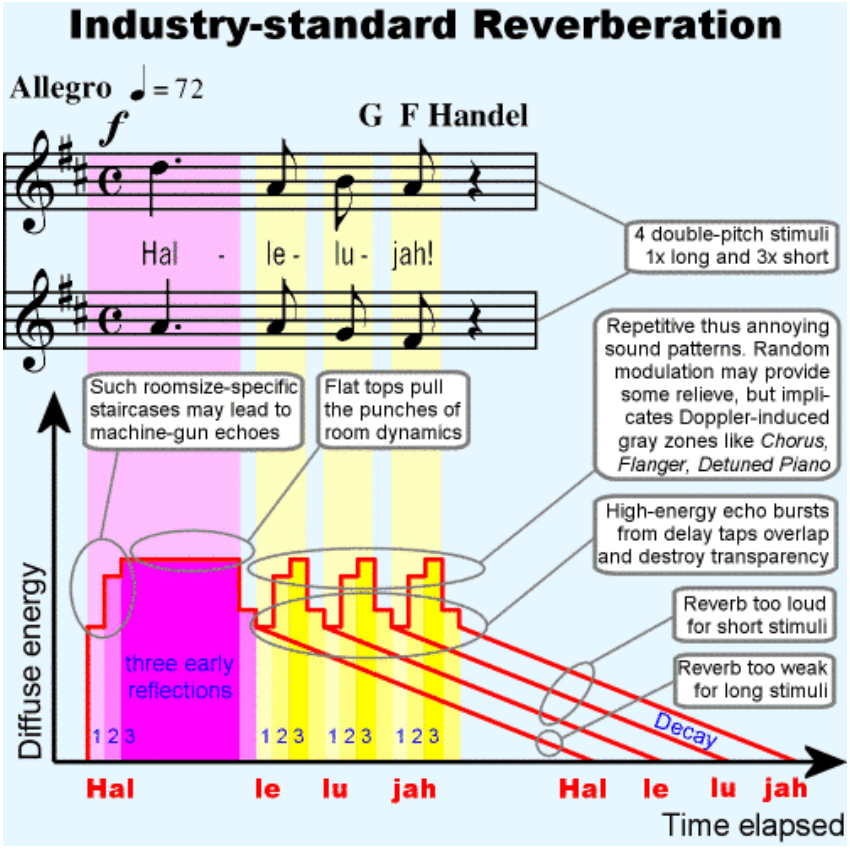


FIGURE 9 QUANTEC IDENTIFICATION OF HALLELUJAH EFFECT IN INDUSTRY-STANDARD REVERBERATION [5]

This problem can be solved in a similar manner by path tracing rays. Rather instead of artificially increasing the attack proportionally to the length of the note, allowing more diffuse energy, the ray tracing approach lends itself to naturally pick up more and more rays containing samples of audio data. The more data it receives, the more data it sounds and the louder the reverberation. This effective and accurate modelling of sound propagation enables an even more organic sounding reverberation effect to be observed.

Another psychoacoustic phenomenon that has been considered is the cocktail party effect. The cocktail party effect is "The "cocktail party effect"—the ability to focus one's listening attention on a single talker among a cacophony of conversations and background noise." [12]

"recognition rates show more than 90% when the signal to noise ratio (SNR) is higher than approximately 5 dB" [7]

By mimicking the reverberation properties of each wall, and modelling the laws governing the conservation of energy, we can enable the user to adjust settings to taste whilst this effect can be observed.

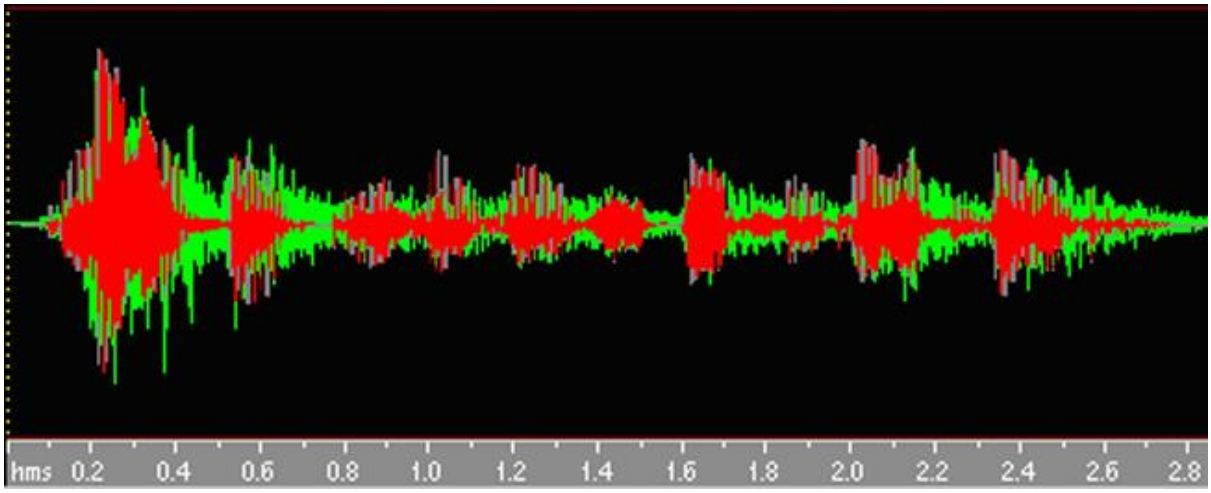


FIGURE 10 AUDIO SIGNAL WITH EXCELLENT EARLY REFLECTIONS AND REVERB[11]

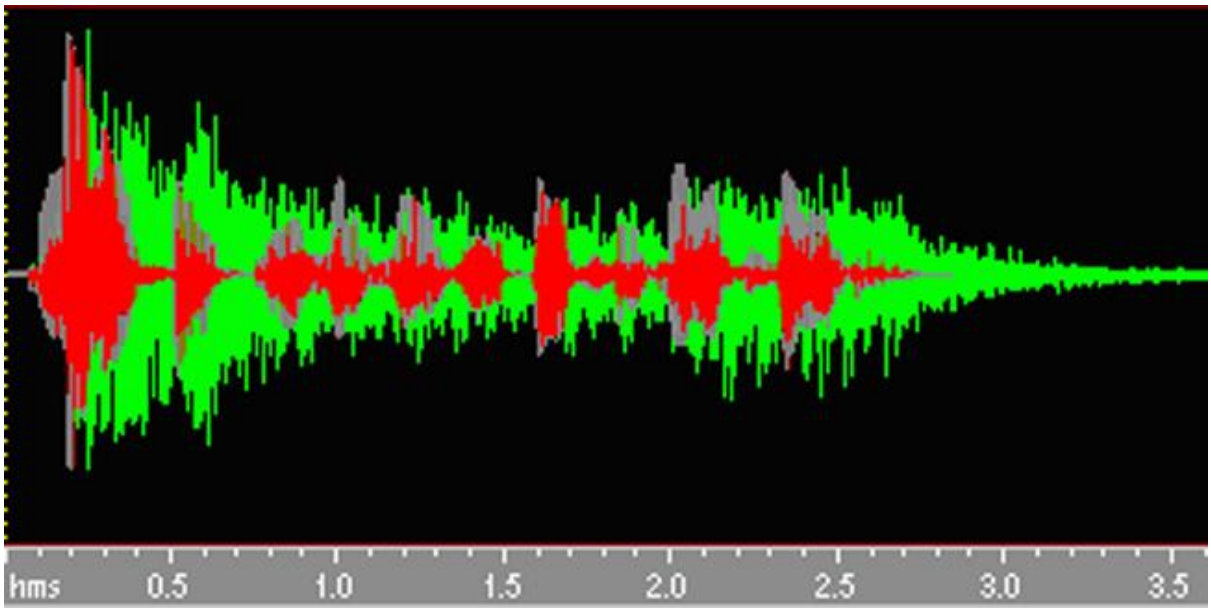


FIGURE 11 AUDIO SIGNAL WITH INFERIOR EARLY REFLECTIONS AND REVERB[11]

Should the walls be too reflective, the early reflections and reverb are very pronounced which can result in the original audio becoming very difficult to distinguish the direct sound and early reflections from the overly resonant late reflections. Another problem with this approach is the Hallelujah effect [5].

WEEK 5

Drafted 3D ray tracer design and concept and created an early high-level design in the form of a UML diagram.

Began work on Interim Report.

