

ENACT: Engine-Native Real-Time A-to-B AmbiX Conversion for Interactive XR

Mattia Mura
Università di Cagliari
Cagliari, Italy
m.mura135@studenti.unica.it

Gianluca Fadda
Università di Cagliari
Cagliari, Italy
gianluca.fadda@unica.it

Maurizio Murroni
Università di Cagliari
Cagliari, Italy
maurizio.murroni@unica.it

Nicuşor Amarie
Transilvania University of Braşov
Braşov, Romania
nicusor.amarie@unitbv.ro

Vlad Popescu
Transilvania University of Braşov
Braşov, Romania
vlad.popescu@unitbv.ro

Abstract—Live Ambisonics workflows can perform A-to-B conversion in digital audio workstations (DAWs), dedicated software, or hardware, also in real time. In these cases, however, the conversion is usually performed outside the extended reality (XR) engine audio graph. This paper investigates an alternative approach in which tetrahedral A-format signals are converted to AmbiX B-format directly inside the XR engine’s real-time audio callback. We present ENACT (Engine-Native AmbiX Capture Tool), a Unity prototype implementing first-order Ambisonics (FOA) A-to-B conversion, polarity correction, Ambisonic Channel Number (ACN) ordering, and Schmidt semi-normalisation (SN3D) inside the engine audio thread. Timing behaviour was compared with a DAW-hosted reference chain using a common impulse stimulus and session-level statistical analysis. In the tested configuration, the ENACT path showed significantly lower trigger-to-first-click detection timing than the external reference path. Tests performed inside the reference chain did not show a comparable timing change when the A-to-B processing was enabled. This suggests that most of the measured difference is related to the complete execution path rather than to the A-to-B conversion alone. The current prototype does not yet include microphone-specific correction filters or empirical calibration of the physical microphone array. Therefore, the present evaluation concerns timing behaviour and does not establish the complete spatial or spectral accuracy of the captured soundfield, and rendering of the live output through an Ambisonic decoder is not yet demonstrated. The strongest supported result is a working conversion prototype in the engine callback, with an initial comparison of complete execution paths. ENACT is intended for interactive XR applications where the captured Ambisonic soundfield needs to be directly available inside the engine audio graph.

Index Terms—Ambisonics, A-to-B conversion, tetrahedral microphone array, engine-native audio, real-time spatial audio, XR audio, First-Order Ambisonics, Internet of Sounds, musical XR

I. INTRODUCTION

The Internet of Sounds (IoS) describes the ensemble of Sound Things, i.e., devices that sense, acquire, process, and exchange sound-related information, together with the network infrastructures, protocols, and representations that connect them [1]. It merges the Internet of Musical Things, concerned with networked devices for the production and reception of

musical content [2], with the Internet of Audio Things. Music in extended reality (XR), or musical XR [3], is one of the most demanding application areas of this paradigm. Networked music performance requires end-to-end delays low enough to preserve musical interaction [4], [5], and audience studies on remote music events report a lower sense of sensory and spatial richness than physical attendance, identifying it as an important requirement for future platforms [6], [7].

Capturing the soundfield of the venue is a direct way to convey this spatial richness. In IoS terms, a tetrahedral first-order Ambisonics (FOA) microphone that streams its capsule signals towards an XR application is a Sound Thing, and the ambiX B-format convention [8] is a shared representation that downstream renderers can consume. For interactive and musical XR, the captured field must become usable at the rendering endpoint with as few additional hosts, routing stages, and clock domains as possible, because each of them adds delay and complexity to the distributed system. Many interactive XR applications need this live soundfield inside the engine’s own audio graph, for example to render it relative to a tracked listener, to mix it with synthetic in-engine sources, or to combine it with scene processing controlled by the application. The raw A-format signals must first be converted to B-format, and in current workflows this conversion takes place outside the engine (Section II).

This paper presents the Engine-Native AmbiX Capture Tool (ENACT), a general-purpose A-to-B converter that runs inside the XR engine. Professional performance production and distribution typically deliver audio, often object-based, to venue or broadcast production endpoints, where a sound engineer produces the front-of-house or broadcast mix [9]–[11]; ENACT is not proposed as a replacement for those workflows.

This paper investigates whether this conversion can be implemented as a real-time process inside the engine while keeping the conversion logic separate from the engine control code. ENACT is implemented as a Unity component that performs tetrahedral A-to-B conversion directly inside the

engine’s real-time audio callback. It targets scenarios in which the XR engine itself is the audio rendering endpoint, such as a XR application on the listener side or an interactive installation.

The contributions of this work are three. First, we implement first-order tetrahedral A-to-B conversion directly inside Unity’s real-time audio callback. Second, the conversion mathematics are kept in an engine-independent core, separated from the Unity wrapper, so that they can be tested without relying on the rendering loop. Third, we report a completed timing and jitter comparison between the ENACT execution chain and a DAW-hosted reference chain, using a common impulse stimulus and session-level statistical analysis.

The scope of the present evaluation is limited. The timing experiment characterises complete execution chains, not the computational latency of the A-to-B matrix alone (Section IV-D). The physical array has not yet been calibrated, and microphone-specific correction filters are not yet implemented (Section III-D), so the spatial and spectral accuracy of the captured soundfield is not evaluated here.

ENACT is one component of a broader research effort on networked spatial audio for hybrid XR. Fig. 2 shows how it integrates with the engine audio graph. ENACT’s A-format input can come from a local array connected to the engine host, as in the current prototype, or from remote arrays at other capture points whose signals reach the engine over the network (distributed ambience acquisition). Hardware capture, network transport, and downstream rendering are separate parts of this wider system and are not evaluated in the present paper.

The rest of the paper is organised as follows. Section II reviews related work on A-to-B conversion tools, spatial audio in XR engines, and networked immersive audio. Section III reviews the mathematical background of tetrahedral A-to-B conversion and the AmbiX standard. Section IV describes the ENACT architecture and the methodology used to evaluate its timing behaviour. Section V reports the timing and jitter results. Section VI discusses ENACT with respect to existing XR and immersive audio workflows. Section VII presents the current limitations and future developments.

