DOI: 10.61782/fa.2025.0243

FORUM ACUSTICUM
EURONOISE 2025

COMPLEX ROOM ACOUSTICS RENDERING WITH MULTI-ZONE AURALIZATION: APPLICATION TO TEATRO PRINCIPAL DE VALENCIA

Jesus Lopez-Ballester[1], **Jaume Segura-Garcia**[2*], **Maximo Cobos**[2]
Rosa Cibrián[3], **Salvador Cerdá**[4], **Alicia Giménez**[5]
[1] Department of Electronic Engineering, Universitat de Valencia, Spain
[2] Department of Computer Science, Universitat de Valencia, Spain
[3] Department of Physiology, Universitat de Valencia, Spain
[4] Department of Applied Mathematics, Universitat Politecnica de Valencia, Spain
[5] Department of Applied Physics, Universitat Politecnica de Valencia, Spain

ABSTRACT

Auralization in complex rooms is not always a straightforward task. The most common and simple approach involves taking an averaged impulse response for the entire room and applying binaural filters. For a more realistic approach, the room acoustic analysis of such halls requires the measurement or the simulation of impulse responses in different areas, to analyze the main room acoustic metrics. At this stage, impulse responses can be grouped to define specific zones of interest. In this project, we present a virtual acoustic demonstration of a theatre in Valencia, originally built in 1832 and last refurbished in 2012. For this theatre, we have developed an application specifically designed to represent audio modified with different Impulse Responses (IRs) within this enclosed space. Using Text-to-Speech techniques based on Artificial Intelligence, we have synthesized human speech, allowing us to convert written text into spoken audio in any language. Each IR has been convolved with this AI-generated anechoic audio. In this project, we use the Teatro Principal de València as a test space, where reverberation zones shape the acoustic environment and the custom-synthesized spoken text is integrated. The goal is to create a dynamic auralization of the poem that inaugurated the theatre, moving through the seating area to enhance the immersive experience.

Keywords: *Auralization, Theatre, Spatialization, Scene recreation*

**Corresponding author*: *jaume.segura@uv.es.*

1. INTRODUCTION

In the last decade, immersive audio experiences have gained significant interest across different domains including virtual reality (VR), architecture, heritage preservation, and acoustic design. Central to these experiences is the accurate simulation of acoustic environments—a process that enables users not only to see but also to hear and feel the spaces they explore. As virtual environments grow more sophisticated, the demand for realistic, spatially-aware sound rendering systems continues to rise.

Auralization [1], the process of rendering audible simulations of a sound field within a virtual environment, has become a cornerstone technology in the field of virtual acoustics. By combining acoustic modeling, impulse response measurements, and binaural rendering techniques, auralization bridges the gap between visual immersion and auditory realism. This capability is especially critical in complex scenarios such as concert hall design, historical reconstructions, and performance space evaluation, where the auditory experience plays a pivotal role.

In traditional auralization methods, it is typically assumed a single source and listener position, limiting their effectiveness in dynamic or multi-user applications [2]. Addressing this limitation, multi-zone auralization introduces a framework where different acoustic zones—each characterized by distinct room impulse responses (RIRs)—can be rendered simultaneously within a shared virtual space. This approach enhances the spatial coherence and realism of interactive audio scenes, allowing users to navigate through and experience acoustical changes within complex environments [3, 4].

This paper presents a multi-zone auralization system based on a use case such as the Teatro Principal de Valencia, a historic theater in Spain. By integrating impulse responses measured in situ with synthetic anechoic audio sources generated via text-to-speech neural networks, we render a layered auditory landscape mapped onto a 3D model of the venue. This method not only recreates the unique acoustic signatures of different areas within the theater but also enables the presentation of performances—spoken or musical—without the need for live recordings.

The proposed system uses Unity and FMOD Studio [5] for spatial rendering and interactivity, highlighting an accessible pipeline for combining real-world acoustic data with modern game engine technologies. This work contributes to the growing body of research in acoustic virtual reality by demonstrating a practical and scalable solution for immersive sound simulation in complex architectural spaces.

