MULTICHANNEL DATA USING UNITY

Simon Andersen

Aalborg University
Department of Architecture, Design and Media Technology
Sydhavn, Denmark
sand22@student.aau.com | sandersen98@live.dk

ABSTRACT

This project aims to create a simulation system in Unity, utilising the Steam Audio plugin, for generating multichannel spatial audio data. The primary objective is to assess the effectiveness of spatial audio processing software in strengthening speech-enhancing machine learning algorithms. High-quality audio recordings from diverse environments are essential for training algorithms to improve the clarity of noisy speech signals. However, obtaining a comprehensive database of such recordings is time-intensive. To address this challenge, the project proposes a simulation approach using software to generate these recordings, potentially streamlining the process. The simulation employs a virtual microphone in Unity to produce audio in various simulated acoustical spaces. The quality of the simulated audio renders will be evaluated by comparing them with real-life equivalents.

1. INTRODUCTION

Artificial Intelligence (AI) is a central concept in today's technology landscape, with new applications emerging regularly. In the audio domain, researchers and developers experimenting with AI to improve speech quality in noisy environments[1]. This improvement often involves the integration of machine learning(ML), particularly through Deep Learning(DL). DL, a subset of machine learning, utilises layered structures, known as *models*, to progressively extract complex features from raw data. In image processing, initial layers recognize basic elements like edges, while subsequent layers identify more advanced concepts such as numbers, letters, or facial features. DL models are trained with extensive data, making predictions, comparing them to actual results, and iteratively adjusting to enhance accuracy.

Therefore, ensuring high-quality data is crucial for effective model training[2]. The more flaws present in the data, the higher risk of flaws in the consequential model that is taught. The principle of "garbage in, garbage out" in computer science underscores that nonsensical input data leads to nonsensical output data in the resulting model.

Hence why, high-quality recordings are essential for effective deep learning models in the audio domain[2]. However, creating such recordings is time demanding, requiring diverse outcomes such as recordings in different environments and conditions. This raises the question: How can data generation for audio machine learning be optimized?

Spatial audio researchers and companies, including Treble[1], have advanced in processing and synthesizing realistic audio for applications like virtual reality (VR). Notably, spatial audio plugins such as Steam Audio[2] and Microsoft's Project Acoustics[3] are available for VR. These solutions aim to enhance the realism of VR environments, simulating effects like water droplets in a cave. However, does the underlying audio processing offer a suitable foundation for realistic audio generation applicable to audio-related machine learning?

This project involves creating a spatial audio data generation system and validating the data quality.

2. ANALYSIS

2.1. Single channel vs. multi channel

Office spaces have been enhanced acoustically to improve the quality of its communicative ability. The same goes for online communication platforms. However, they can at times be problematic, especially when compounded by issues like poor internet connectivity, excessive background noise, and subpar microphone quality. To address these challenges, companies such as Microsoft, Google, and Zoom, known for their communication platforms, are focusing on developing speech enhancement algorithms using deep learning. These platforms primarily handle single-channel signals from a range of devices, a factor that complicates the task of improving audio quality for deep learning models. Nonetheless, these models are proficient in differentiating between background noise and voices, enabling many of these platforms to effectively offer noise reduction features. Moreover, the signals are mostly being delivered in a single-channel format which is satisfactory for noise suppression. However, this is not the case for speech-to-speech situations.

2.2. Speech-to-speech

With a speech-to-speech signal the algorithm should aim to enhance specific speech in a signal, such as distinguishing between coworkers in close proximity attending different virtual meetings. Even with state-of-the-art models, the challenge of suppressing 'jammers' or unwanted speech persists. Different approaches, such as 'speaker embedding' to recognize and isolate the user's voice[3],

[1] treble.tech
[2] https://valvesoftware.github.io/steam-audio/
[3] https://learn.microsoft.com/en-us/gaming/acoustics/what-is-acoustics

and multichannel- or spatially aware neural networks[4], are being explored to address this issue. In which spatial information is utilised from the recordings. This becomes particularly relevant with the emergence of headsets like Jabra Evolve2 85[4] and Apple AirPods Max[5], equipped with multiple microphones for capturing additional spatial information in the recorded signal.