WEEK 6

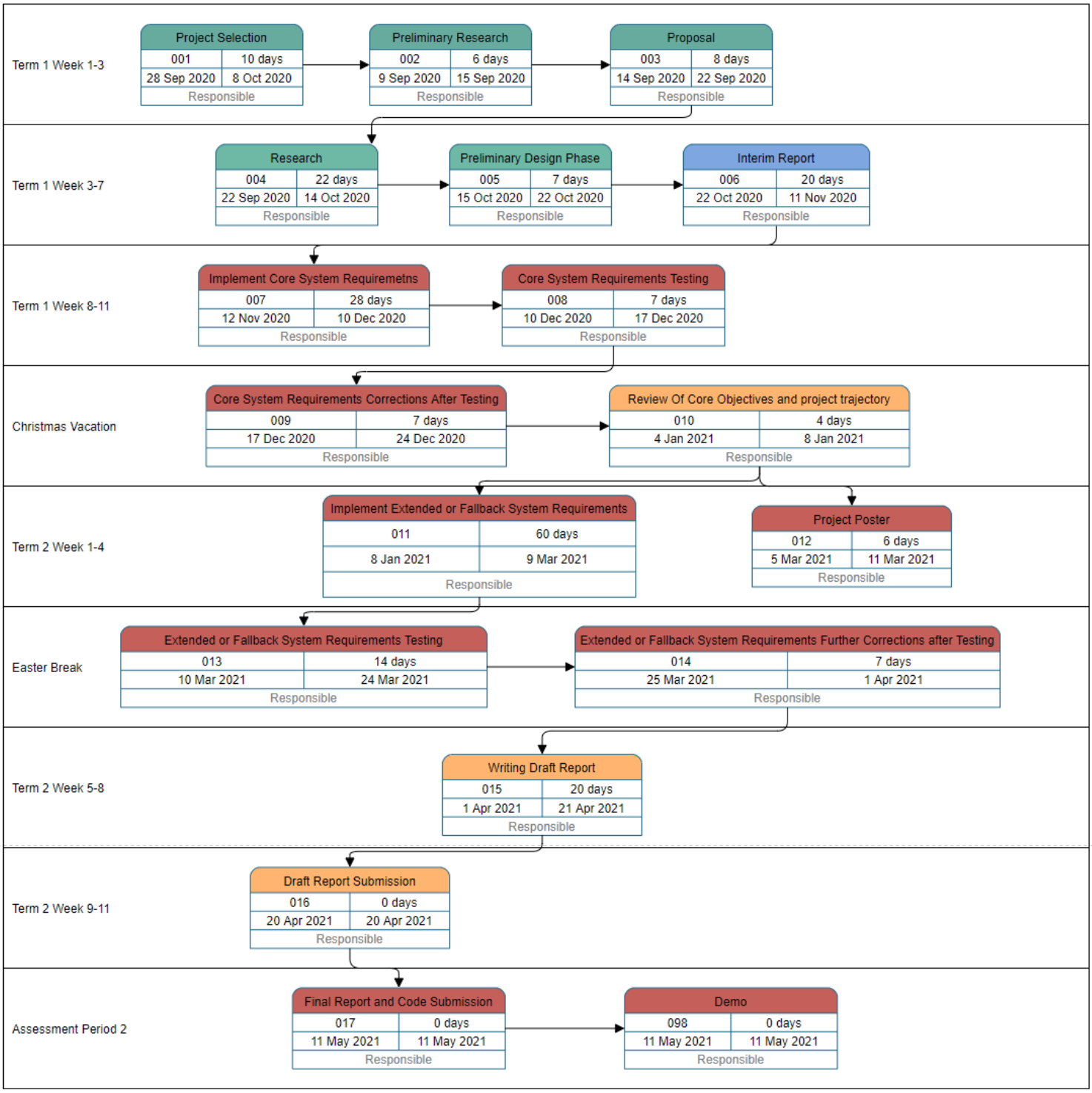
Further work on 3D ray tracer design.

Pressure wave amplitude over distance follows an inverse square law.

WEEK 7

This week has been writing up the interim report. There has been a back and forth of several drafts and feedbacks and a final meeting with Dr Kingsley Sage on Wednesday to discuss any final changes to the interim report.

Interim Report Submitted.



WEEK 8

Begun work implementing basic classes from the initial high level design as proposed in the interim report.

WEEK 9

Further developed ray tracer to allow multiple rays to be cast and rays to be detected within in a 2D cartesian pane.

WEEK 10

Continued work on system requirements.

WEEK 11

Testing currently implemented code is functioning as expected

WINTER VACATION PERIOD

Continued implementation of the core requirements and further iterations on the initial design work and review of progress to date. During this period, current understanding of the project we are creating has been further developed through trial and error. Some key problems to note:

- Immensely CPU intensive. Conclusion that current design is unfeasible so further developed the Ray Tracer Algorithm to have several optimising features.

WEEK 1

- Begin research into real time acceleration techniques.

- Examined many example JUCE projects during this week.

Techniques examined. CUDA, openMP, OMP PARALLEL

WEEK 2

In week

WEEK 3

https://developer.nvidia.com/blog/getting-started-openacc/

https://www.youtube.com/watch?v=wX75Z-4MEoM&ab_channel=NetworkChuck

https://www.youtube.com/watch?v=OohvV_5su2I&ab_channel=TechAbetSolutions

Switched over to linux OS to experiment with openACC IN LINUX USING CENTOS.openACC COMPILER. This process felt cumbersome and unrealistic to learn and implement in the current time frame for the project given the current progression relative to the core and extended requirements so we moved back over to windows and began researching windows based parallelization.

https://www.openacc.org/sites/default/files/inline-files/OpenACC_Programming_Guide_0.pdf

Here we experimented with CUDA runtime and openMP as methods for parallelizing elements of the algorithm with the goal of improving performance. Given the complexity of writing for CUDA, we opted to approach optimization of the algorithm for CPU.

```
#include <cuda_runtime.h>
```

https://www.openmp.org/resources/openmp-compilers-tools/

https://on-demand.gputechconf.com/gtc/2018/presentation/s8344-openmp-on-gpus-first-experiences-and-best-practices.pdf

Research into wrapper formats was continued here.

WEEK 4

https://github.com/adamstark/AudioFile.

Implemented an external audio library as method for reading in audio sources.

Further optimizing runtime as currently runtime was to slow to produce any kind of output in a reasonable amount of time.

Find the sum of last 50 samples iterated through.

Cast one ray with the sum of the last 50 samples.

WEEK 5

Audio playback engine forming Version 5 built in JUCE finalized.

WEEK 6

Creation of Poster.

Ray casting in all directions Version 6-7 Finalized.

WEEK 7

Algorithm adjustments and moved to stereo. Version 8-9 Finalized.

WEEK 8

GUI built Version 10-15 Finalized.

Final Report Writing.

WEEK 9

Microphones implemented and built into GUI Versions 16-21 Implemented.

Final Report Writing.

WEEK 10

Code refactored and tidied up.

Testing.

Final Report Writing.

WEEK 11

Final Build and final report writing.

ASSESSMENT PERIOD 2

Code and Final Report Submission.

16. Appendix

Project Proposal Document

Reverberation Room Proposal

Candidate number: 181334

Candidate Name: Jed Walton

Supervisor: Dr Kingsley Sage

Reverberation Room

"Artificial reverberation is widely used in music production. Its purpose is to make sound appear as it was generated in a realistic physical space. This can be achieved through modelling how sound propagates around a room."

This project will develop a room simulator powered by ray tracing concepts effectively emulating how sound travels from a point source to an observer, bouncing within a modelled acoustic space.

Aims

The project aims to create a reverberation room VST3 plugin to allow greater flexibility in mixing raw microphone input by emulating a selection of reverberation chambers and microphone placements to emulate realistic sounding reverberation.

One of many examples uses of this could be to correct a balance issue in the mix where the drummer plays one instrument (e.g. kick drum) too quietly, in the room mics and overheads, this cannot be corrected easily. Real room mics could be supplemented with artificial room mics from the reverberation room to boost the sound of the quiet instrument (in this example, kick drum) that will allow greater flexibility in mixing room microphones by allowing the room mic mix to be corrected and enhanced to a greater extent than was previously possible.

This will also enable a full room microphone mix when recording with limited microphones where no room mics are available. (A standard drum mic set up is 2 overheads, mic on each tom or a single ribbon microphone for each tom drum, a snare mic or 2 (top and bottom of snare), kick drum microphone)

This could simply enable a one microphone setup to have a full room mix. For example, one unidirectional SM57 microphone against a guitar amp, would be able to artificially create a full room mic setup to pickup the reverberation response for a selection of acoustically treated rooms and microphone options and placements.

This project aims to accurately emulate a variety of acoustic reverberation rooms such as but not limited to: Large hall, medium size hall, acoustically treated hall & emulate a variety of microphone types and placements in relation to a noise source the artificially generated reverberation room.

Core Objectives

1) Implement a basic reverberation room in C++ taking one input audio track, applying reverberation and outputting the audio.
2) Research and study reverberation rooms to replicate real audio waves and improve ray tracing algorithm in core objective 1.
3) Embed reverberation room into a VST3 plugin wrapper format to be used in any modern DAW supporting VST3 plugins.
4) Implement graphical user interface using HCI design pricipals allowing user basic controls of the VST3 plugin. (Size/type of reverberation room/gain/volume etc).
5) Replicate a selection of types of rooms and acoustic options & acoustic properties of the rooms.