2. DESIGN AND IMPLEMENTATION

As described in the previous section, in our design we will perform text-to-speech voice synthesis [6] anechoically. Then we will perform an auralization by zones that will be finally taken to a 3D virtual environment [5]. The three parts of our design will be: text-to-speech audio generation [7, 8], auralization and interactive 3D playback. The different solutions adopted in the final design for each of these stages are detailed below.

2.1 Graphics engine

For the development of the zone rendering system, it is necessary to create a 2D or 3D environment in which to place these zones and then configure them as needed. In this case, working with impulsive responses, we are looking for an application rich in properties to work in 3D environments. Thanks to the strong growth of the video game industry, a large number of graphics engines have emerged that allow large companies and individuals to develop their video games according to their needs. Large companies often use their own graphics engines, such as Frostbite, Rockstar Advanced Game Engine... Nowadays, Unity and Unreal Engine are the most used outside the big companies with their own engine, their level of creativity, diversity and technology, make them the first and second choice when developing an application. For this project, both meet the expectations, however, we will have to select which one is more convenient when implementing our components.

- Unity is a multiplatform video game engine [9]. It allows the creation of 2D and 3D games and applications in real time, based on an API programmed in the C# programming language. In addition, it includes a series of tools that facilitate the creation of interactive content, such as drag-and-drop mechanics or its own store with thousands of high-quality products, most of them free. This tool is used worldwide by different development studios. However, its main intention is to offer independent developers a place to learn and develop their projects.
- Unreal Engine [10] is a video game engine developed by the company Epic Games presented as a first-person shooter, over the years it has been adopted by all kinds of video games of three-dimensional genre. Programmed in C++ and flexible in platform portability, it integrates a large number of features, such as pipeline integration, map development, animations, simulations and effects, material rendering, light rendering, developer tools, etc. Unreal Engine 4, released in 2014 as a paid platform, has been available for free download since 2015.

For this project, both tools far exceeded the requirements. However, we opted for Unity for its user-friendly and intuitive interface, along with the ease of integration with FMOD Studio for audio processing.

2.2 Audio engine

Both Unity and Unreal Engine incorporate their own audio engines, although with limited features for very specific uses. If a higher audio quality is required, it is necessary to resort to third-party libraries or engines to process the sound. In the case presented, different ones have been

evaluated, including Unity's native plugin, Audio SDK, as well as FMOD Studio or Wwise.

- Unity Audio Native Plugin SDK [9]. Composed of 2 parts, the first requires the DSP (Digital Signal Processing) plugin to be implemented in Windows as a .dll in C or C++. This means that it must be recompiled specifically for each platform on which you want to run the application to optimize its performance. The second part consists of creating a user interface in C# to control the attributes of the previously implemented plugin. It incorporates an extension called Audio Spatializer that allows to control the way the sound is emitted from the source to the scene space.

- FMOD Studio [11]. It is a sound engine and authoring tool for applications and video games developed by Fireflight Technologies, which allows the loading, playback and processing of audio files in different formats and platforms. It stands out for its simplicity, power and easy integration with tools such as Unity or Unreal Engine, in addition to incorporating a simple and intuitive user interface. It facilitates the tasks of working with large projects and the possibility of obtaining instant feedback thanks to the fact that it allows listening and modifying the processed audio in real time.

- Wwise [12]. It is a software developed by Audiokinetic created mainly for sound engineers who want to give a professional look to their applications, such as video games, AR and VR applications or simulations, incorporating spatial audio and virtual acoustics in its latest updates. It is integrable with graphics engines such as Unity or Unreal Engine and, therefore, cross-platform. In addition, it includes a wide variety of plug-ins designed to meet your specific needs.

In our case, for the selection of the audio engine, FMOD Studio was one of the best options in terms of simplicity and performance. Very good options such as Wwise were also evaluated, but they consumed more learning time as they were less intuitive to achieve similar results in our application. On the other hand, both CSound and Unity's native SDK plugin were discarded since we had the alternative of FMOD Studio, an easy-to-use and highly compatible tool with Unity.