2.3. Spatial Audio Data Generation

Jabra's R&D team is developing a multichannel voice enhancement concept, with a functional model based on one of their multi-microphone headsets. However, these tests demand extensive hardware setups, and testing specific outcomes necessitates repeating the setup process. Given the availability of free spatial audio simulation software for VR platforms, could these plugins realistically substitute physical data generation with a virtual approach?

2.4. Plugin performance

While spatial audio plugins may not perfectly replicate realistic acoustic properties, validating their performance against real acoustical tests can determine their compatibility with DL algorithms. Combining simulated and real data in the algorithm could enhance its depth[5], provided the simulated acoustical principles align reasonably well with reality, potentially leading to improved performance in certain instances[5].

2.5. Drawbacks

The spatial audio plugins are equipped with a comprehensive array of adjustable parameters designed to fine-tine acoustical effects. These parameters are highly sensitive and interdependent, making it challenging to identify the ideal settings for accurately recreating an environment. This complexity, due to the numerous interacting elements, *could* render the process as time-consuming as establishing a physical counterpart.

2.6. Advantages

Nevertheless, the engines of these plugins and 3D development suites (Unity and Unreal) wield significant power and influence in the field. They facilitate the rapid creation of mock-ups and ideas, allowing for real-time examination and adjustment of configurations. Which is an advantageous feature for speech-processing algorithms. The scalability of surroundings within the 3D engine is also a favorable asset, making it easier to test and modify virtual replicas of environments. Additionally, extreme cases can be efficiently produced on a massive scale in the virtual realm, which may not be as readily available in real life.

3. PROBLEM STATEMENT

Does simulated acoustical data rendered via a 3D development suite using a spatial audio plugin perform well enough to be used as input for a voice enhancing algorithm?

[4]https://www.jabra.dk/business/office-headsets/jabra-evolve/jabra-evolve2-85

[5]https://www.apple.com/dk/airpods-max/

4. DESIGN

To create this system, it must take a non-spatial audio file as input and produce a spatial audio file as output. The processing algorithm of the plugin should also be adjustable for various acoustic environments through a user-friendly interface.

4.1. Plugin & Platform

Both 'Project Acoustic' and 'Steam Audio' use ray tracing as their engine for spatial audio simulation[6][7]. Since both company's selling points are sound propagation using low resources for VR audio[6][8] it's hard to pinpoint which one would be better for this project. They also both contain a black-box algorithm that cannot be addressed directly. Therefore, Steam Audio was chosen due to previous experience in the company.

Unity and Unreal engine are the available platforms for Steam Audio and due to previous experience in the company, the lightweight framework, and C# support, Unity was chosen as the 3D development platform.

4.2. SOFA Files

In order to make use of the spatial audio plugins a SOFA file is used. A SOFA file is a file format used for storing and exchanging spatially-oriented acoustic data. This enables the accurate reproduction of spatial audio scenes in applications. For testing purposes, Jabra has their own SOFA files made from their headsets. These should be used in the simulated renders to get them as close as the real test data.

4.3. Far & near field sources

Virtual recordings of far- & near field sources are required in pursuance of comparing the simulated renders to the real test data. Far field recordings are made from sources placed around a microphone arriving at different distances and angles. Near field recordings accounts for the voice recording of a user using the headset. This means that a source is placed around 17 cm from the microphone replicating the distance of a person's mouth.

With this information, a short use-case list was created as seen in section 4.4.

4.4. Use-cases

1. Select audio input
2. Select the duration of the generated data
3. Select the amount of sources in the render
4. Configure source audio parameters
5. Select room dimensions
6. Configure room acoustics parameters
7. Render far- & near-field sources
8. Select custom HRTF SOFA files.
9. Randomise position of the receiver and sound sources
10. Calculate elevation, azimuth and distance of the sound sources

4.5. System Requirements

Then a more comprehensive system requirement list was made of what the system should include to satisfy the render simulation.