Extended Objectives

2) Allow users to add, remove and relocate microphones (reverberation perceptors) within the 3D space relative to the point sound source.
3) Emulate a plethora of microphone varieties. (Ribbon, omnidirectional, unidirectional… etcetera).
4) [1] Improve room to concurrently emulate the real-acoustic and psychoacoustic phenomena: [1] "
    3.1 Unobtrusive initial reflections
    3.2 Gradual build-up
    3.3 Strict adherence to the law of conservation of energy
    3.4 physical feeling of pressure with extremely small rooms
    3.5 The inherent subsonic tremble of huge rooms
    3.6 history-based random interferences – no random generators
    3.7 Capture acoustic sequences (e.g. trills) along extended propagation paths
    3.8 Hundreds of positional room mode spectra across the sound field
    3.9 Cocktail party effect.
    3.10 Hallelujah effect."[1]
5) Work in real time utilizing Non-Ray Tracing enabled hardware. (CPU or GPU) and allow the user to tune the number of rays to run a real time model so that it is able to run on the majority of modern hardware.
6) Work in an offline non-real-time manner for DAW systems with limited computational resources (CPU). (lowering the number of rays may enable runtime operation in limited computational resourced DAW systems & then run the ray tracing algorithm in an offline manner). Another option could be to create a standalone for offline use.

Fallback Objectives

1) Get a Basic VST3 plugin working
2) Create a basic reverberation room with only 10 rays and one perceptor (emulated microphone) that works in real time
3) Create a simple VST3 plugin with few controls but basic functionality exists.
4) Research and provide methodology for implementing any missed core and extended objectives and design development plans to implement this.

Relevance

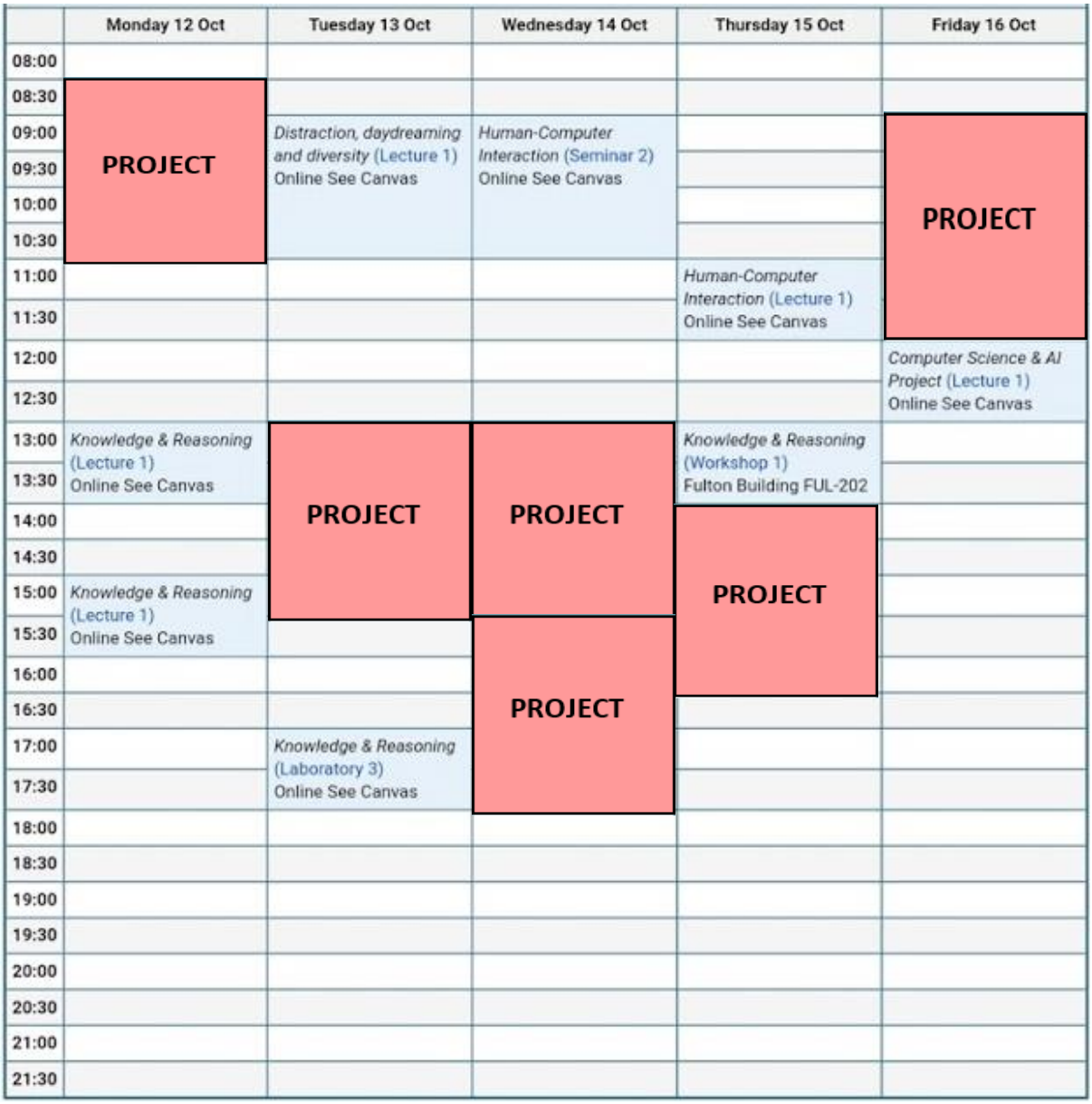
This project will go in depth with core-computing skills specifically programming in C++ for VST3 wrapper format, software engineering, HCI design principals, user-testing. Not only this, the project will test my ability as a programmer, it will also test my ability to manage an agile-developed project focused

around user-centred design. The project requires elements of acoustics and audio engineering demonstrating the application of core-computing skills to create an interdisciplinary application.

Resources Required

- Project will require access to a ray tracing enabled GPU and a multithread CPU.
- Wrapper format to create the VST3 plugin such as the wrapper format, “Projucer”.
- Project will require interviews with audio engineers to better understand the problem space.
- Require access to a C++ environment and compiler.

Timetable



Background reading/Bibliography

[1] Quantec.com. 2020. *QUANTEC: Room Simulation*. [online] Available at: <http://www.quantec.com/index.php?id=room_simulation> [Accessed 12 October 2020].

[2] Redwoodaudio.net. 2020. *JUCE For VST Development - Setup*. [online] Available at: <http://www.redwoodaudio.net/Tutorials/juce_for_vst_development__intro2.html> [Accessed 12 October 2020].

[3] Prata, S., n.d. *C++ Primer Plus*. 6th ed.

[4] NVIDIA Developer. 2020. *Vrworks - Audio*. [online] Available at: <https://developer.nvidia.com/vrworks/vrworks-audio> [Accessed 12 October 2020].

[5] NVIDIA Developer. 2020. *Vrworks Audio SDK In-Depth*. [online] Available at: <https://developer.nvidia.com/vrworks-audio-sdk-depth> [Accessed 12 October 2020].

[6] Docs.juce.com. 2020. *JUCE: Tutorial: Create A Basic Audio/MIDI Plugin, Part 1: Setting Up*. [online] Available at: <https://docs.juce.com/master/tutorial_create_projucer_basic_plugin.html> [Accessed 12 October 2020].

[7] Kvraudio.com. 2020. *KVR Forum: How To Create VST Plugins? Information For Those Just Getting Started*. [online] Available at: <https://www.kvraudio.com/forum/viewtopic.php?t=329696> [Accessed 12 October 2020].

[8] Kvraudio.com. 2020. *KVR Forum: Recommend A (Printed) Book For Learning C++ Please*. [online] Available at: <https://www.kvraudio.com/forum/viewtopic.php?t=345702> [Accessed 12 October 2020].

[9] Kvraudio.com. 2020. *KVR Forum: What Are The Most Important Parts Of C++ For Coding Plug-Ins?*. [online] Available at: <https://www.kvraudio.com/forum/viewtopic.php?t=52342&start=0> [Accessed 12 October 2020].

[10] Forum.cockos.com. 2020. *Iplug C++ Code Framework For VST And AU Audio Plugins And Guis - Cockos Incorporated Forums*. [online] Available at: <https://forum.cockos.com/showthread.php?t=9524> [Accessed 12 October 2020].

[11] Teragonaudio.com. 2020. *How To Make VST Plugins In Visual Studio*. [online] Available at: <http://teragonaudio.com/article/How-to-make-VST-plugins-in-Visual-Studio.html> [Accessed 12 October 2020].

[12] Cs.unc.edu. 2020. *Ray Tracing: Graphics For The Masses*. [online] Available at: <https://www.cs.unc.edu/~rademach/xroads-RT/RTarticle.html> [Accessed 12 October 2020].

[13] 2020. *Mixing Gerald Heyward | Ross Rothero-Bourge*. [image] Available at: <https://www.youtube.com/watch?v=PiXjBvPNCPw&ab_channel=VicFirth> [Accessed 12 October 2020].

[14] Reaper.fm. 2020. *REAPER | Audio Production Without Limits*. [online] Available at: <https://www.reaper.fm/> [Accessed 12 October 2020].

Interim log (see log documents for more in depth)

Week 0:

Reading around topic and researching potential project directions.

Week 1:

First Lecture on proposal. Further background research and exploring project directions.

Week 2:

Met with supervisor. Further research into project direction. Worked on project proposal & set out some early Core, Extended and Fallback Objectives and Finished Project Proposal.