2.3 Voice synthesis

TTS (text-to-speech) systems have an enormous range of uses. The first applications created that included this technology served as reading systems for the visually impaired. Early systems were very mechanical sounding, but were very well received by this community because it was not always possible to read braille or have a real person reading aloud. Today, systems incorporating TTS technology improve and facilitate machine-person interaction for this group. The main facet, and the one to which most importance is attached, is the quality of the system. Speech synthesis is composed of multiple factors that produce a more or less acceptable result, factors such as segmenting, organizing, and decoding text, and converting that text into a voice that must include phonetics, phonology, and pronunciation.

In the TTS, the input is writing and the output is voice. There are several models that treat information differently. In practice, systems are two or more combinations of the different models, such as: common form model, signal-to-signal, pipelined, text-as-language, grapheme-phoneme model, full linguistic analysis, or full prosody generation.

For the requirements of the project, we need a simple TTS system, compatible with Unity, using a C# or Python library that implements a speech synthesis system. This system must also be able to run at runtime and save the incoming string in an.mp3 or.wav file format. Although the library 'eSpeak NG' [7] seemed to be a strong candidate, there were problems in the implementation part and in the final quality of the synthesized voice; we finally opted for the 'pyttsx3' Python library or module [13], which met all necessary functional and quality requirements.

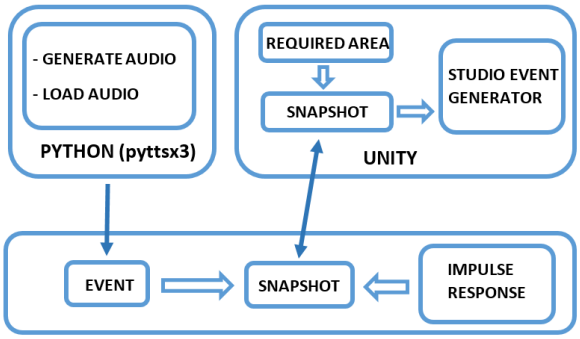


Figure 1. Design scheme

As shown in Figure 1, the final application is divided

into three parts, the first one corresponds to the Python part that allows us to generate or select the audio, which will later be loaded with code and played in the Unity scene as 'Programmer Sound'. Each snapshot is manually created by FMOD and then assigned to each rendered zone. Each snapshot will include a convolution reverb configuration with its coupled impulse response. They are assigned to a single event and are triggered by the source code in Unity. Each event is assigned a Programmer Sound that receives the audio through a script called from its callback function [11]. When launching the Unity application, all FMOD banks are loaded, with the snapshots and events previously compiled in FMOD Studio, thanks to the integration of FMOD in Unity. After having all the resources ready, just place them in the scene and activate and deactivate the components as needed.

3. SYSTEM IMPLEMENTATION AND TESTING

3.1 Sound rendering in FMOD

In this implementation, the user does not directly interact with the FMOD Studio application. All audio processing is performed beforehand for later compilation and data handling. Here, we will use FMOD to modify and prepare the audio that will be played within a Unity scene (shown in the lower part of Figure 1).

The process starts with the creation of a 3D event. In this case, only one event is needed to play a single audio clip. Different types of instruments can be assigned to an event—ranging from those that play a single sound, to those that can randomly choose from multiple sounds, as well as programmer instruments and snapshot instruments.

At first glance, a sound source that generates a single sound and can be processed with effects might seem sufficient. However, this type of sound source must be loaded with a pre-imported audio file within the FMOD Studio project. That audio file must then be compiled and stored in a sound bank, which Unity will later access. In our application, the audio file needs to be generated at runtime, making the manual preparation process unfeasible. For this reason, a programmer instrument is used, as it allows audio playback with files generated during runtime. At this stage, convolution reverb is applied as an audio effect. Here, this convolution is done with the averaged impulse responses obtained in each zone from the simulation with ODEON software (Figure 2 shows the list of returns and the configuration of one of them with the corresponding IR). The goal is to play an event with a sound effect, such as the convolution with the impulse response, that can change dynamically based on instructions from Unity. To achieve this, the program is prepared with different "states" as reverberation zones.