II. RELATED WORK

A. A-to-B Conversion Tools

The conversion of tetrahedral A-format signals to B-format dates back to Gerzon’s formulation of periphony [12] and is part of current Ambisonics recording and production practice [13], [14]. Depending on the application, it is carried out in post-production, in a real-time chain hosted in a digital audio workstation (DAW), in dedicated software, or in the manufacturers’ own hardware and software. Practical converters combine the conversion matrix with correction filters for the specific microphone [15], [16] (Section III-D), and are thus usually distributed for a given microphone, as dearVR AMBI MICRO is for the Sennheiser AMBEO VR [17]. Among open-source tools, the SPARTA suite includes an array encoder with presets for several commercial A-format and higher-order arrays [18], while the IEM Plug-in Suite processes and visualises



Fig. 1. The physical A-format capture rig used for ENACT’s input: four Behringer C-2 cardioid capsules, each in an individual shock mount, arranged in a tetrahedral configuration.

signals that are already in Ambisonic format [19]. All these tools can operate in real time, but as DAW, standalone, or hardware processors outside an XR engine’s audio graph.

B. Spatial Audio in XR Engines

XR audio middleware operates at a different point of the spatial audio chain. Meta XR Audio can render an already encoded Ambisonic soundfield, while Resonance Audio provides spatial processing of positioned audio sources [20], [21]. Scene-aware rendering estimates or synthesises acoustic responses from scene information instead of capturing the soundfield with a physical microphone [22] (Section VI-C). None of these tools converts live signals from a physical tetrahedral microphone array inside the engine’s own audio callback.

C. Networked Immersive Audio and Musical XR

Research on networked music performance has characterised latency tolerance thresholds and enabling architectures [4], identified the latency factors of immersive networked systems [5], compared the latency of spatial audio plugins [23], and adapted the Ambisonics order to network conditions in real time [24]. Giacomelli et al. [25] demonstrated a distributed immersive audio production workflow in which native multichannel recordings were transmitted and processed in real time at remote production nodes. Networked XR music-performance systems handle immersive audio within dedicated production environments [9]–[11]. Real-time Ambisonics processing is thus not itself a new capability. ENACT addresses the complementary case in which the XR engine is the audio endpoint and must accept a live A-format stream: its contribution is the placement of the A-to-B conversion inside the engine’s own real-time audio callback, so that the resulting Ambisonic stream is directly available to the runtime audio graph.

III. BACKGROUND

A. Tetrahedral A-Format Capture

An FOA tetrahedral microphone is made of four capsules with cardioid directivity, arranged so that their principal axes point towards the corners of a regular tetrahedron. The capture rig used for ENACT (Fig. 1) has four Behringer C-2 cardioid

capsules, each with its own shock mount, arranged in this configuration. Each capsule is conventionally labelled with the octant it faces: Front-Left-Up (FLU), Front-Right-Down (FRD), Back-Left-Down (BLD), and Back-Right-Up (BRU). The four raw capsule signals recorded in this configuration are together called the A-format [12]. The A-format is microphone dependent: its exact relation to the sound field depends on the real capsule directivity, sensitivity matching, and precise capsule placement, which are not guaranteed to match exactly the idealised cardioid model [13].

B. A-to-B Conversion

The transformation from the four A-format capsule signals to B-format components is referred to as encoding or decoding in different Ambisonics literature and software. In this paper, we use the more general term *A-to-B conversion* when referring to this transformation.

Each cardioid capsule sums a pressure (omnidirectional) component with a velocity (directional) component. For this reason, sum and difference operations across the four capsule signals can recover the pressure and the three orthogonal velocity components that form the B-format [12]:

$$W_{\text{raw}} = \text{FLU} + \text{FRD} + \text{BLD} + \text{BRU}, \quad (1)$$

$$X_{\text{raw}} = \text{FLU} + \text{FRD} - \text{BLD} - \text{BRU}, \quad (2)$$

$$Y_{\text{raw}} = \text{FLU} - \text{FRD} + \text{BLD} - \text{BRU}, \quad (3)$$

$$Z_{\text{raw}} = \text{FLU} - \text{FRD} - \text{BLD} + \text{BRU}. \quad (4)$$

When the capsules are mounted with an inward-facing (convergent) orientation, instead of the outward-facing orientation used in Gerzon's formulation [12], the directional components are recorded with inverted polarity with respect to the acoustic scene, while the scalar pressure component W is not affected. The correction is a sign flip on the directional channels:

$$X = -X_{\text{raw}}, \quad Y = -Y_{\text{raw}}, \quad Z = -Z_{\text{raw}}. \quad (5)$$

C. The ambiX Standard and SN3D Normalisation

The resulting W, X, Y, Z signals need to be normalised and ordered according to an interchange convention before they can be considered a valid B-format stream. ENACT targets the *ambiX* convention: Ambisonic Channel Number (ACN) ordering with Schmidt semi-normalisation (SN3D) [8], [13]. For first-order signals, ACN places the channels in the order (W, Y, Z, X) , corresponding to indices 0, 1, 2, 3. With SN3D, a unit-amplitude plane wave from azimuth φ and elevation ϑ is encoded as $W = 1$, $X = \cos \varphi \cos \vartheta$, $Y = \sin \varphi \cos \vartheta$, and $Z = \sin \vartheta$; that is, the zeroth-order and first-order components have equal peak gain.

For an ideal cardioid capsule with response $\frac{1}{2}(1 + \mathbf{u} \cdot \mathbf{d}_i)$ for a plane wave from direction $\mathbf{u}$, with tetrahedral axes $\mathbf{d}_i \in \{(\pm 1, \pm 1, \pm 1)/\sqrt{3}\}$, Eqs. (1)–(4) give

$$W_{\text{raw}} = 2, \quad \{X, Y, Z\}_{\text{raw}} = \frac{2}{\sqrt{3}} \{u_x, u_y, u_z\}. \quad (6)$$

The raw directional sums are therefore a factor $\sqrt{3}$ weaker, relative to W_{raw} , than SN3D requires. Including the polarity correction of Eq. (5), the SN3D-normalised output is

$$W_{\text{out}} = \frac{1}{2} W_{\text{raw}}, \quad \{X, Y, Z\}_{\text{out}} = -\frac{\sqrt{3}}{2} \{X, Y, Z\}_{\text{raw}}. \quad (7)$$

In ENACT, the raw sums first receive a uniform gain of $1/2$ from the conversion matrix, and the three directional channels then receive an additional factor of $\sqrt{3}$.

The factor $\sqrt{3}$ holds for the ideal cardioid. For a general first-order capsule pattern $a + (1 - a) \cos \theta$, the required ratio of directional to omnidirectional gain becomes $\sqrt{3} a / (1 - a)$, which can be adjusted in the conversion core if the measured capsule pattern deviates from a cardioid (Section III-D).