1. **Environment Simulation:**
   - *Realistic Soundscapes:* The program should be able to simulate various environments with accurate audio rendering.
2. **Audio Generation:**
   - *Spatial Audio:* The program should support 3D positional audio, allowing sounds to come from specific directions in the virtual environment.
   - *Sound Sources:* Ability to create, manipulate, and position sound sources (e.g. voices) within the environment.
   - *Ambient Sounds:* Support for ambient audio layers or jammers (e.g. distant chatter) to enhance immersion and challenge the algorithm.
   - *Audio Effects:* Integration of audio effects like reverb, echo, and filtering for realistic audio experiences (e.g. the spatial audio plugin).
3. **User Interface:**
   - *GUI:* An intuitive graphical user interface for users to interact with the simulation settings.
   - *Sound Selection:* Ability to choose and configure different soundscapes, environments, and audio sources.
   - *Real-time preview:* Provide a real-time audio preview of the selected environment and soundscapes.
4. **Recording functionality:**
   - *Audio Capture:* The program should be able to record in-game audio during simulations.
   - *Timer:* A timer should keep track of the render.
   - *Recording Settings:* Allow users to configure audio recording settings such as sample rate, format (e.g., .wav, .mp3), and duration.
   - *Save and Export:* Capability to save recorded audio files for external analysis and testing.
   - *Randomisation:* Randomising parameters and position on each iteration of render.
5. **Performance:**
   - *Optimization:* Ensure efficient resource utilization to maintain a high frame rate and audio quality during simulations.
   - *Scalability:* The ability to handle complex soundscapes and large-scale environments without significant performance degradation.
   - *Save and Export:* Capability to save recorded audio files for external analysis and testing.
   - *Automation:* Automatise the render process as much as possible
6. **Exporting Recorded Audio:**
   - *Compatibility:* Ensure compatibility with various audio analysis tools and formats used in the vocal enhancement AI models.

4.6. Design pattern

In order to maintain the scalability of the code in Unity different coding approaches can be made. The MVVM (model - view - view model) pattern will be implemented to divide each area of functionality into its own class. As well as using the Singleton pattern for ensuring that only one instance of the manager class exists. In addition, when working with asynchronous actions, like timers, parts of the code will be waiting for each other. This is why the observer pattern would be very advantageous to be implemented in this case. Implying that every class subscribed to the observer will be notified whenever the asynchronous action is completed.

5. IMPLEMENTATION

5.1. System Design

The MVVM pattern was implemented by organizing each functionality into its dedicated class. The user-interacting view is represented by a single class linked to a Singleton manager class containing models for sources and persisted data attributes

The Singleton pattern was "simulated" using an empty Unity scene object as a manager class, with the singleton class as a script attached. This setup allows the class to have comprehensive knowledge of scene assets, enabling the manager class to keep track of the scene and collaborate with ViewModels. Adding and modifying rooms during the render process becomes feasible by attributing them to the manager class.

The observer pattern was successfully implemented through an interface in the view class. A 'notify' method in the manager class triggers the view update, effectively employing the observer pattern.

5.2. Acoustic simulation

Steam Audio enables spatial audio capabilities for simulating realistic acoustics that align with the 3D environment in a Unity scene. Using ray tracing, Steam Audio simulates sound reflections in the digital environment, creating a simulated reverberation in the active scene. Accurate reflection of the reverberation ray requires information about the angle and distance for rays scattered from the audio source.

This is implemented by applying the Steam Audio Geometry script to the physical objects constituting the scene. In this project, simple rectangular rooms were created in various sizes to emulate small and medium office spaces or group rooms. The script allows customization of audio absorption, with 100% absorption indicating a completely soundproof room. Additionally, audio transmission through the geometry material and scattering of rays can be customized. When the scene is complete, the geometry needs to be exported in a file which Steam Audio uses to recognize the edges of the scene. During runtime, the rays interact with this geometry file, accordingly composing the reverberation.

5.3. Rendering

Unity provides built-in audio processing tools, including the UnityEngine MonoBehaviour class, which features the OnAudioFilterRead() script. This script grants access to the audio processed through the Unity executable. During rendering, all audio data is read through the script and saved as a .wav file. The implementation involves using the C# System.IO class, where a FileStream writes audio bytes from Unity to a file, incorporating the necessary .wav header data for creating an audio file[9]. This process is achieved by adding a script with the OnAudioFilterRead() function to an object in the 3D scene. This object can then recognize and transmit the processed audio data to another script responsible for converting Unity Audio to a .wav file. To optimize performance, a flag is set to trigger the initiation of writing data to the converter script.