Figure 2. FMOD Studio lits of "returns" (up) and configuration of a "return" (down).

In FMOD Studio's Mixer tab, under the routing section, multiple signal returns have been created, each one with the corresponding zone IR. Figure 3 shows in different colors, the different reverberation zones associated to the different IRs considered. Within the "Source" group that contains the event, send buses are added to duplicate the signal and route it to these Returns, which ultimately feed into the main output bus. Each Return has a unique convolution reverb effect assigned, each associated with a different impulse response. These impulse responses are imported into the project as audio assets. This is where snapshots come into play.

Since all Returns are routed through the master bus, they would all play simultaneously by default. Snapshots allow the creation of saved states for the mixer configuration. In this case, snapshots are used to disable all but one Return at a time, enabling only the one desired. As a result, a unique snapshot is required for each Return signal. This process will be used in different zones by applying an Attack-Relay process in the change of zone.

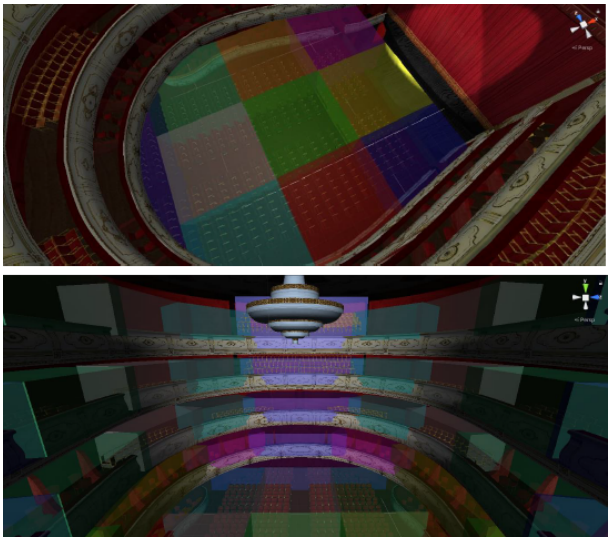


Figure 3. Reverberation zones within the Theatre.

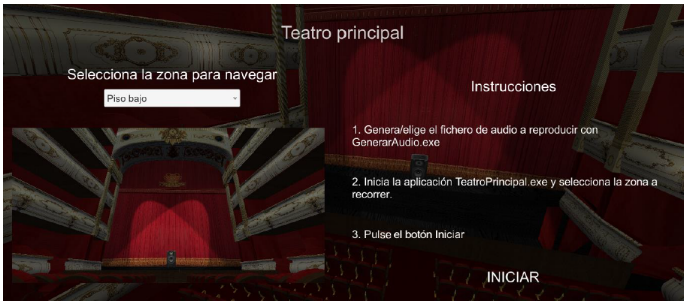


Figure 4. Graphical User Interface for the location selection.

3.2 Scene rendering and multi-zone rendering in Unity

Unity represents the final stage of the entire development process—everything discussed previously converges here (as shown in the upper-right part in Figure 1). This section will follow the order in which the user encounters components within the application.

Upon launching the application, the user first encounters the **main menu** (shown in Figure 4), which is implemented as a **Canvas GameObject** containing text, dropdowns, images, and buttons. This menu has an attached script called **MainMenu.cs**, which manages all menu-related functionality. It retrieves the selected value from the dropdown and displays a corresponding image of the exploration area using a 'switch' statement. It includes a method called 'startApp()', triggered when the "Start" button is pressed. This method hides the main menu and calls the 'IniciarAplicacion()' function from the **GameManager** object, passing the selected dropdown value to determine the player's starting position in the scene. The **GameManager** object, as its name implies, oversees game administration. It contains a script responsible for initializing all scene components—activating and positioning the player and initializing the sound source if it hasn't been initialized yet.