D. Non-Coincidence Correction Filters

The conversion of Eqs. (1)–(4) and (7) assumes ideal, perfectly coincident cardioid capsules. In practice, the physical spacing between the capsules introduces non-coincidence errors, including phase and directivity deviations that increase with frequency and reduce the accuracy of the resulting B-format signal [15]. The actual sensitivity, directivity, and placement of each capsule introduce further deviations.

Practical A-to-B converters therefore combine the sum-and-difference matrix with correction filters derived from anechoic or controlled measurements of a specific microphone model or individual unit [15], [16]. Because these filters depend on the capsule spacing, directivity, and matching of the particular array, A-to-B conversion is usually distributed by manufacturers as software or hardware tied to a given microphone. For the same reason, open-source encoders also rely on presets for specific commercial arrays [18].

ENACT currently applies the matrix conversion without correction filters, and the physical array has not yet been calibrated against a reference measurement. Its output should be considered an uncorrected implementation of the ideal conversion rather than a calibrated representation of the physical microphone response. The engine-independent core is designed so that filters for the specific unit can be inserted before the matrix stage once the array has been measured, without changes to the engine integration (Section VII-A).

E. Higher-Order Extension

The formulation above is limited to FOA ($N = 1$, four channels). Extending the system to Higher-Order Ambisonics (HOA) would require either an array with a correspondingly larger number of microphone signals or an explicit estimation/upscaling method applied to the available tetrahedral signals. The second option does not provide additional measured spatial information directly and would require a dedicated evaluation. Both approaches also introduce additional considerations regarding channel count, bandwidth, and calibration, as discussed in the literature on networked performance [24].

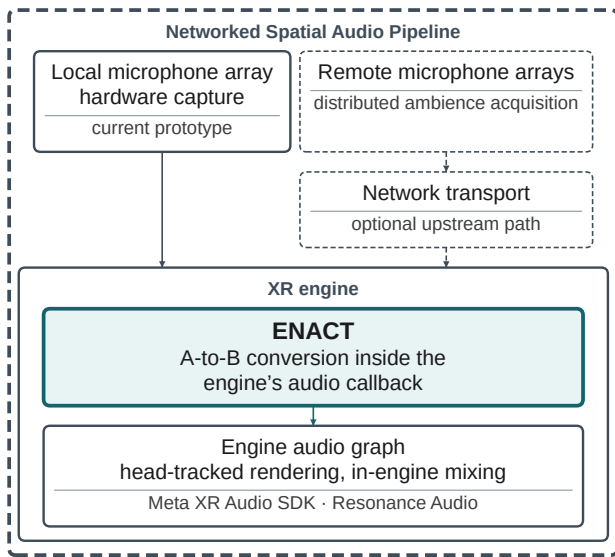


Fig. 2. ENACT within the networked spatial audio pipeline. The highlighted block is the scope of this paper: live A-to-B conversion inside the engine’s audio callback. ENACT runs inside the XR engine, which acts as the rendering endpoint. Its AmbiX output is intended to feed the engine audio graph, where it would be rendered relative to the tracked listener and mixed with in-engine sources (e.g., Meta XR Audio SDK, Resonance Audio); this downstream integration is not yet demonstrated in this paper (Table I). The A-format input comes from the local microphone array used in the current prototype (Section III-A) or, optionally, from remote arrays through network transport (distributed ambience acquisition, dashed). Components other than ENACT belong to the broader system and are not evaluated in this paper. Professional front-of-house or broadcast production [9]–[11] is not targeted.

IV. ENACT ARCHITECTURE AND METHODOLOGY

A. System Scope and Design Overview

ENACT is organised around an A-to-B conversion core that runs inside the engine’s real-time audio path. The conversion mathematics are kept separate from the Unity code, while a thin component connects the core to Unity’s audio callback. Fig. 2 shows how ENACT integrates with the engine audio graph.

The current prototype uses the tetrahedral array of four Behringer C-2 cardioid capsules described in Section III-A. The conversion core is not tied to this microphone model at the software level; the acoustic accuracy of its output depends on the array, as discussed in Section III-D.

The present work evaluates two aspects of ENACT. First, we consider the implementation of the A-to-B conversion inside Unity’s audio callback. Second, we evaluate the timing behaviour of the resulting execution chain against a DAW-hosted reference chain. The benchmark architecture is shown in Fig. 3.

B. Engine-Native A-to-B Conversion Core

The A-to-B conversion described in Section III, including the conversion matrix, the polarity correction for the inward-facing array, and the AmbiX scaling, is implemented in a module separate from the Unity classes. This separation allows

the conversion logic to be tested using known input signals without requiring a running scene or a live audio device.

The module also provides auxiliary functions used by ENACT, including a Fibonacci-sphere grid generator, a max- r_E -weighted directional decoder, exponential smoothing of the level meters over the block duration, and an estimator of azimuth and elevation from the active intensity vector. The visualisation of directional energy shown in Fig. 4 uses the Fibonacci-sphere decoder.

The visualisation follows the general approach of decoding to many directions used by the IEM Plugin Suite’s EnergyVisualizer [19]. ENACT uses an independent implementation with a Fibonacci-sphere distribution of up to 400 points and does not link against or reuse code from the IEM Plugin Suite.

Keeping the conversion core separate from the engine code was also useful during implementation. Tests on the isolated module exposed an earlier error in the mapping between Unity’s left-handed coordinate system and the right-handed convention used by the Ambisonics formulation. The corrected mapping is used in the current implementation.

A `MonoBehaviour` wrapper connects the core to a four-channel `AudioSource` and executes it from `OnAudioFilterRead`. The callback runs on Unity’s audio thread, so the conversion is driven by the audio buffer processing cycle rather than by the rendering frame rate. A main thread implementation based, for example, on polling `AudioClip.GetData()` would instead be affected by the rendering loop. If the callback does not receive four channels, the wrapper leaves the buffer unchanged rather than applying an invalid conversion.

The converted signal is written back to the same callback buffer as four channels in ACN/SN3D order. At this point Unity treats it as ordinary four-channel audio: Unity’s documented Ambisonic workflow applies the selected Ambisonic decoder to audio clips flagged as Ambisonic at import, and ENACT’s live callback output does not pass through that path. In the present prototype, the converted output is inspected only through ENACT’s own directional energy decoder (Fig. 4). Routing the live stream into an Ambisonic renderer, applying head-tracked rotation, and rendering it to a listener are not yet implemented or demonstrated (Table I, Section VII-C).

C. Implementation and Evaluation Scope

Table I separates the functionality implemented in the current prototype from the aspects that have been experimentally evaluated and those that remain future work. This distinction is important because implementation of the ideal A-to-B conversion does not by itself establish the acoustic accuracy of the physical microphone array.