Week 3:

REVERBERATION ROOM

Final Report

MAY 11, 2021

CANDIDATE NUMBER: 181334

University of Sussex – Computing Sciences with Artificial Intelligence

Statement of Originality

This is to certify the content of this dissertation is, to my best knowledge, a product of my own labour except where indicated clearly. This report is submitted as a requirement for the Computer Sciences & Artificial Intelligence BSc (Hons) degree at the University of Sussex.

This work is based 'Reverb Room Simulator' project first conceptualized by Dr Kingsley Sage and has been continually developed under guidance throughout my third and final year at the university.

Acknowledgements

I would like to thank Dr Kingsley Sage for his immeasurable contribution, continued patience and thorough guidance in supervising this project.

Contents

Introduction

"Artificial reverberation is widely used in music production. Its purpose is to make sound appear as it was generated in a realistic physical space. This can be achieved through modelling how sound propagates around a room."

Objectives

The objective of this project is to design and develop a reverberation room plugin for digital audio workfaces powered by ray tracing concepts to accurately emulate a variety of acoustic reverberation rooms including but not limited to: Large hall, medium size hall, dampened hall & emulate a variety of microphone types and placements in relation to a noise source within an artificially generated reverberation room.

The project will be made using both a compiled language and wrapper format. The compiled language of C++ is well suited as it produces machine language than can directly understood by the system resulting in a highly efficient runtime. The use of a wrapper format enables multiple data streams to be embedded into an application programming interface such as: VST2, VST3, AUv2, AUv3, AAX for use in any digital audio workstations that supports any of the supported application programming interface.

In this project, users will be able to apply ray tracing powered reverberation to an audio track within any digital audio workstation that supports the application programming interface. The reverberation is achieved by modelling the propagation of sound moving in within a virtual acoustic space or reverberation chamber by modelling propagation of sound moving in space by tracing rays.

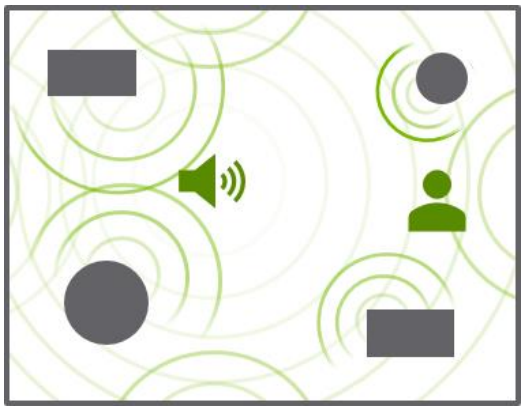


FIGURE 9 PROPAGATION OF SOUND MOVING IN SPACE [1]

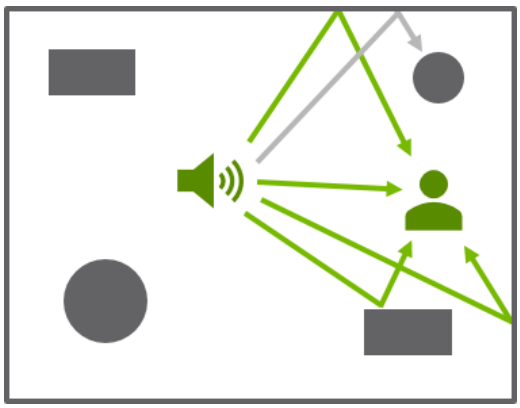


FIGURE 10 RAY TRACED AUDIO BY MODELLING PROPAGATION OF SOUND MOVING IN SPACE [1]

Professional Considerations

This project consists of work produced with, to my best knowledge, strict adherence to the Code of Conduct published by BCS – The Chartered Institute for IT [2]. The code of conduct has been familiarized and strictly adhered to my best knowledge. The following statement has taken from the official BCS website and modified to apply this project to ensure it has been thoroughly understood and applied to thoughtfully to this project.

PUBLIC INTEREST:

> *I shall have due regard for public health, privacy, security and wellbeing of others and the environment. I shall have due regard for the legitimate rights of third parties by using only licence permitting third party material and referencing it clearly using Harvard style referencing. Any professional activities occur without discrimination on the grounds of sex, sexual orientation, marital status, nationality, colour, race, ethnic origin, religion, age or disability, or of any other condition or requirement. This project promotes equal access to the benefits of IT and seek to promote the inclusion of all sectors in society wherever opportunities arise.*

PROFESSIONAL COMPETENCE AND INTEGRITY

> *I shall only undertake to do work or provide a service that is within my professional competence and NOT claim any level of competence that I do not possess. This project will allow me to develop my professional knowledge, skills and competence on a continuing basis whilst maintaining awareness of technological developments, procedures, and standards that are relevant to my field. In doing so, I shall ensure that I have the knowledge and understanding of relevant legislation and I shall comply with such legislation, in carrying out any professional responsibilities. I both respect and value alternative viewpoints and seek, accept and offer honest criticisms of work. I shall avoid injuring others, their property, reputation, or employment by false or malicious or negligent action or inaction and reject and will not make any offer of bribery or unethical inducement.*

DUTY TO RELEVANT AUTHORITY:

> *I shall carry out my professional responsibilities with due care and diligence in accordance with the University of Sussex requirements while exercising my professional judgement at all times. I shall actively seek to avoid any situation that may give rise to a conflict of interest between me and my relevant authority. I accept professional responsibility for my work and this project will be entirely product of both my own work and input from my technical supervisors. I shall NOT disclose or authorise to be disclosed, or use for personal gain or to benefit a third party, confidential information except with the permission of the University of Sussex, or as required by legislation. I will NOT misrepresent or withhold information on the performance of products, systems or services (unless lawfully bound by a duty of confidentiality not to disclose such information) or take advantage of the lack of relevant knowledge or inexperience of others.*

DUTY TO THE PROFESSION:

> *I accept my personal duty to uphold the reputation of the profession and will not take any action which could bring the profession into disrepute. I seek to improve professional standards through participation in their development, use and enforcement. I shall do everything in my power to uphold the reputation and good standing of BCS, The Chartered*

Institute for IT and act with integrity and respect in my professional relationships with all members of BCS and with members of other professions with whom I work in a professional capacity. I shall notify BCS if convicted of a criminal offence or upon becoming bankrupt or disqualified as a company director and in each case give details of the relevant jurisdiction and I shall encourage and support fellow members in their professional development.

Background and User Research

Background Research

Key principals of realistic reverberation:

a. Direct sound
b. Early reflections.
c. Late reflections.
d. Gradual build-up.
e. Reverb Tail.
f. Frequency response.
g. Strict adherence to the law of conservation of energy.
h. “Physical feeling of pressure with extremely small rooms.” [5]
i. “The inherent subsonic tremble of huge rooms.” [5]

User Research

This reverberation software project is designed with the aim of providing a tool for Musicians and or Audio Engineers to allow them to emulate realistic reverberation audio effects within modern digital workstations or as a standalone application.

A common scenario that engineers and musicians alike find themselves in is having to balance the live recording tracks from close microphones with ambient room microphones to create a balanced and great sounding mix capturing the acoustics of the room.

Usually, if one instrument or voice is played too quietly it can easily be partially corrected by adjusting the gain of the close microphone of the instrument that needs correcting. However, this is not easy to correct in the room mix as it picks up the ambience of the entire room. [6]

An existing solution to this, even when a very realistic sounding reverberation plugin such as Quantec [5] is deployed for this very purpose, it can be tough to match this reverberation sound to that captured in the room mix. This is because room microphones pick up sound relative to the location of the sound source and this reverberation applied is one dimensional.

On top of this, in many budget recording setups, room microphones are not available to the musician or audio engineer due to having access to a limited number of microphones. This software will enable users to emulate additional room microphones and apply a realistic reverberation effect

Current reverberation applications such as Quantecs’ reverberation simulator [5] apply reverberation to the track creating a realistic sounding reverb but do not emulate different microphone types nor placements within the reverberation chamber. This means emulating a stereophonic sounding reverberation effect with user defined microphone placements is next to impossible to pull off convincingly. In the proposed system, due to modelling propagation of sound accurately via ray tracing, this software will be able to create a stereo recording based on the emulated microphone placing relative to the sound source whilst applying a realistic sounding reverberation.

This enables the budget studio the ability to emulate a professional stereo, multi microphone, recording in a variety of acoustically treated rooms with very limited recording equipment.

Another issue is the Hallelujah effect is a problem the traditional non ray tracing reverberation algorithm first observed by Quantec in 2010 [5]. They solved this by increasing the attack of the note or sound the longer the note plays and then delaying the launch of this sound to produce a realistic

reverberation effect. However, there are some inaccuracies in this process. From the 2010, this was the only reverberation software that identified and took steps to solve this issue. By modelling sound propagation via a ray tracing model of sound propagation, along with strict adherence to the physical laws governing reverberation of pressure waves, we can solve the hallelujah effect naturally without excessive corrections needing to be applied to achieve a greater level of realism in the reverberation effect. This is achieved by simply firing rays for every sample of the sound, each ray containing the bit information of the sample which is then reconstructed into audio data at the microphones.