The **sound source** is represented by a GameObject resembling a speaker, downloaded from the Unity Asset Store as part of the "HQ Acoustic System by Next Level 3D" (licensed under the Standard Unity Asset Store EULA). This object includes two main scripts, StudioEventEmitter and ProgrammerSounds. The **StudioEventEmitter** is a part of the FMOD integration package. This script handles the emission of FMOD events from code and manages playback parameters and metadata. A custom method, 'ChangeSnapshot()', was added to this class, allowing snapshots to be updated dynamically based on player location. This method stops the current snapshot before applying a new one. The **ProgrammerSounds** is a modified version of an example script from FMOD's official documentation. It uses a callback function to retrieve the path to an audio file stored locally, which will be played as an event.

So far, we have seen the construction and function of the main menu and sound source, as well as how the player and source are initialized by the GameManager. The **player character** consists of a stretched cylinder with a camera attached at the top. The parent GameObject, named "Player", handles typical first-person movement. It includes two key components, the **Character Controller** (Unity native) that defines movement properties such as climbable slope angle, stair height, and collision bounds, and the **Custom movement script** which assigns values for speed, gravity, and jump force.

The same project includes camera setup. However, here the Unity's default **Audio Listener** needs to be replaced with the **FMOD Studio Listener** to correctly process FMOD audio events.

The application takes place in a virtual model of the **Teatre Principal de Valencia**, which includes all five levels of the real building (show in Figure 5). The **orchestra section** contains 21 rows of red velvet-upholstered seats with detailed metallic and wooden elements. A stage, equipped with a speaker, and a wooden orchestra pit are

also present, alongside detailed curtains. Each box seat area mirrors the real structure in both interior and exterior details—gold textures on the fronts and red velvet inside, complete with chairs and dividers. The ceiling features a grand chandelier with an ornate metallic design.

Figure 5. Navigation inside the Teatre Principal de Valencia.

Lighting is carefully set up to mimic the theater's ambiance: two directional lights for general illumination, two spotlights on the stage, two simulating the chandelier, and additional lights on the exterior of the balconies. The **rendered zones** are used to manage audio transitions. Each zone is a cube with a **Box Collider** and an **ActivaSnapshot.cs** script. The collider is set as a trigger, so it activates behavior rather than blocking movement.

Each zone receives parameters specifying which snapshot to activate in the FMOD list of returns (in Figure2 as a pair of row and seat) and the associated anechoic sound source. These zones trigger snapshot changes using Unity's native functions like 'OnTriggerStay()', 'OnTriggerEnter()', and 'OnTriggerExit()'. The theater is filled with many of these zones, each linked to a unique snapshot. Each snapshot is distributed based on the **properties of its impulse response**, generally categorized by row and seat number.

3.3 User tests

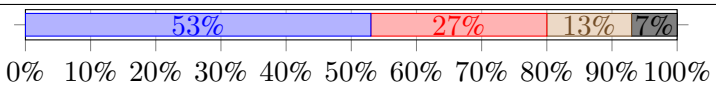
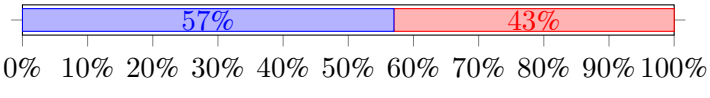
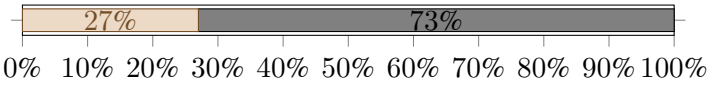
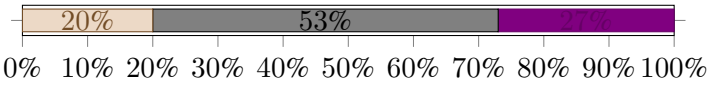
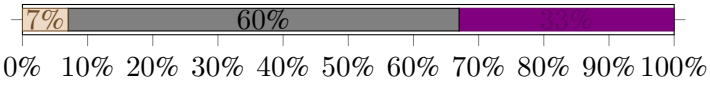
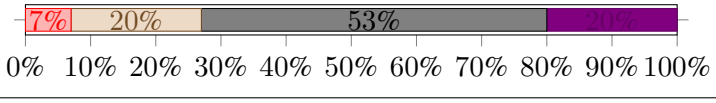
User testing involved presenting the application to a small group of participants, who provided feedback from their perspective. This external insight offered a broader view of the application's functionality. Each participant received the application as a compressed file containing two executables, without any instructions. Alongside the application, a **Google Form survey** was provided to collect feedback. The survey included aspects, such as:

- **Age group**, categorized into: Teen (under 21), Young adult (21–35), Adult (36–59), and Senior (60+)
- **Audio playback method**: Headphones and Loudspeaker
- **Sound realism**: Very unrealistic, Slightly lacking, Realistic and Very realistic
- **3D model quality**: Very poor to Excellent (5-point Likert scale)
- **Lighting quality**: Very poor to Excellent (5-point Likert scale)
- **Performance**: Very low frame rate to No issues (5-point Likert scale)
- **Problem report section**
- **Additional comments**

All responses were compiled into Table 1 averaging the scores in Likert categories, rated on a scale of 1 to 5 (1 = worst, 5 = best). This scale assigns numerical values to responses and analyzes the average and response trends to derive insights about user perception. These categories obtained four response options and were scaled with a minimum value of 2 (in the case of performance).

Some participants provided qualitative feedback, while others reported some issues. These issues were categorized into four main areas: user interface, controls, scene design (i.e. two users were unsure how to exit the application; one user had issues with movement controls), and general application functionality. Some user reported some comment about the model (i.e. "The character was too tall" or "The place felt a bit overwhelming, almost claustrophobic").

Table 1. Table summarizing the results of the survey.

Question	Response	Average
Age group		NA
Audio playback method		NA
Sound realism		4.73
3D Model Quality		4.06
Lighting Quality		4.27
Performance		3.87

These remarks suggest that some visual aspects of the application significantly influenced user perception. As users receive much of their information visually, visual immersion plays a crucial role.

4. CONCLUSIONS AND FUTURE WORK

In conclusion, several insights can be drawn from the completion of this project. Here it is shown a full Virtual reality application combining multi-zone acoustic rendering, which is a highly comprehensive field with wide-ranging applications across various modern technologies. A clear example is the video game industry, which increasingly focuses on creating immersive environments, particularly in 3D games, where virtual acoustics and 3D sound positioning play a significant role.

This application combines virtual acoustics with artificial intelligence (for anechoic speech generation from text), although in a basic form. Major companies such as Google and Amazon are actively investing in and supporting the development of these technologies, recognizing their potential as part of an emerging market with a promising future.

Regarding the user evaluation of the application, it was perceived as a good application in terms of quality (model and lighting), but the performance was not so good because the model was a little bit heavy and the client app had high hardware requirements (at least CPU i7 with 16GB RAM or GPU). Each of the Likert evaluated elements contributed multiplicatively to the user experience — if one of them was undervalued, the overall experience was significantly affected. Additionally, user age appeared to influence expectations. Younger users tended to be more critical of graphics and lighting, likely due to greater exposure to high-quality digital experiences. Older users were less demanding in these areas. Audio perception, however, was more influenced by the output device (speakers vs. headphones) than by user age. Headphones typically offer superior spatial audio localization due to their direct ear placement, outperforming speakers in 3D audio rendering. Despite this, most users rated the audio quality positively, regardless of the device used.

A possible direction for future development involves expanding the number of buildings and heritage sites in the city of Valencia. The primary goal would be to create a virtual network of tourist attractions, allowing users to explore these locations remotely. Theatres and concert halls would be particularly well-suited for such projects, as they could use virtual demos to attract and inform prospective visitors. Museums and other cultural institutions could

also offer virtual previews of their collections.