The timing evaluation was selected because the architectural question considered here concerns where the conversion stage is placed and which processing path is needed to make the converted soundfield available inside the engine. Timing is therefore the metric that discriminates between the tested architectures. It is not an acoustic evaluation of the encoder

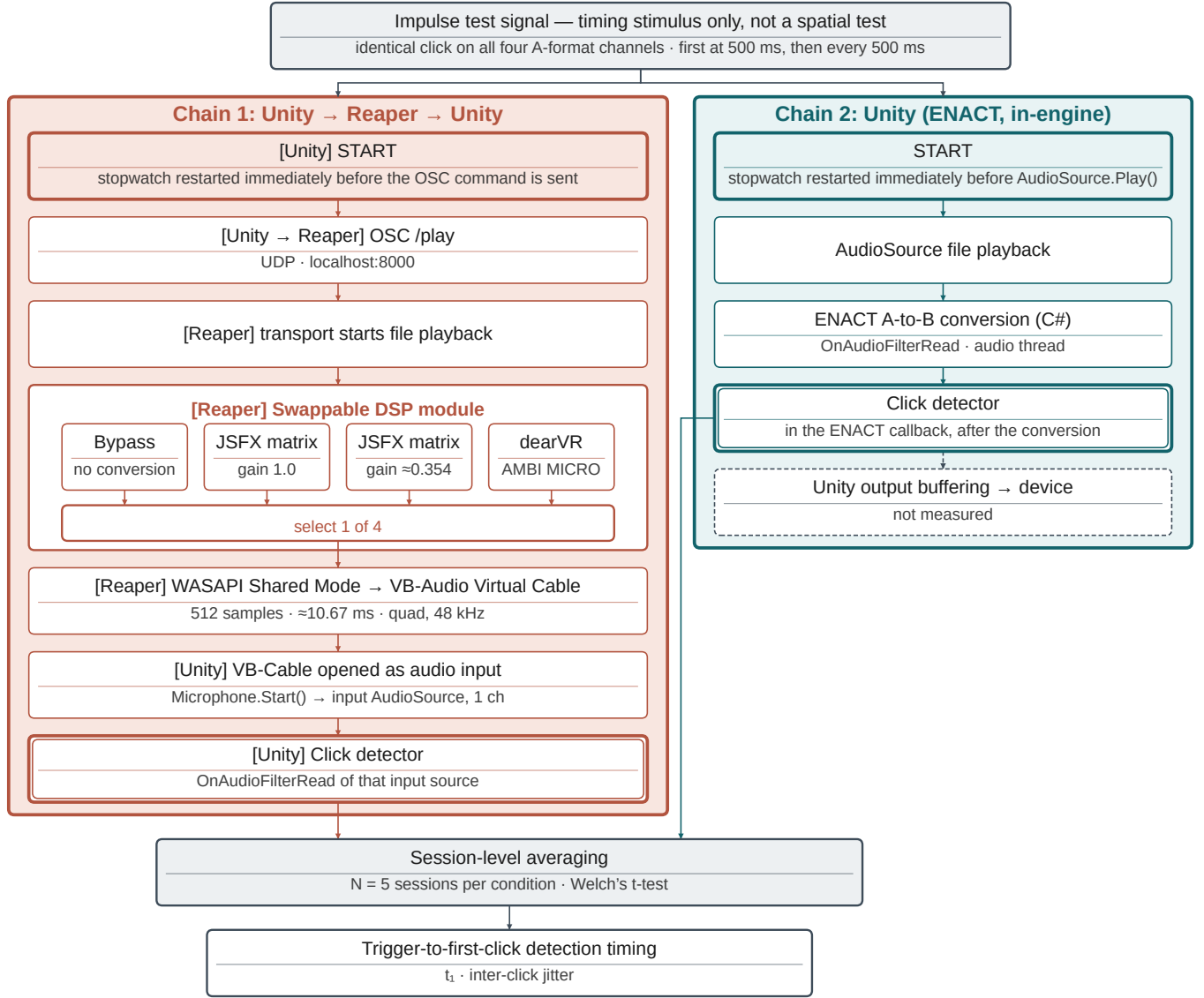


Fig. 3. Benchmark architecture used in this work. The same impulse file, with an identical click on all four channels, is used as a timing stimulus by Chain 1 (Reaper, DAW-hosted; bypass, JSFX, or dearVR AMBI MICRO processing) and Chain 2 (ENACT, in-engine); it is not a test of spatial encoding. In each chain, the stopwatch starts at the trigger (START): immediately before the OSC /play command in Chain 1 and immediately before `AudioSource.Play()` in Chain 2. The chains are benchmarked in separate executions, both timed in Unity, and each ends in its own copy of an identical click detector, placed at a different location in the signal path. Chain 1 is a round-trip loop: Unity sends OSC /play to Reaper, Reaper’s output returns to Unity through the virtual cable, captured as a single channel with `Microphone.Start()`, and is detected in `OnAudioFilterRead` of that input source. In Chain 2, playback, conversion, and detection all take place in the same `OnAudioFilterRead` callback, before Unity’s output buffering and the audio device. The measured intervals therefore include different control, routing, buffering, and execution stages.

output, which would additionally require the calibration and work on correction filters described in Section III-D.

D. Timing Methodology

The timing experiment compares ENACT with a DAW-hosted reference chain implemented in Reaper [26]. Both configurations use the same four-channel test signal, containing impulses spaced exactly 500 ms apart, with the first impulse located at 500 ms from the start of the file, as shown in Fig. 3. A detection time t_1 equal to 500 ms would correspond to zero additional delay with respect to the file timeline. The click is

identical on the four A-format channels, so after conversion it excites only the omnidirectional component W . The signal is thus a timing stimulus only and provides no information about the directional or spatial accuracy of the conversion.

In Chain 2, the test signal is played inside Unity and ENACT performs the A-to-B conversion described in Section IV-B. Unity’s output speaker mode is set to Quad, so that the callback receives the four channels of the test signal, and the detector runs after ENACT in the same callback chain, so that it processes the converted signal. For each run, `System.Diagnostics.Stopwatch` is restarted im-

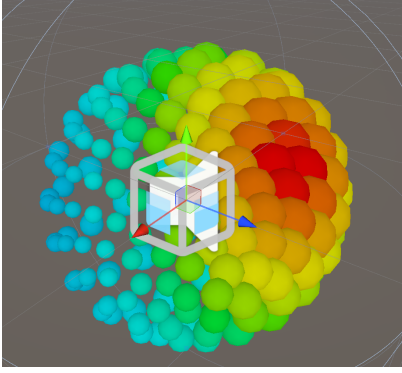


Fig. 4. Visualisation of directional energy produced by ENACT's Fibonacci-sphere decoder on a live test signal inside the Unity Editor. Colour represents the RMS energy computed for each virtual direction.