Requirements

System Functional Requirements

To ensure the user requirements can be met, we give rise to the following system requirements.

Core System Requirements

For this project to be considered successful, the following core system requirements must be met.

5) System allows audio upload and playback to apply ray tracing reverberation to an audio track in an offline manner and then be able to render the audio track with the reverberation effect applied using the CPU.
    a. The user can select from a selection of sample rates and bit depths for the output audio.
    b. The audio effect will monophonic and generated from a single listening point.
    c. Reverberation room shall be 2 dimensional.
    d. User can upload an audio track for processing within the software.
    e. Playback the uploaded track.
    f. The user should be able to render the audio track with the reverb effect applied.
6) System should form a digital audio workstation plugin.
7) Create several pre-configured microphone placement and room type configurations.
8) Implement a GUI and code to allow user to adjust:
    a. Sample rate of reverberation audio track forming the number of rays traced per second.
    b. The bit depth of each sample.
    c. Gain.
    d. Volume.
    e. pre-configured microphone placement and room type.

Extended Requirements

Should time permit, and after all the core requirements have been met, the following extended requirements can be met. For these extended system requirements to be met, after all the core system requirements have been met; the system shall:

6) The system allows user to manually add and relocate existing microphones and sound sources within the 3D space.
7) Create a class for each microphone type:
    a. Cardioid
    b. Omnidirectional
    c. Figure-8
8) Enable Directional Reverberation (Stereo) based on where the microphone/microphones are placed within the room relative to the sound source.
9) The reverb algorithm should be improved to emulate the following key features to creating reverberation:
    a. Direct sound
    b. Early reflections.
    c. Late reflections.
    d. Gradual build-up.
    e. Reverb Tail.
    f. Frequency response.

g. Strict adherence to the law of conservation of energy.
h. “Physical feeling of pressure with extremely small rooms.” [5]
i. “The inherent subsonic tremble of huge rooms.” [5]

10) Research optimized runtime utilizing CUDA acceleration technology [14][15] to speed up render times for rendering out the audio track into a selection of audio format with options of both sample and bitrates.

Fallback System Requirements

The system shall:

4) Exist as a Plugin or standalone in any format that allows a user to upload an mp3 audio track and apply a basic reverberation effect with a monophonic output.
5) Basic controls forming a GUI:
    a. Sample rate of reverberation audio track forming the number of rays traced per second.
    b. The bit depth of each sample.
    c. Stereo or mono input audio file and outputting a monophonic audio track.
    d. Gain.
    e. Volume.
    f. pre-configured microphone placement and room type.
6) Select from a limited selection of microphone placements and room types.

Project plan

Here is a program evaluation and review technique (PERT) chart forming a project plan. During the implementation stages, an agile approach should be deployed detailing sprint progress along with testing. During the implementation stage, a GitHub Repository shall be utilized to help document the development process

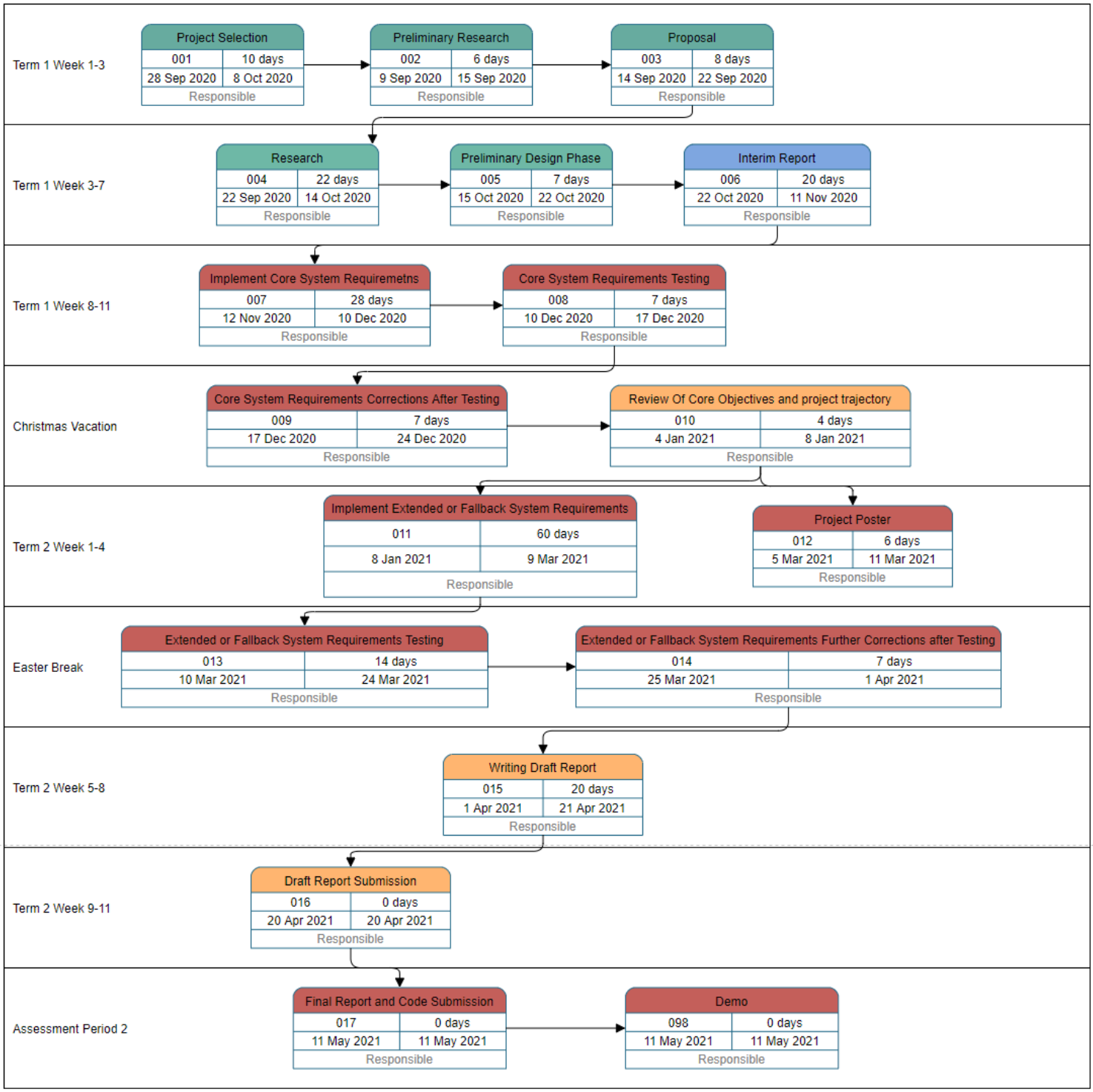


FIGURE 11 PERT CHART DETAILING OVERVIEW OF PROJECT PLAN

Progress to date

Reverberation

Reverberation is the prolongation of a sound through resonance. Reverberation time is the measure of time for the sound to decay completely in an enclosed area after the source of the sound has stopped.

A formula was first devised and developed by Wallace Clement Sabine[3] allowing the reverberation time for a room to be calculated.

T = reverberation time (s) , c = sound speed, V = volume and S = total interior surface area of the enclosure in question respectively, αa = averaged absorption coefficient of the enclosure,

αi = individual absorption coefficient of each subsurface, Si.

$$T = 13.8\frac{4V}{\mathrm{c\,S}\,\alpha_a}, \text{ with } \alpha_a = \frac{\sum_i \alpha_i S_i}{\sum_i S_i},$$

FIGURE 12 [3]

Henry Eyring pointed out the need for a more general formula for enclosed spaces with highly absorptive conditions.

$$T = 13.8\frac{4V}{-\mathrm{c\,S}\ln(1-\alpha_a)}.$$

FIGURE 13 [3]

This forms the fundamental information required for non-ray tracing powered reverberation programs to model reverberation in each room size.

Ray Trace Reverberation

We can emulate reverberation accurately by modelling sound propagation using a ray tracing approach. To do this we will first need to calculate the speed of sound in air which can be given by the formula:

V = speed (m/s), Tc = Temperature (°C).

V = 331 + 0.59 Tc

Given that a reasonable room temperature is 20 °C. We calculate the speed of sound in air for our model to equal:

V = 331 + 0.59 * 20

V = 342.8 m/s

V = 343 m/s (rounded to nearest integer)

We can use V to calculate the time taken for the ray to reach the microphone from the source. This is the delay for the one sample to reach the microphone.

We know that a pressure wave otherwise known as a sound wave loses amplitude with distance following an inverse square law, so we can calculate the amplitude of the ray with the distance it has travelled. We can also define how much the wave will decay upon reflecting off any given surface. This can give us an output amplitude.

Here we can see how sound pressure or volume decays over time.