5.4. Timer

Implemented using Unity's coroutine function, a timer dictates when to conclude the render functionality. Set with a specified duration, WaitForSeconds() pauses for one second before resuming. The duration progressively decreases until reaching zero, signaling the end of the render.

5.5. GUI

To establish a continuous testable system, a graphical user interface (GUI) was created, incorporating all audio-related parameters from Unity and Steam Audio. The C# view class was connected to the canvas, enabling real-time parameter adjustments and direct audio feedback. This addition enhances the system's controllability and facilitates the adjustment of room acoustics and audio. Refer to figures 1 and 2 in the appendix for snippets of the GUI.

5.6. Data Persistence

A class was created for usability, saving selected parameters as JSON data. During runtime, the system reads these parameters from the persisted file, allowing users to resume their work. This simplifies system usage, particularly as Steam Audio cannot export the geometry scene during runtime.

5.7. Environment selection

A scene selection was implemented to increase the amount of environments to choose from. It was made with scalability in mind in such new rooms could be created and used for new acoustical environment. This was done by creating new scenes with physical attributes and adding the Steam Audio Geometry script on the walls of the rooms. These scenes were added to the Unity build manager for runtime access. A callback method was necessary to load new room parameters and synchronize the interface with the updated data.

5.8. Randomisation

To automate the rendering process and introduce variability, parameters were randomized. Source positions were randomized to obtain new angles and distances for renders, along with changing parameters for each new position. The workflow involved calculating new values for source volume within a 10-20 percent interval, adjusting Steam Audio's direct level mix from 90-100 percent, and reflection level mix from 10-30 percent. Due to the sensitivity of the near field source's reflection level mix, an additional randomization interval of 0.1-0.2 percent was specifically incorporated.

The randomisation of source and microphone position was achieved by creating a threshold where the position was allowed to be replaced inside the active scene. When the timer ended, a method was called to generate the new parameters and positions. Replacement occurred for both vertical and horizontal positions, excluding extreme cases in the top or bottom corners. Threshold values were set between 1.6-2 units vertically and always 3 units from the walls horizontally.

5.9. Automation

To streamline the render process automation, a comprehensive workflow logic was implemented. When a render commences, the timer initiates a countdown, resets the positions of the source and receiver, randomizes source parameters, and selects the first SOFA file for use. The system then plays the audio in the scene and begins writing bytes into a .wav file until the timer expires.

The system then manages its status by assessing the remaining SOFA files and repeats the process with the next in line. Once all SOFA file are processed, the system evaluates the far field renders' status based on how many the user input. This scenario repeats until all far field renders are completed.

Upon completing all SOFA files and the specified number of far field renders, the system proceeds to the near field. It plays the near field source, writes to a .wav file, and repeats until rendering is completed for every SOFA file. This marks the conclusion of the entire render process.

6. RESULTS

With the proof-of-concept implemented, the simulated acoustical test data became input for a voice-enhancing model. 18 render folders were created within a virtual room measuring 16x8 units in Unity, enclosed by a roof and floor. Each of these folders included ten far field recordings and one near field recording, with distinct acoustic settings for each individual folder. This variation meant some renders had a more reverberant scene than others. The simulated data was incorporated into the test framework, generating multiple clean vocal recordings, vocals obscured by jammers, and vocal enhancements for the noisy files.

In Figure 3 (appendix), the spectrogram displays the noisy vocal, while Figure 4 (appendix) shows the corresponding enhanced vocal, clearly reflecting successful noise suppression. When the audio is played, it is evident that the jammer is effectively suppressed to a degree.

The rendered results were compared against a set of similarly simulated renders and approximately 300 real recordings. This comparative test involved averaging results across 700 speech signals and training on three neural networks: NsNet2[10], CRUSE[11], and DEMUCS[12], as illustrated in Figure 5 and 6.