TABLE I
IMPLEMENTATION AND EVALUATION STATUS OF ENACT.

Component or aspect	Status
Ideal A-to-B conversion, polarity correction, and AmbiX (SN3D) scaling	Implemented; not yet timed
Unity audio callback integration	Implemented; exercised in real time
Unity–Ambisonics coordinate mapping	Verified with reference inputs
Visualisation of directional energy	Implemented; demonstrated
Routing of live output to an Ambisonic decoder; head-tracked rendering	Not yet demonstrated
Calibration of the physical array	Not yet performed
Microphone-specific correction filters	Not implemented
DAW-hosted benchmark chain	Implemented
Shared click detection and session-level analysis	Implemented
Timing comparison of the two chains	Evaluated with earlier gain setting (Sec. V)
Signal/spatial accuracy of the physical array	Not yet experimentally evaluated
HOA extension and native engine-independent plugin	Future work

mediately before the call to `AudioSource.Play()`. Click timestamps are then registered when the detector processes the corresponding samples in `OnAudioFilterRead`. Detection in Chain 2 thus occurs when the samples are processed on the audio thread, before Unity's output buffering and the audio device; the Chain 2 interval does not include output device latency.

Chain 1 is implemented as a round-trip loop that starts and ends in Unity. For each run, Unity restarts the stopwatch immediately before it sends the Open Sound Control (OSC) `/play` command over UDP to a separate Reaper process at `localhost:8000`. Reaper then plays the same test file, applies the selected processing condition, and outputs the result to VB-Audio Virtual Cable [27]. The virtual device is configured for four channels at 48 kHz, with WASAPI Shared Mode and a verified 512-sample buffer (≈ 10.67 ms). The virtual cable is in turn opened in Unity as an audio input device

with `Microphone.Start()`, and the captured signal is played by an `AudioSource` whose `OnAudioFilterRead` callback runs the click detector. Unity's microphone input delivers a single channel, so the Chain 1 detector processes the first channel of the virtual cable signal: FLU in the bypass condition and W (ACN 0) with the JSFX and dearVR modules. Because the click is identical on the four A-format channels, this channel carries the impulse in every condition. The Chain 1 interval therefore includes OSC control, Reaper transport start and processing, WASAPI output buffering, the virtual cable, and Unity's audio input capture and buffering, none of which is part of the Chain 2 interval.

Two processing modules are evaluated inside the Reaper chain. The first is a custom 4-in/4-out JSFX implementation of the same conversion used by ENACT. The JSFX processing order is the A-to-B sum-and-difference matrix, the polarity correction and scaling, followed by a global `masterGain`. With `masterGain = 1.0`, its conversion coefficients are algebraically identical to those used by ENACT. A second JSFX condition uses `masterGain  $\approx 0.354$  ( $1/(2\sqrt{2})$ )` as an additional global gain condition included in the benchmark.

The timing measurements reported in Section V were acquired with an earlier gain setting of the prototype ($1/\sqrt{2}$ for W and $1/2$ for X, Y, Z), applied identically in ENACT and in the JSFX module. Eq. (7) changes the relative weighting of the omnidirectional and directional channels. The global gain check described below scales all channels equally, so it does not test whether this change in relative weighting affects a detector that takes the maximum across channels. The reported timing values consequently characterise the tested paths with the earlier gain setting; the corrected coefficients have not been benchmarked. However, the test impulse is identical on the four A-format channels, so the directional sums of Eqs. (2)–(4) are zero and only W carries the impulse in both ENACT and the JSFX module. For this stimulus, the correction therefore reduces to a constant scaling of W from $1/\sqrt{2}$ to $1/2$ (approximately -3 dB). This is smaller than the approximately -9 dB global gain step between the two JSFX conditions, for which no timing difference was detected (Section V). We expect the corrected coefficients not to change the reported timing for this stimulus; this remains an inference until the corrected implementation is timed.

The second reference module is the commercial dearVR AMBI MICRO VST3 plug-in [17], configured in its “Upside Down” mode, with plug-in delay compensation verified at zero samples. dearVR AMBI MICRO is designed for the Sennheiser AMBEO VR microphone, while the physical array used in this work has a different construction. The cited documentation describes correction for the AMBEO VR microphone but does not provide the internal conversion coefficients used by the plug-in. For this reason, dearVR is used here as a practical DAW-hosted timing reference and not as sample-level or acoustic ground truth for the ENACT microphone.

The two chains are benchmarked in separate executions. In both, timing uses `System.Diagnostics.Stopwatch` in

Unity, and the clicks are detected by an identical detector implementation running in the `OnAudioFilterRead` callback of a `Unity AudioSource`: the virtual cable input source in Chain 1 and the ENACT source in Chain 2. Only the location of the detector in the signal path differs. The detector applies an threshold of 0.5 to the maximum absolute sample value across the channels of each frame (four channels in Chain 2, one in Chain 1), followed by a 250 ms cooldown. Because the cooldown is shorter than the 500 ms click period, a single click cannot be counted twice; the cooldown does not, however, guarantee that every click is detected. The timestamp is read from the stopwatch when the callback processes the block containing the threshold crossing, so timestamps are quantised to the audio block duration rather than to individual samples. In Chain 2, using the maximum across the channels avoids depending on one fixed channel when the impulse energy is distributed differently by the processing.

The bypass condition in Chain 1 applies no A-to-B conversion. It provides a reference for evaluating whether activating the conversion module produces a measurable timing change inside the same external execution path. The JSFX and dearVR conditions provide additional comparisons within this path. The two chains use different trigger points. Chain 1 starts timing immediately before sending the `OSC /play` command, whereas Chain 2 starts immediately before `AudioSource.Play()`. In both cases, the endpoint is the same threshold click detector running in Unity, but it processes the captured virtual cable input in Chain 1 and the internally converted signal in Chain 2. Consequently, the difference between the chains represents the trigger-to-first-click detection timing of the two complete benchmark paths under the tested configuration. It must not be interpreted as the isolated computational latency of the A-to-B conversion.