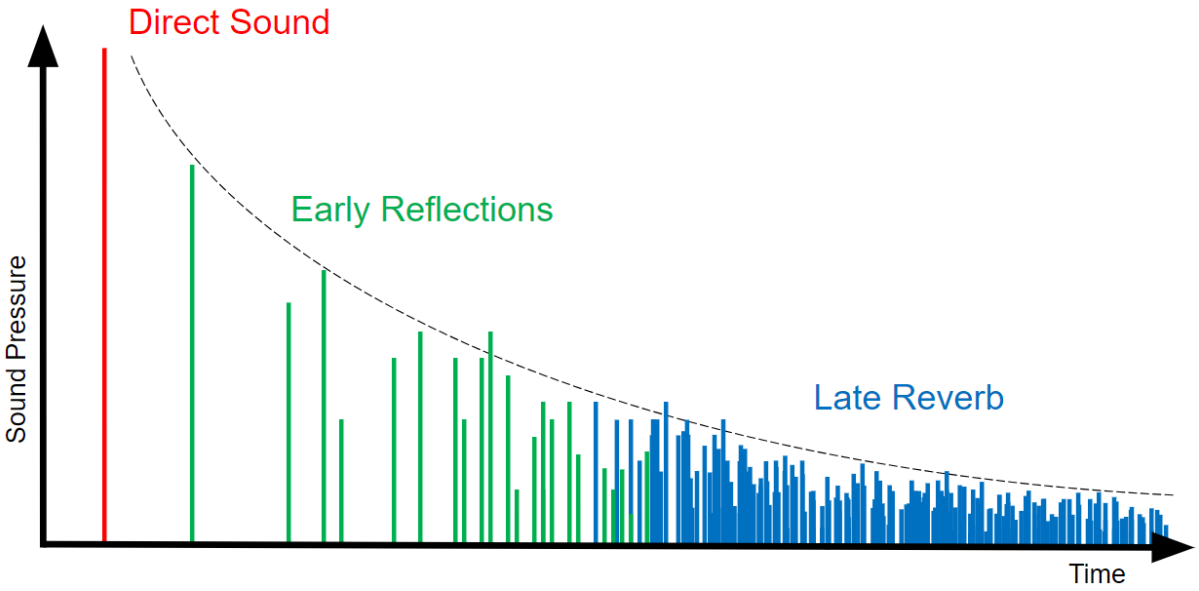


FIGURE 6 GRAPH SHOWING SOUND PRESSURE OVER TIME WITH LABELLED DIRECT SOUND AND EARLY REFLECTIONS [4]

We project rays in every direction from a point sound source. Each ray will be a class object storing information on its current location, sample number of the audio file, distance travelled and current amplitude.

Key principals of realistic reverberation:

a. Direct sound
b. Early reflections.
c. Late reflections.
d. Gradual build-up.
e. Reverb Tail.
f. Frequency response.
g. Strict adherence to the law of conservation of energy.
h. “Physical feeling of pressure with extremely small rooms.” [5]
i. “The inherent subsonic tremble of huge rooms.” [5]

The following UML design is intended to be scalable to include the given principals.

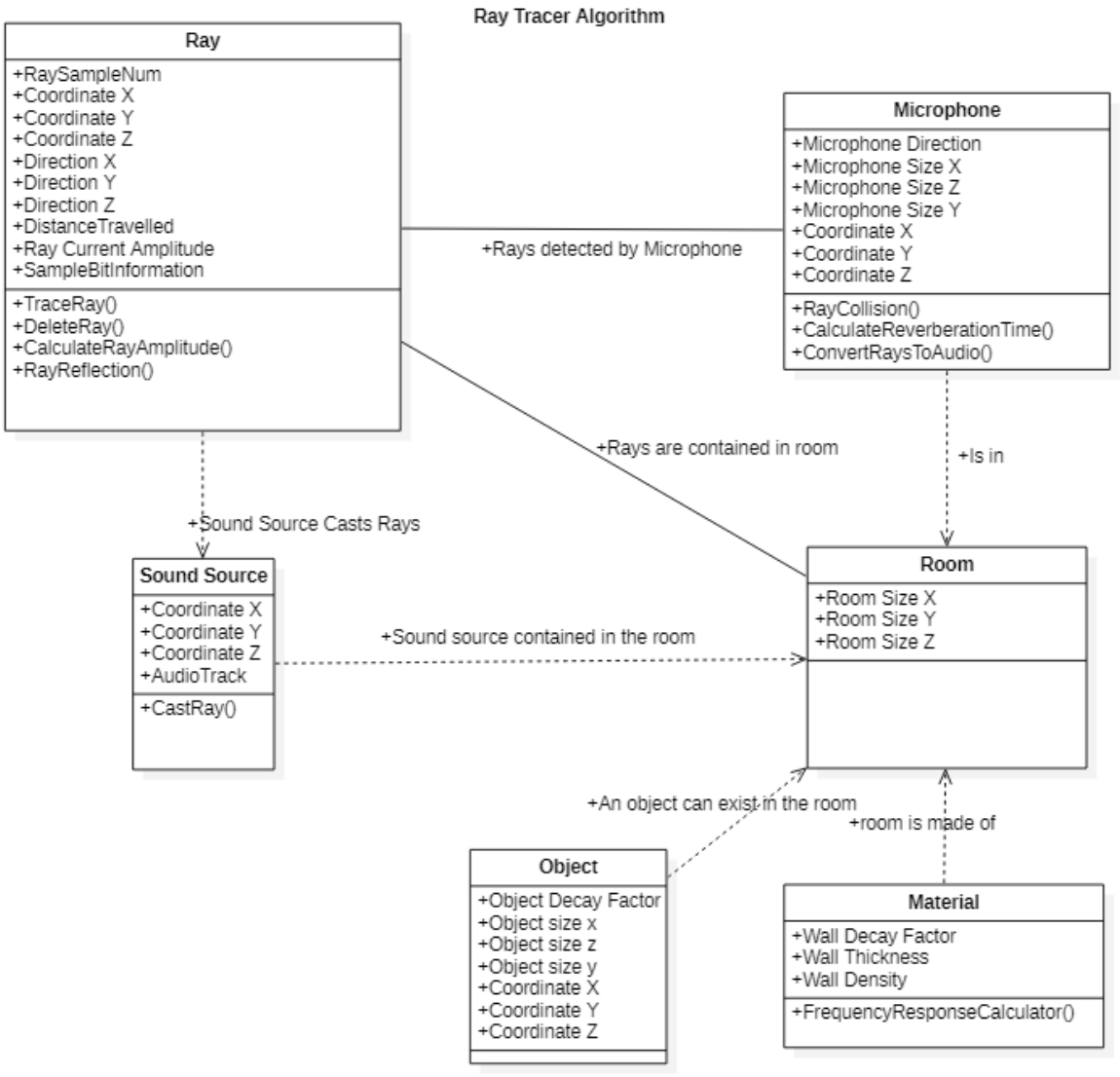


FIGURE 7 PRELIMINARY RAY TRACER DESIGN UML

We define the room dimensions as a 3D matrix, in this example, we will give the room dimensions as label X, Y and Z.

Now the ray will continue to move in the direction in the matrix as specified by the ray direction variable in the ray trace class updating the current coordinates and the Ray Distance Travelled as it goes.

Should the ray current coordinates x, y or z exceed X, Y or Z respectively, the ray should change direction in the axis where coordinate x, y or z exceeded X, Y or Z. This allows the angle of incidence to equal the angle of reflection. When the ray reflects from a wall, a method call takes place in the material object to calculate the frequency response of the wall that will be induced on the ray sample bit information stored in the ray object. This frequency response is governed by wall thickness, size, and density. The ray amplitude will reduce by the wall decay factor. On top of this, the ray amplitude or sound pressure, also known as the intensity of the sound decays with distance following an inverse square law with distance. This is accounted for while tracing the individual ray, the amplitude will decay with distance and initially depends on the near field size of the given pressure wave frequency.

Next a ray detector or microphone as we will implement as, can be added at a location within matrix X, Y, Z and given a size in the form of a smaller matrix. Now should the ray currently in the process of being traced (x, y, z) be within the defined microphone placed size by coordinate, the ray can flag up as detected by the microphone and we read the sample number and use that to retrieve the bit data for that sample which is then used to form a new, output sample to compose a new, output audio track.

We can calculate the time taken for the ray to reach the microphone from the source using the information in the ray object. This is the delay for the one sample to reach the microphone.

As we know the bit data for the sample forming the ray that collided with the microphone and the time it arrived, we can play use this to form output audio with a reverberation effect applied.

Wrapper Formats

As we will be targeting this project for audio engineers and musicians it is important to ensure the target users will be able to utilize the plug in within their preferred workstation. In order to do so, we need to select a wrapper format or frameworks to allow the same source code to compile and run on as many platforms as possible whilst having a clear graphical user interface for the users to use the program. After broad research into DAW plugin wrapper formats, three potential solutions have been overviewed.