The values used for measuring speech quality in this test were

- **DNSMOS**[13] measures the speech quality with 3 features. The signal quality (sig), the background intrusiveness (bak) and the overall quality (ovrl). It lies in a range of 1-5 and a higher rating is regarded as being better.

- **WARP-Q**[14] measures prediction accuracy of the Short-time Fourier-transform and a low value is deemed as being better.
- **STOI**[15] measures the short time objective intelligibility from 0 to 1 where a higher value is better.
- **PESQ**[16] measures the perceptual evaluation of speech quality from 1-4.75 with the higher the better.

The system-simulated values are represented by "unity" in the table. Namely: *NsNet2_unity_IR*, *Cruse_unity_IR* and *Demucs_unity_IR*. The table indicates that the speech quality values from the simulated renders closely align with the measurements of the real recordings. However, they exhibit a slightly lower performance compared to the real ones.

7. CONCLUSION

This project aimed to determine whether simulated acoustical data, generated in a 3D development suite with a spatial audio plugin, can effectively serve as input for voice-enhancing algorithms.

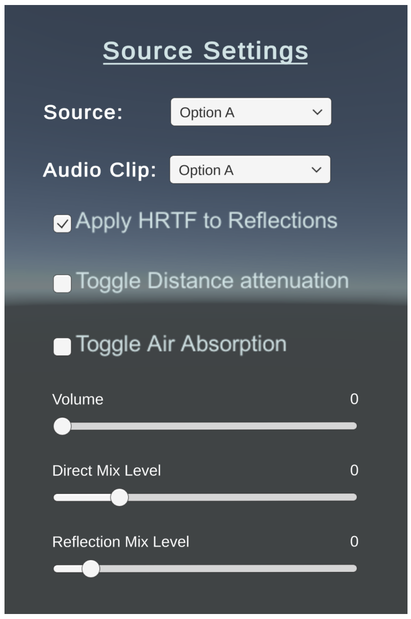
The results obtained from the jammer suppression test and voice quality measurements demonstrate that simulated acoustical data, rendered via a 3D development suite using a spatial audio plugin, does perform well enough to be utilised as input for voice-enhancing algorithms. Moreover, it exhibits only a slight degradation in voice quality compared to its real counterpart, with a noticeable decline in jammer suppression performance. It is noteworthy that the Unity-trained model utilised significantly less data compared to the other models, indicating substantial potential for improvement.

The system requirements status indicates that customization of room dimensions is unfinished. However, this can be achieved by creating a new virtual room, albeit with a slightly more time-consuming process. The same applies to GUI customization of the render duration.

8. ACKNOWLEDGMENTS

A heartfelt acknowledgment to Clément Laroche for his role as the lead supervisor and, notably, for adding the word ’funky’ to my vocabulary. Additionally, gratitude extends to Diego Caviedes Nozal, Karim Haddad, and Jesper Udesen for their assistance throughout the project. Thanks also to other student helpers for contributing to a collegial and energetic atmosphere.

9. APPENDIX


Source Settings

Source: Option A

Audio Clip: Option A

[✓] Apply HRTF to Reflections

[ ] Toggle Distance attenuation

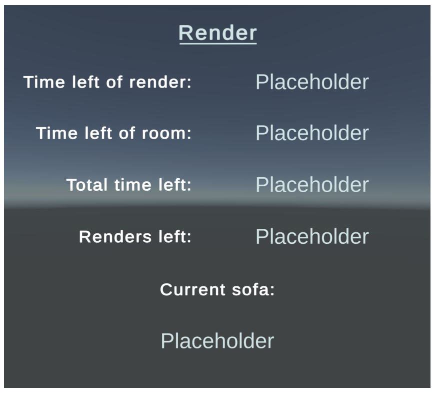
[ ] Toggle Air Absorption

Volume 0

Direct Mix Level 0

Reflection Mix Level 0

Figure 1: *Snippet of the source parameter configuration*


Render

Time left of render: Placeholder

Time left of room: Placeholder

Total time left: Placeholder

Renders left: Placeholder

Current sofa:

Placeholder

Figure 2: *Snippet of the render feedback interface*

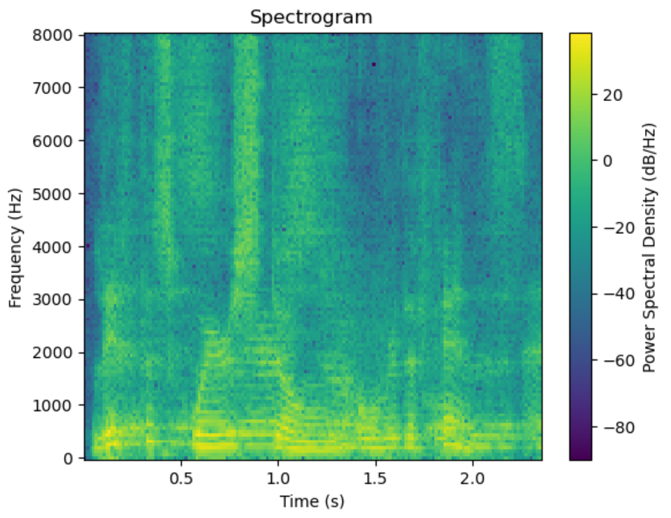


Figure 3: *Spectrogram of the noisy data*

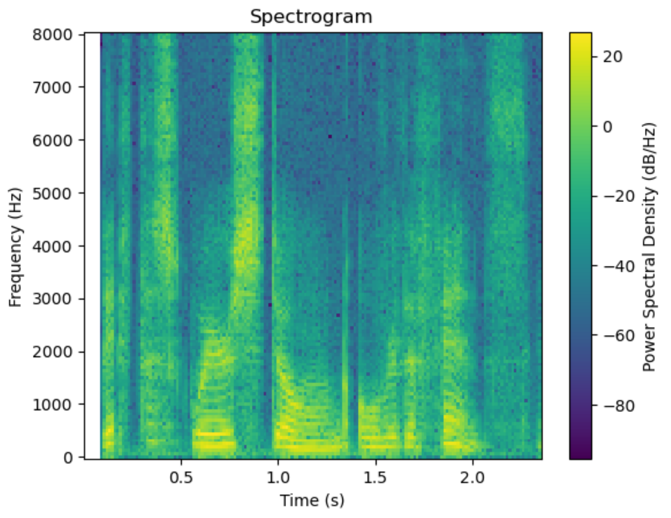


Figure 4: *Spectrogram of the enhanced data*

NN results	DNSMOS sig	DNSMOS bak	DNSMOS ovrl
NsNet2_unity_IR	2,62	3,95	2,64
NsNet2_flex_IR	2,68	3,93	2,68
NsNet2_trainIR	2,91	3,92	2,58
Cruse_unity_IR	2,61	3,84	2,33
Cruse_flex_IR	2,65	3,84	2,34
Cruse_train_IR	2,55	3,86	2,24
Demucs_unity_IR	3,26	4,05	2,92
Demucs_flex_IR	3,20	4,0	2,89
Demucs_train_IR	3,35	4,06	3,05
Noisy_train_IR	2,29	1,96	1,73
Noisy_test_IR	2,46	1,86	1,74

Figure 5: *Comparisons of DNSMOS measurements of speech quality*

NN results	WARPQ	STOI	PESQ
NsNet2_unity_IR	1,05	0,9	2,27
NsNet2_flex_IR	1,034	0,91	2,28
NsNet2_trainIR	1,06	0,89	2,13
Cruse_unity_IR	0,99	0,89	1,93
Cruse_flex_IR	0,99	0,89	1,95
Cruse_train_IR	1,04	0,85	1,8
Demucs_unity_IR	0,89	0,97	2,20
Demucs_flex_IR	0,98	0,91	2,04
Demucs_train_IR	0,76	0,97	2,69
Noisy_train_IR	1,21	0,79	1,12
Noisy_test_IR	1,04	0,76	1,123

Figure 6: *Comparisons WARPQ, STOI and PESQ measurements of speech quality*

10. REFERENCES

[1] Microsoft Editorial, "Dns challenge: Icassp 2023 deep noise suppression challenge - microsoft research," Available at https://www.microsoft.com/en-us/research/academic-program/deep-noise-suppression-challenge-icassp-2023/.