Each condition was evaluated in five sessions, with 10 runs of 15 clicks per session. Consecutive runs are separated by a 1.5 s pause; in Chain 1, an `OSC /stop` command is also sent 0.5 s before each `/play`. The stopwatch is restarted once at the beginning of each run, and the detector stores the cumulative elapsed time for each of the 15 detected clicks. Startup timing is defined as the timestamp of the first detected click (t_1). Each session provides 10 startup observations, which are averaged to obtain one session-level value. One Unity observation (Session 2, Run 1; 663.08 ms) was identified as a cold-start event in the benchmark report and excluded from that session mean. Because runs within the same session share operating system, driver, buffering, and warm-up conditions, the five session-level means, rather than the individual runs, are used as the independent observations ($N = 5$) for the statistical comparisons reported in Section V.

Inter-click timing is analysed separately from startup timing. For a 15-click run, consecutive intervals are computed as $\Delta t_i = t_i - t_{i-1}$ for $i = 2, \dots, 15$, giving 14 intervals per run and 140 intervals per session before exclusions. Four intervals of approximately 996 ms occurred in the Reaper bypass condition, all in Session 4. Because they are approximately twice the nominal 500 ms period, they were treated as missed clicks

TABLE II
MEAN TRIGGER-TO-FIRST-CLICK DETECTION TIMING BY CONDITION ($N = 5$ SESSION-LEVEL VALUES PER CONDITION), MEASURED WITH THE EARLIER GAIN SETTING (SECTION IV-D); THE CORRECTED SN3D IMPLEMENTATION HAS NOT BEEN TIMED.

Condition	Mean (ms)	SD (ms)
ENACT / Chain 2 (Unity, in-engine)	504.54	2.56
Chain 1 bypass (Reaper baseline)	708.77	25.06
Chain 1 + dearVR AMBI MICRO	693.43	6.58
Chain 1 + JSFX (masterGain= 1.0)	698.98	16.60
Chain 1 + JSFX (masterGain $\approx$ 0.354)	698.99	15.22

in the inter-click analysis. These events occur after the first detected click, so they do not alter the startup observations used in Table II. The logged timestamps do not show whether these misses arose in the signal path (e.g., a dropout in the virtual cable or input capture) or in the detector.

Amplitude scaling can in principle change the exact threshold sample of a band-limited signal. For this reason, the two JSFX gain conditions were also compared inside the same Reaper path. Their results are reported in Section V. This check is used only to assess whether the tested gain setting produces a measurable timing difference with the current detector; it is not a test of equivalence of the signals or acoustic accuracy.

V. RESULTS

A. Trigger-to-First-Click Timing

Under the tested configuration (Table II), the ENACT chain shows a mean trigger-to-first-click detection timing of 504.54 ± 2.56 ms, while the Reaper bypass condition shows 708.77 ± 25.06 ms. The mean difference between the two complete benchmark paths is therefore approximately 204 ms. A Welch's t -test performed on the $N = 5$ session-level observations per condition detects a statistically significant difference ($t \approx -18.13$, $p \approx 4.7 \times 10^{-5}$). Given the limited number of independent sessions, this inferential result is interpreted together with the descriptive statistics and only for the tested configuration.

Relative to the 500 ms position of the first impulse in the test file, these values correspond to approximately 4.5 ms for ENACT and 208.8 ms for the Reaper bypass condition. As discussed in Section IV-D, this difference characterises the two complete benchmark paths, with different start points and detection points, and not the computational cost of the A-to-B conversion. In particular, it is not evidence of a general latency advantage of engine-native processing.

Excluding the Unity cold-start observation (Section IV-D) does not change this conclusion. Assuming the remaining nine Session 2 observations are close to the reported ENACT mean, including the 663.08 ms observation would raise the Session 2 mean by approximately 16 ms and the ENACT mean across sessions by approximately 3 ms, to about 508 ms, with a SD across sessions of about 7.5 ms. The difference from the Reaper bypass condition would remain approximately 201 ms ($t \approx -17$).

Some Unity startup observations and some inter-click intervals are slightly below the nominal 500 ms reference period, with values around 485 ms. Such variations are consistent with operating system scheduling and audio buffer block quantisation, including the 512-sample WASAPI blocks used in the reference configuration. A threshold crossing can consequently be registered in an earlier or later processing block. In Chain 2, the audio thread can also process buffers ahead of their playback at the device, which is consistent with detections slightly before the nominal impulse position.

The comparisons performed within Chain 1 provide additional information about the contribution of the A-to-B processing itself. The Reaper bypass condition gives 708.77 ± 25.06 ms, while the same chain with dearVR AMBI MICRO active gives 693.43 ± 6.58 ms. A Welch's t -test does not detect a statistically significant difference between these two conditions ($t \approx -1.32$, $p \approx 0.25$). This result does not establish equivalence between them, but it gives no evidence in the present measurements that activation of the Ambisonics conversion accounts for a timing increase comparable to the approximately 204 ms difference observed between the two complete chains.

A similar result is obtained with the custom JSFX module. The condition with `masterGain= 1.0` gives 698.98 ± 16.60 ms, while the additional `masterGain $\approx$  0.354` condition gives 698.99 ± 15.22 ms. No statistically significant difference is detected between these two conditions ($t \approx 0.00$, $p \approx 0.999$). Within the resolution and sample size of the present experiment, the tested global gain setting does not produce a measurable change in the reported timing metric. The measurements within Chain 1 are consistent with the interpretation that the A-to-B computation itself is not responsible for most of the timing difference between the chains. The benchmark does not determine how much of that difference is contributed by OSC control, virtual audio routing, the audio driver, Unity's audio input capture, buffer alignment, or other stages specific to each architecture, nor does it establish that engine-native processing is generally faster than professional production systems. Separating these contributions follows the approach of prior work on end-to-end latency in networked music performance [5] and on the latency of spatial audio plug-ins [23], and is left for future work (Section VII-C).

B. Jitter

Intra-session jitter was computed from the consecutive inter-click intervals defined in Section IV-D. The ENACT chain shows a mean intra-session jitter of approximately 13 ms. For the reference chain with dearVR AMBI MICRO active, the values across the five sessions are 14.85, 15.26, 13.54, 13.24, and 15.75 ms, respectively. The individual jitter values per session for the Unity condition were not retained in the benchmark data. For this reason, only its overall mean is available and no statistical comparison of jitter between the two chains is possible. The available values are reported descriptively only; they indicate similar intra-session variability but support no conclusion about equivalence or difference. The four intervals

with missed clicks excluded in Section IV-D belong to the Reaper bypass condition and do not affect the jitter values reported here.

Table II also reports the standard deviation of the mean trigger-to-first-click timing. This value is 2.56 ms for ENACT and 6.58 ms for the dearVR reference condition. These values describe the variability observed in the five sessions of the present experiment; given the limited number of independent sessions, they are reported descriptively rather than as evidence of a general stability advantage.