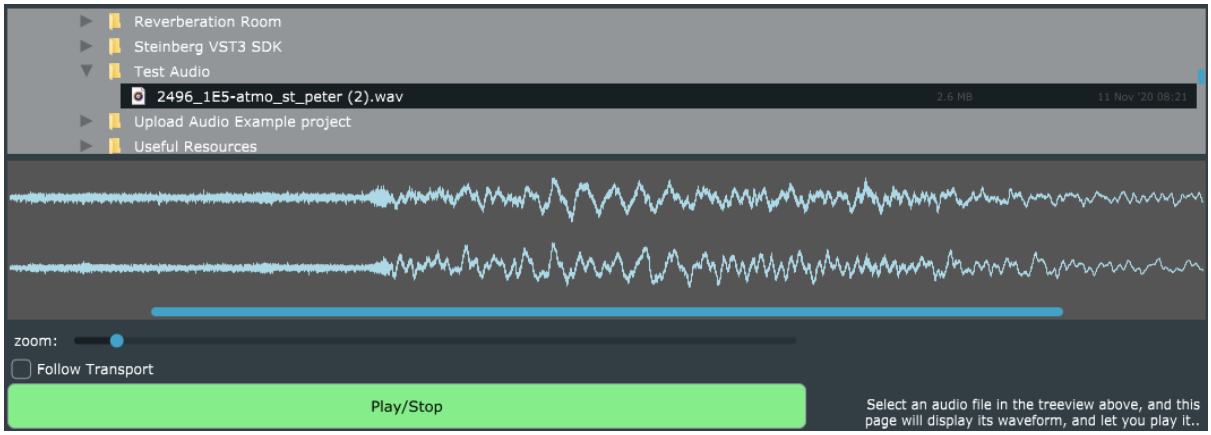


FIGURE 8 JUCE [8] DEMO PROJECT ALLOWING AUDIO TO BE UPLOADED AND PLAYED BACK.

JUCE [8] allows the same C++ source code to compile and run identically on Windows, macOS, Linux, and even mobile platforms as a plugin, mobile application and standalone desktop or even web applications.

There are free personal and free education licences. We can use the Educational use licence, which has no revenue or funding limit, and it is free to use. On top of this there are many example projects and a multitude of clear documentation and tutorials on the website. The only notable downside discovered at this stage is the Splash Screen watermark displaying 'Made with JUCE' with the free versions.

iPlug 2 [9] allows the same C++ source code to compile and run identically on Windows, Mac and Linux. This includes any platform that can utilize the VST2, VST3, AUv2, AUv3, AAX (Native) and the Web Audio Module (WAM) plug-in APIs. This is very similar to JUCE with marginally less support for the added benefit of not have the watermark on the educational license version. On their official page, they state that they do not consider iPlug 2 to be production ready. This means there are likely

some problems with the software. For the only foreseeable benefit of choosing iPlug2 [9] over JUCE [8] is the lack of watermark, at the cost of cross platform support, and an increased risk of encountering an unexpected software problem that could interfere with the project progression, the preference currently remains JUCE [8].

jVSTwRapper [10] is another option to create the plugin however this time using the language of Java. This has the added benefit of allowing relatively easier Graphical User Interface (GUI) programming as C++ is known to be relatively cumbersome when creating GUI windows.

However, the alternative C++ approach, JUCE [8], has a huge plethora of information available on creating GUIs. Overall, this should result in a similar development time for both approaches. As Java is an interpreted language, it will inherently lead to a slower runtime than C++ as it requires a Java virtual machine to interpret the code at runtime whereas C++ is compiled directly into machine code resulting in more efficient runtime. Should the extended objective of optimizing for CUDA acceleration [14] be fulfilled, we shall need to use the language of C or C++. This is hugely important to optimize for runtime due to the computationally intensive nature of the project. This necessitates C++, thus ruling out jVSTwRapper.

After careful evaluation of the wrapper formats; JUCE [8], iPlug 2 [9] and jVSTwRapper [10], we conclude that JUCE [8] is the best choice of wrapper format as it maximize the chances completing the system objectives to the greatest possible extent.

Weekly Log

This section contains rough notes of the accomplishments and some notes key pieces of understanding in each week that has not been previously mentioned. To prevent repeating information and help coherence of the report, the log has been simplified and some information exists only in previous sections.

Week 1

Preliminary Research around wrapper formats and ray tracing algorithms.

Drafted Project Proposal.

Other Rough Notes:

Further research is required in selecting suitable engines/implementations of the reverberation room. Further research is required in selecting C++ programming environments and choice of compiler.

This project aims to create a VST 3 plugin using JUCE or similar VST 3 software and create a reverberation room to simulate a variety of acoustic options and microphone placements.

Takes either MIDI audio input or RAW audio input (mp3 etc).

Should have a plethora of acoustic environments/mic placements. For example: simulating stereo overheads when using a single overhead mic, additional room mics and/or hall microphones.

Environments could include but not limited to: Cathedral, sound treated studio, bedroom, outdoors etc.

May be used to simulate a singular room mic to help create a thicker sound.

Research will be carried out in terms of various professional mic placements in a variety of reverberation rooms (acoustic environments (see earlier)).

Week 2

Project meeting with supervisor.

Preliminary research into wrapper format fundamentals.

Researched language options. Concluded that C++ is the best language for audio programming as it is what the majority of wrapper formats use, and it is most efficient for these purposes.

A traditional reverberation plugin does not model propagation of sound rather it relies on formulas to calculate reverberation time and utilizes this to model reverberation

Began planning reverberation model.

Written Proposal.

Week 3

Practiced going through example projects in JUCE and compiled a VST3 printing hello world.

Examined a project with a gain dial and one more project that allowed the user to upload an audio track.

Researched fundamental principles of reverberation. Summary can be seen above in the background research section.

Examined several rough reverb algorithms.

Continued research into what makes up realistic reverberation and results have been summarized in the requirements.

Noted a reverberation property: “physical feeling of pressure with extremely small rooms.” [5]

This is Inherently created from the near field.

“The near field of a source is the region close to a source where the sound pressure and acoustic particle velocity are not in phase. In this region the sound field does not decrease by 6 dB each time the distance from the source is increased (as it does in the far field). The near field is limited to a distance from the source equal to about a wavelength of sound or equal to three times the largest dimension of the sound source (whichever is the larger).” [13]

In practice this can be implemented by allowing ray amplitude to not decay and no effect will be applied to the sample until the ray has travelled a certain distance and having no frequency response applied to the bit data in the ray sample.

Week 4

JUCE code to upload one audio track from example project loaded.

Project Meeting with supervisor

Early design conceptualization: Simple matrix, one microphone, cast 1 ray.

Written basic 2D ray tracer.

For a realistic sounding reverberation effect to be created, Quantec [5] had to deal with the flat tops of the diffuse energy over time causing unwanted reverberation effects. One way they dealt with this was to increase the attack of the emulated reverb the longer each note or sound is played.

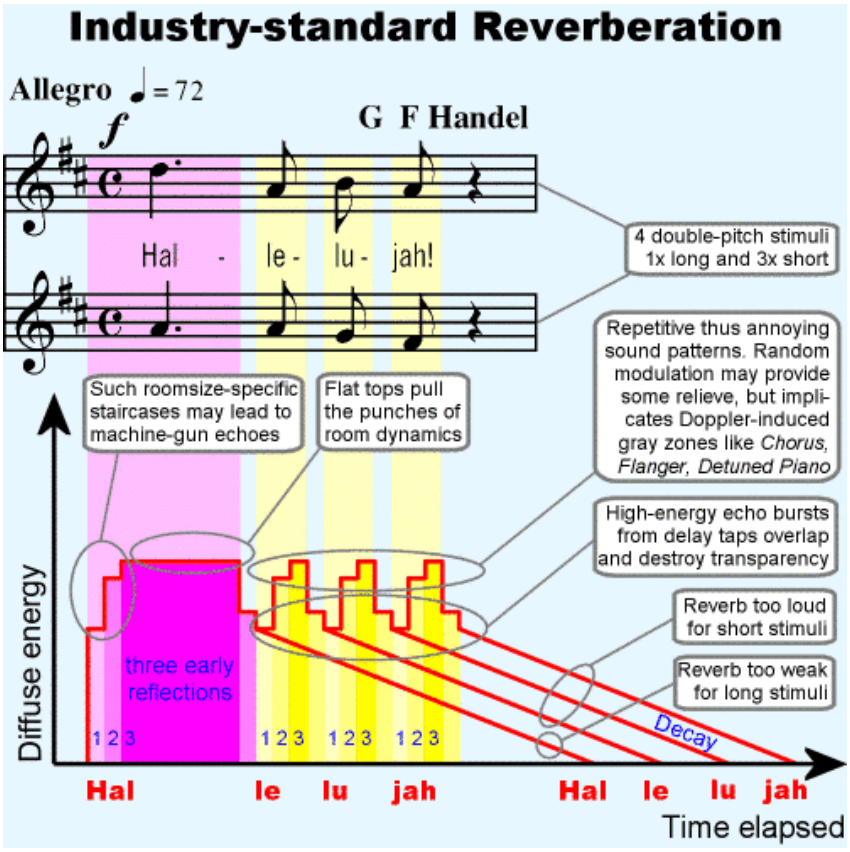


FIGURE 9 QUANTEC IDENTIFICATION OF HALLELUJAH EFFECT IN INDUSTRY-STANDARD REVERBERATION [5]

This problem can be solved in a similar manner by path tracing rays. Rather instead of artificially increasing the attack proportionally to the length of the note, allowing more diffuse energy, the ray tracing approach lends itself to naturally pick up more and more rays containing samples of audio data. The more data it receives, the more data it sounds and the louder the reverberation. This effective and accurate modelling of sound propagation enables an even more organic sounding reverberation effect to be observed.