With further investment and development, a multiplayer feature could be implemented, enabling virtual performances of theatrical plays or concerts for remote audiences. In today's context, projects of this nature are increasingly relevant, and such cultural and tourist sites would benefit greatly from offering immersive digital experiences.

Another potential enhancement involves developing voice synthesis, particularly by incorporating emotional speech synthesis. Through specific techniques, it would be possible to imbue synthetic voices with emotional expressiveness by controlling parameters such as pitch, intensity, articulation, and automating these through ASR or ADSR (Attack, Decay, Sustain, Release) envelopes. This improvement would significantly enhance the application's quality, especially in the context of virtual theatre performances.

Further improvements could focus on upgrading the graphical component, including creating more detailed 3D models and possibly migrating the application to a more powerful engine such as Unreal Engine to achieve higher visual fidelity. Additionally, switching the audio engine —for instance, replacing FMOD with Wwise —could provide new functionalities and more advanced audio capabilities.

5. ACKNOWLEDGMENTS

This research was funded by the Spanish Ministry of Science and Innovation/Spanish Research Agency (MCIN/AEI) within the project Agriculture 6.0 with reference TED2021-131040B-C33 and the project ECO4RUPA with reference PID2021-126823OB-I00, funded by MCIN/AEI/ 10.13039/501100011033 and by the European Union "NextGenerationEU"/PRTR. Also, the authors would like to thank the Generalitat Valenciana for the grant CIBEST/2023/101 and the grant CIAEST/2022/91.

6. REFERENCES

[1] M. Vorlander, *Auralization: Fundamentals of Acoustics, Modelling, Simulation, Algorithms and Acoustic Virtual Reality*. RWTHedition, Springer Berlin, Heidelberg, 2014.

[2] T. Lokki, "How many point sources is needed to represent strings in auralization," 2017.

[3] J. Kang, F. Aletta, T. T. Gjestland, L. A. Brown, D. Botteldooren, B. Schulte-Fortkamp, P. Lercher, I. van Kamp, K. Genuit, A. Fiebig, J. L. Bento Coelho, L. Maffei, and L. Lavia, "Ten questions on the soundscapes of the built environment," *Building and Environment*, vol. 108, pp. 284–294, 2016.

[4] S. Cecchi, A. Carini, and S. Spors, "Room Response Equalization—A Review," *Applied Sciences*, vol. 8, no. 1, 2018.

[5] Fireflight Technologies, "FMOD," 2021. https://www.fmod.com/ (Visited 02/04/2025).

[6] P. Taylor, *Text-to-Speech Synthesis*. Cambridge University Press, 2009.

[7] M. Delille, R. Dunn, V.Aldisvi, J.Wilk, and A.V.Hon, "ESpeak-ng." https://github.com/espeak-ng/espeak-ng/blob/master/README.md (Visited 02/04/2025).

[8] A. van den Oord, S. Dieleman, H. Zen, K. Simonyan, O. Vinyals, A. Graves, N. Kalchbrenner, A. Senior, and K. Kavukcuoglu, "WaveNet: A Generative Model for Raw Audio," *arXiv:1609.03499 [cs.SD]*, 2016.

[9] Unity Technologies, "Unity Environment." https://unity.com/es (Visited 01/04/2025).

[10] Unreal Engine Technologies, "Unreal Engine Framework." https://www.unrealengine.com/es-ES (Visited 01/04/2025).

[11] Fireflight Technologies, "Scripting Examples. Programmer Sounds." https://www.fmod.com/resources/documentation-unity?version=2.02&page=examples-programmer-sounds.html (Visited 02/04/2025).

[12] A. Inc., "Wwise documentation," 2024. https://www.audiokinetic.com/en/library/edge/?source=Help&id=welcome_to_wwise (Visited 01/04/2025).

[13] N. M. Bhat, "pyttsx3 - Text-to-speech x-platform," 2021. https://pyttsx3.readthedocs.io/en/latest/ (Visited 02/04/2025).