VI. DISCUSSION

A. Positioning Relative to Existing Ambisonics and XR Audio Workflows

As reviewed in Section II, A-to-B conversion, including open-source implementations [18], and real-time Ambisonics processing in distributed production systems [25] already exist. We therefore limit the claimed gap to the engine-native use case: the difference considered here is where the conversion runs, inside the XR engine's audio callback rather than in a DAW host or a remote processing node. ENACT's AmbiX output can then be used by a compatible rendering stage in the same way as any other B-format source.

B. Engine-Native Use Cases and Production Scope

The main practical consequence of the engine-native architecture is that the converted soundfield is a native signal of the runtime audio graph (Fig. 2). Consider a remote listener in an XR application who hears the live ambience of a venue captured by the tetrahedral array. Once connected to an Ambisonic renderer, the AmbiX stream could be rotated by the listener's head-tracking data, mixed with synthetic in-engine sources, and attenuated or ducked according to the application state, for instance when the listener moves between virtual rooms, all within one audio graph and one audio clock. This downstream path motivates the architecture but is not yet demonstrated (Section IV-B). An external chain hosted in a DAW or on a network node can provide equivalent functions, but it requires an additional host, a routing stage, and synchronisation between the external process and the application logic. ENACT removes these stages where the engine itself is the intended audio endpoint.

This scope is different from professional live-sound or broadcast production. Networked XR music-performance systems such as those of Cairns *et al.* [9], Genovese *et al.* [10], and Hupke *et al.* [11] handle immersive audio as part of the production infrastructure, where latency constraints and the need for a mix controlled by an engineer favour dedicated production environments over processing inside a game engine. ENACT does not target these workflows.

For the same reason, DAW-hosted processing is not equivalent to offline processing: a DAW chain can run in real time, as the reference configuration of this work shows. The distinction made here is only between processing inside and outside the engine audio graph.

C. Relationship to Scene-Aware Acoustic Rendering

Scene-aware acoustic rendering addresses a related problem from a different direction. Systems such as the approach described in [22] estimate or synthesise acoustic responses from scene information instead of capturing the soundfield with a physical microphone. ENACT assumes that a physical capture array is available and makes its converted signal accessible inside the engine. The two approaches can be complementary. A future system could use live capture where a physical array is available and acoustic processing derived from the scene for other parts of the virtual environment. This integration is outside the scope of the present work.

VII. LIMITATIONS AND FUTURE WORK

A. Acoustic Validation and Microphone Correction

The next step is to characterise the physical array under controlled conditions, derive correction filters for the specific unit (Section III-D), and evaluate the converted output against reference measurements through signal, spectral, and spatial/directivity analysis. These tests complement the timing evaluation reported here and are required before the capture system can be considered fully characterised.

B. Implementation Portability and Higher-Order Extension

ENACT is currently implemented as a Unity C# component. The A-to-B conversion core is already separated from the Unity wrapper, which provides a basis for a future migration to a native, engine-independent audio plug-in. Such a version could allow the same conversion core to be used outside the current Unity prototype and would remove its dependency on the C# scripting layer. Any performance benefit of this migration should be measured separately rather than assumed.

The current system is also limited to FOA. A HOA extension would require either a capture array with additional spatial information or an explicit estimation/upsampling method applied to the available signals. In both cases, the extended system would require its own calibration and evaluation. This work is considered a later development rather than part of the present prototype.

C. Extended Timing and System Evaluation

Future timing work will decompose the measured paths into their individual contributions (audio driver, buffering, OSC control, virtual audio routing, Unity's audio input capture, A-to-B processing, and downstream rendering) and repeat the comparison for different audio buffer sizes, to distinguish fixed contributions of each architecture from effects that depend on the buffer. The benchmark will also be repeated with the corrected coefficients of Eq. (7), with jitter per session retained for all conditions and a detection point at the audio device output for both chains. The first integration step is to route ENACT's live output into an Ambisonic renderer with head-tracked rotation inside the engine, followed by integration with the broader networked spatial audio system of Fig. 2.

VIII. CONCLUSION

This paper presented an architectural pattern for live tetrahedral A-to-B Ambisonics conversion inside an XR engine's real-time audio callback: an engine-independent conversion core, a thin engine wrapper driven by the audio thread rather than the rendering loop, and a benchmark that compares the resulting execution path with an external DAW-hosted chain. The Unity prototype ENACT shows that the conversion can run in the engine callback in practice. Under the tested configuration, the ENACT path showed a significantly shorter trigger-to-first-click detection timing than the external chain, while activating the conversion inside the external chain produced no measurable timing change. These results characterise the two tested paths only and do not establish a general latency advantage.

Because the conversion core is decoupled from Unity, the same pattern can be ported to other engines or to a native plug-in, and microphone-specific correction filters or HOA extensions can be added without changing the engine integration. For the Internet of Sounds and musical XR, the pattern lets a live tetrahedral array act as a Sound Thing whose stream reaches an XR endpoint in a shared representation, without an additional host or routing stage in between. ENACT should be regarded as a working conversion prototype in the engine callback, with an initial comparison of complete execution paths. Routing its output into Ambisonic rendering with head tracking, acoustic calibration, and correction filters are the next steps towards a fully characterised capture system.

ACKNOWLEDGMENT

The research activities described in this paper were carried out within the "HEAT – Hybrid Extended reAlity" Project GA 101135637 funded by the EU Horizon Europe Framework Programme (HORIZON) and supported by a grant of the Romanian Ministry of Research, Innovation and Digitization, CNCS/CCCDJ - UEFISCDI, project number PN-IV-P8-8.1-PRE-HE-ORG-2025-0331, within PNCDI IV.