Another psychoacoustic phenomenon that has been considered is the cocktail party effect. The cocktail party effect is "The "cocktail party effect"—the ability to focus one's listening attention on a single talker among a cacophony of conversations and background noise." [12]

"recognition rates show more than 90% when the signal to noise ratio (SNR) is higher than approximately 5 dB" [7]

By mimicking the reverberation properties of each wall, and modelling the laws governing the conservation of energy, we can enable the user to adjust settings to taste whilst this effect can be observed.

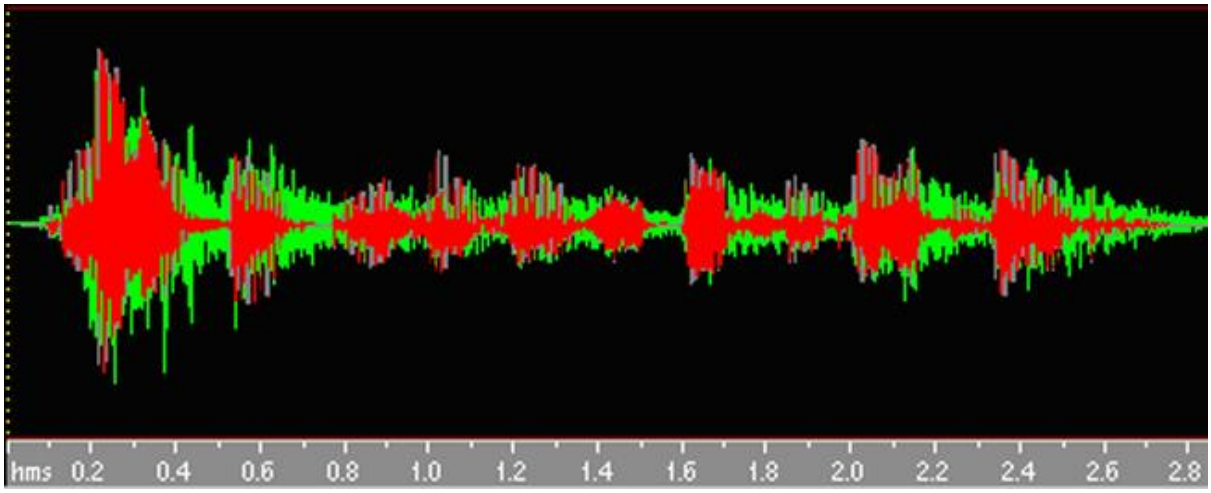


FIGURE 10 AUDIO SIGNAL WITH EXCELLENT EARLY REFLECTIONS AND REVERB[11]

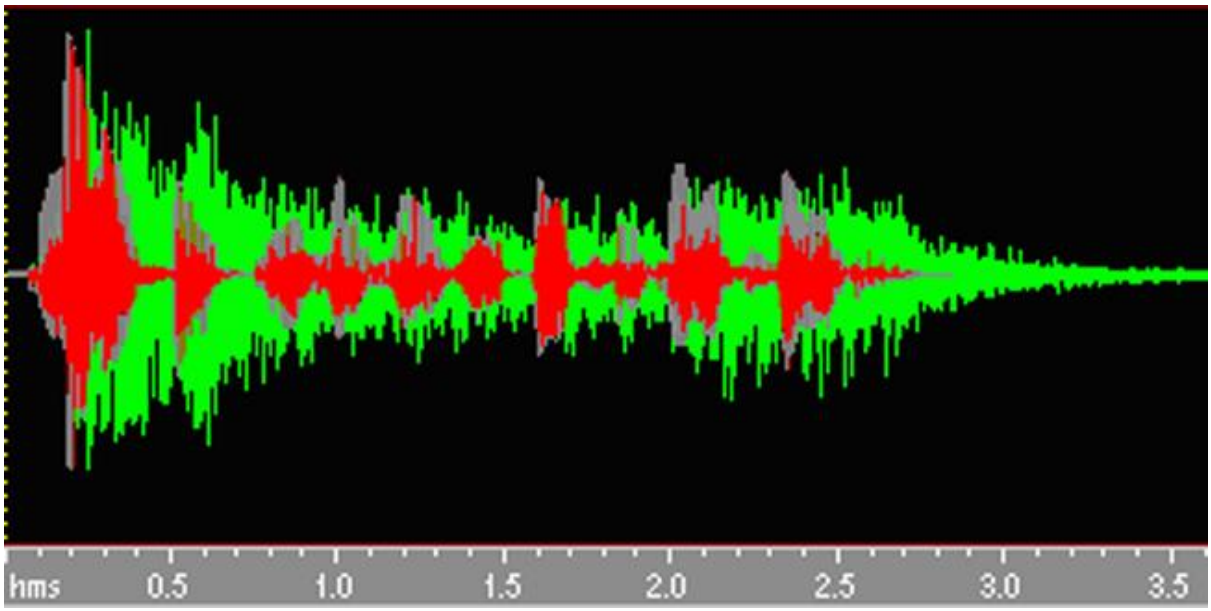


FIGURE 11 AUDIO SIGNAL WITH INFERIOR EARLY REFLECTIONS AND REVERB[11]

Should the walls be too reflective, the early reflections and reverb are very pronounced which can result in the original audio becoming very difficult to distinguish the direct sound and early reflections from the overly resonant late reflections. Another problem with this approach is the Hallelujah effect [5].

Week 5

Drafted 3D ray tracer design and concept and created an early high-level design in the form of a UML diagram.

Began work on Interim Report.

Week 6

Further work on 3D ray tracer design.

Pressure wave amplitude over distance follows an inverse square law.

Week 7

This week has been writing up the interim report. There has been a back and forth of several drafts and feedbacks and a final meeting with Dr Kingsley Sage on Wednesday to discuss any final changes to the interim report.

Interim Report Submitted.

References

[1] NVIDIA Developer. 2020. *Vrworks - Audio*. [online] Available at: <https://developer.nvidia.com/vrworks/vrworks-audio> [Accessed 12 October 2020].

[2] Bcs.org. n.d. *BCS Code Of Conduct*. [online] Available at: <https://www.bcs.org/membership/become-a-member/bcs-code-of-conduct/> [Accessed 9 November 2020].

[3] Xiang, N., 2020. Generalization of Sabine's reverberation theory. *The Journal of the Acoustical Society of America*, 148(3), pp.R5-R6.

[4] Inc., A., 2020. *Audiokinetic Blog*. [online] Blog.audiokinetic.com. Available at: <https://blog.audiokinetic.com/image-source-approach-to-dynamic-early-reflections/> [Accessed 9 November 2020].

[5] Quantec.com. 2020. *QUANTEC: Room Simulation*. [online] Available at: <http://www.quantec.com/index.php?id=room_simulation> [Accessed 12 October 2020].

[6] 2020. *Mixing Gerald Heyward | Ross Rothero-Bourge*. [image] Available at: <https://www.youtube.com/watch?v=PiXjBvPNCPw&ab_channel=VicFirth> [Accessed 12 October 2020].

[7] Hidetoshi, N., Yoshifumi,, C., Tsuyoshi, U. and Masanao, E., 2003. Frequency domain binaural model based on interaural phase and level differences. *J-STAGE*, [online] Available at:

[8] Juce.com. 2020. *JUCE | JUCE*. [online] Available at: <https://juce.com/> [Accessed 11 November 2020].

[9] GitHub. 2020. *Iplug2/Iplug2*. [online] Available at: <https://github.com/iPlug2/iPlug2> [Accessed 11 November 2020].

[10] Jvstwrapper.sourceforge.net. 2020. *Jvstwrapper - Java-Based Audio Plug-Ins*. [online] Available at: <http://jvstwrapper.sourceforge.net/> [Accessed 11 November 2020].

[11] Doyle, K., 2020. *Education_Reflections*. [online] Kevindoylemusic.com. Available at: <http://www.kevindoylemusic.com/education_reflections/> [Accessed 9 November 2020].

[12] Arons, B., 2020. A Review of The Cocktail Party Effect. *Journal of the American Voice I/O Society, 1992*, p.1.

[13] Hansen, P., 2020. FUNDAMENTALS OF ACOUSTICS. *https://www.who.int/*, [online] pp.5-6. Available at: <https://www.who.int/occupational_health/publications/noise1.pdf> [Accessed 6 November 2020].

[14] Harris, M., 2020. *An Even Easier Introduction To CUDA | NVIDIA Developer Blog*. [online] NVIDIA Developer Blog. Available at: <https://developer.nvidia.com/blog/even-easier-introduction-cuda/> [Accessed 11 November 2020].

[15] Gupta, P. and Perelygin, K., 2020. *CUDA Refresher: The CUDA Programming Model | NVIDIA Developer Blog*. [online] NVIDIA Developer Blog. Available at: <https://developer.nvidia.com/blog/cuda-refresher-cuda-programming-model/> [Accessed 11 November 2020].