[2] B.D Klein and D.F Rossin, "Data quality in neural network models: effect of error rate and magnitude of error on predictive accuracy," *Omega*, vol. 27, no. 5, pp. 569–582, 1999.

[3] Mohamed Nabih Ali, Alessio Brutti, and Daniele Falavigna, "Direct enhancement of pre-trained speech embeddings for speech processing in noisy conditions," *Computer Speech & Language*, vol. 81, pp. 101501, 2023.

[4] Zhong-Qiu Wang, Jonathan Le Roux, and John R. Hershey, "Multi-channel deep clustering: Discriminative spectral and spatial embeddings for speaker-independent speech separation," in *2018 IEEE International Conference on Acoustics, Speech and Signal Processing (ICASSP)*, 2018, pp. 1–5.

[5] George Loizou, "Data simulation: unlocking innovation & empowering organizations," https://mostly.ai/blog/data-simulation, June 7, 2023.

[6] Microsoft Editorial, "What is project acoustics?," *Microsoft Game Dev Docs*, 02/03/2023.

[7] Valve Editorial, "Steam audio unity integration," Available at https://valvesoftware.github.io/steam-audio/doc/unity/index.html, accessed September 06, 2023.

[8] Valve Editorial, "Steam audio product page," Available at https://valvesoftware.github.io/steam-audio/, accessed September 06, 2023.

[9] Abolfazl Tanha, "Audio recorder in unity," Available at https://www.youtube.com/watch?v=tI0PuFlfMfI, April 7. 2023.

[10] Sebastian Braun and Ivan Tashev, "Data augmentation and loss normalization for deep noise suppression," in *International Conference on Speech and Computer*. Springer, 2020, pp. 79–86.

[11] Sebastian Braun, Hannes Gamper, Chandan Reddy, and Ivan Tashev, "Towards efficient models for real-time deep noise suppression," in *ICASSP 2021 - 2021 IEEE International Conference on Acoustics, Speech and Signal Processing (ICASSP)*, 06 2021, pp. 656–660.

[12] Alexandre Defossez, Gabriel Synnaeve, and Yossi Adi, "Real time speech enhancement in the waveform domain," *arXiv preprint arXiv:2006.12847*, 2020.

[13] Chandan KA Reddy, Vishak Gopal, and Ross Cutler, "Dnsmos p. 835: A non-intrusive perceptual objective speech quality metric to evaluate noise suppressors," in *ICASSP 2022-2022 IEEE International Conference on Acoustics, Speech and Signal Processing (ICASSP)*. IEEE, 2022, pp. 886–890.

[14] Andrew Hines, Jan Skoglund, Michael Chinen, and Wissam Jassim, "Warp-q: Quality prediction for generative neural speech codecs," *Accepted for presentation at IEEE ICASSP 2021.*, 2021.

[15] Cees H Taal, Richard C Hendriks, Richard Heusdens, and Jesper Jensen, "A short-time objective intelligibility measure for time-frequency weighted noisy speech," in *2010 IEEE international conference on acoustics, speech and signal processing*. IEEE, 2010, pp. 4214–4217.

[16] Antony W Rix, John G Beerends, Michael P Hollier, and Andries P Hekstra, "Perceptual evaluation of speech quality (pesq)-a new method for speech quality assessment of telephone networks and codecs," in *2001 IEEE international conference on acoustics, speech, and signal processing. Proceedings (Cat. No. 01CH37221)*. IEEE, 2001, vol. 2, pp. 749–752.

[17] Bahareh Tolooshams, Ritwik Giri, Andrew H. Song, Umut Isik, and Arvindh Krishnaswamy, "Channel-attention dense u-net for multichannel speech enhancement," in *ICASSP 2020 - 2020 IEEE International Conference on Acoustics, Speech and Signal Processing (ICASSP)*, 2020, pp. 836–840.

[18] Anna Watson and M. Angela Sasse, "The good, the bad, and the muffled: The impact of different degradations on internet speech," in *Proceedings of the Eighth ACM International Conference on Multimedia*, New York, NY, USA, 2000, MULTIMEDIA '00, p. 269–276, Association for Computing Machinery.