REFERENCES

- [1] L. Turchet, M. Lagrange, C. Rottondi, G. Fazekas, N. Peters, J. Østergaard, F. Font, T. Bäckström, and C. Fischione, "The Internet of Sounds: Convergent trends, insights, and future directions," *IEEE Internet of Things Journal*, vol. 10, no. 13, pp. 11 264–11 292, 2023.
- [2] L. Turchet, C. Fischione, G. Essl, D. Keller, and M. Barthet, "Internet of Musical Things: Vision and challenges," *IEEE Access*, vol. 6, pp. 61 994–62 017, 2018.
- [3] L. Turchet, R. Hamilton, and A. Çamci, "Music in extended realities," *IEEE Access*, vol. 9, pp. 15 810–15 832, 2021.
- [4] C. Rottondi, C. Chafe, C. Allocchio, and A. Sarti, "An overview on networked music performance technologies," *IEEE Access*, vol. 4, pp. 8823–8843, 2016.
- [5] L. Turchet and M. Tomasetti, "Immersive networked music performance systems: Identifying latency factors," in *2023 Immersive and 3D Audio: From Architecture to Automotive (I3DA)*. Bologna, Italy: IEEE, 2023, pp. 1–6.
- [6] S. Ppali, M. Scorer, E. Ppali, B. Branch, and A. Covaci, "Remote rhythms: Audience-informed insights for designing remote music performances," in *Proceedings of the 2024 ACM Designing Interactive Systems Conference (DIS '24)*. New York, NY, USA: Association for Computing Machinery, 2024, pp. 2675–2690. [Online]. Available: <https://doi.org/10.1145/3643834.3660719>

- [7] S. Ppali, B. Branch, A. Boem, E. Ppali, M. Scorer, C. S. Ang, L. Turchet, and A. Covaci, "The virtual concert-goer: Audience perspectives on remote music performances," *Proc. ACM Hum.-Comput. Interact.*, vol. 9, no. 7, Nov. 2025. [Online]. Available: <https://doi.org/10.1145/3757431>
- [8] C. Nachbar, F. Zotter, E. Deleflie, and A. Sontacchi, "ambix – a suggested ambisonics format," in *Proceedings of the Ambisonics Symposium*, Lexington, KY, 2011.
- [9] P. Cairns, A. Hunt, D. Johnston, J. Cooper, B. Lee, H. Daffern, and G. Kearney, "Evaluation of metaverse music performance with bbc maida vale recording studios," *Journal of the Audio Engineering Society*, vol. 71, no. 6, pp. 313–325, June 2023.
- [10] A. F. Genovese, Z. Nguyen, M. Gospodarek, R. Pahle, C. Brenner, and A. Roginska, "Holodeck: A research framework for distributed multimedia concert performances," in *2024 IEEE 5th International Symposium on the Internet of Sounds (IS2)*. Erlangen, Germany: IEEE, 2024, pp. 1–10.
- [11] R. Hupke, S. Preihs, and J. Peissig, "Immersive room extension environment for networked music performance," in *Audio Engineering Society Convention 153*, Oct. 2022, express Paper 28. [Online]. Available: <https://aes2.org/publications/elibrary-page/?id=21909>
- [12] M. A. Gerzon, "Periphony: With-height sound reproduction," *Journal of the Audio Engineering Society*, vol. 21, no. 1, pp. 2–10, 1973, aES E-Library <https://www.aes.org/e-lib/browse.cfm?elib=2012>.
- [13] F. Zotter and M. Frank, *Ambisonics: A Practical 3D Audio Theory for Recording, Studio Production, Sound Reinforcement, and Virtual Reality*, ser. Springer Topics in Signal Processing. Cham, Switzerland: Springer, 2019, vol. 19.
- [14] M. Frank, F. Zotter, and A. Sontacchi, "Producing 3d audio in ambisonics," in *AES 57th International Conference: The Future of Audio Entertainment Technology – Cinema, Television and the Internet*, Hollywood, CA, 2015, aES E-Library <http://www.aes.org/e-lib/browse.cfm?elib=17605>.
- [15] C. Faller and M. Kolundzija, "Design and limitations of non-coincidence correction filters for soundfield microphones," in *AES Convention 126*. Munich, Germany: Audio Engineering Society, May 2009, convention Paper 7766.
- [16] A. Farina, "A-format to B-format conversion," <https://www.angelofarina.it/Public/B-format/A2B-conversion/A2B.htm>, accessed Sep. 2026.
- [17] Sennheiser electronic SE & Co. KG, "Get Access to Your Ambisonics Track via dearVR AMBI MICRO," <https://help.sennheiser.com/hc/en-us/articles/27728862252690-Get-access-to-you-ambisonics-track-via-dearVR-AMBI-MICRO>, accessed Sep. 2026.
- [18] L. McCormack and A. Politis, "SPARTA & COMPASS: Real-time implementations of linear and parametric spatial audio reproduction and processing methods," in *Proc. AES International Conference on Immersive and Interactive Audio*, York, UK, Mar. 2019.
- [19] Institute of Electronic Music and Acoustics (IEM), "IEM Plug-in Suite," <https://plugins.iem.at/>, University of Music and Performing Arts Graz, accessed Jul. 2026.
- [20] Meta Platforms, Inc., "Play Ambisonic Audio in Unity – Meta XR Audio SDK," <https://developers.meta.com/horizon/documentation/unity/meta-xr-audio-sdk-unity-ambisonic/>, accessed Jul. 2026.
- [21] Google LLC, "Resonance Audio SDK for Web – Getting Started," <https://resonance-audio.github.io/resonance-audio/develop/web/getting-started>, accessed Jul. 2026.
- [22] T. Xu, J. Li, P. Zu, P. Sahay, M. Kim, J. Obeng-Marnu, F. Miller, X. Qian, K. Passarella, M. Rachumalla, R. Nongpiur, and D. Shin, "Enhancing xr auditory realism via multimodal scene-aware acoustic rendering," in *Proceedings of the 38th Annual ACM Symposium on User Interface Software and Technology*, ser. UIST '25. New York, NY, USA: Association for Computing Machinery, 2025.
- [23] M. Tomasetti, A. Farina, and L. Turchet, "Latency of spatial audio plugins: A comparative study," in *2023 Immersive and 3D Audio: From Architecture to Automotive (I3DA)*. Bologna, Italy: IEEE, 2023, pp. 1–10.
- [24] P. Ostan, C. Centofanti, M. Pezzoli, A. Bernardini, C. Rinaldi, and F. Antonacci, "Dynamic real-time ambisonics order adaptation for immersive networked music performances," 2025. [Online]. Available: <https://arxiv.org/abs/2508.00509>
- [25] S. Giacomelli, C. Centofanti, J. P. Pereira dos Santos, M. Galbiati, T. Salvi, F. Graziosi, and C. Rinaldi, "Remote immersive audio production: State of the art implementation, challenges, and improvements," in *2024 IEEE 5th International Symposium on the Internet of Sounds (IS2)*. Erlangen, Germany: IEEE, 2024, pp. 1–10.
- [26] Cockos Incorporated, "REAPER – Digital Audio Workstation," <https://www.reaper.fm/>, accessed Sep. 2026.
- [27] VB-Audio Software, "VB-CABLE Virtual Audio Device," <https://vb-audio.com/Cable/>, accessed Sep. 2026.