DOI: 10.22032/dbt.39962
DOI (proceedings): 10.22032/dbt.39936

Verband Deutscher Tonmeister e.V.

Full Reviewed Paper at ICSA 2019

Presented* by VDT.

The Virtual Acoustic Spaces Unity Spatializer with custom head tracker

T. Resch[1], M. Hädrich[2]

[1] *Hochschule für Musik Basel FHNW, Research and Development | TU Berlin, Audiocommunication Group*
Email: thomas.resch@fhnw.ch

[2] *TU Berlin, Audiocommunication Group, Germany, Email: markus.haedrich@tu-berlin.de*

Abstract

The virtual acoustic spaces (VAS) unity spatializer is a plugin for dynamic binaural synthesis for Unity. It can handle impulse responses (IRs) of arbitrary length (limited only by hardware resources). Hence, it is possible to calculate the binaural synthesis not only with head related transfer functions (HRTFs), but also on the basis of binaural room impulse responses (BRIRs). The plugin can also virtualize reflections calculated by raytracing and it is possible to load an individual IR set for each instance. In addition to being compatible with off-the-shelf cross reality (XR) hardware it features a Bluetooth binding for an easily built, custom-made head tracker based on an ESP32 board. It is therefore predestined for audio augmented reality applications.

1. Introduction

The game engine Unity has become very popular. Not only in the field of game development, but also in scientific areas, for example in the virtualization of acoustic environments. In combination with off-the-shelf XR systems such as the HTC Vive, Unity provides a very simple setup for dynamic binaural synthesis: the angles (azimuth and elevation) between sound source and the person wearing the XR glasses are sent directly from the integrated head tracker to the spatializer plugin. The binaural synthesis is calculated depending on the user's head orientation. But this is only a solution if a complete virtual reality experience is desired. For an audio-only augmented reality (AR), a standalone head tracker is necessary. Furthermore, the existing binaural spatializers for Unity can either not load custom IRs, or at most one set, or the length of the IRs is limited. Most of them are not available for all operating systems. Due to these constraints (which are described in detail in section 2) a first version of the VAS Unity Spatializer was developed for the project Analog Speicher [1] where architecture and the corresponding acoustics of various ancient buildings were simulated. The plugin had to be able to load ten different BRIR sets with lengths of up to 0.5 s into one Unity scene simultaneously. For the game LosEmal which is currently being developed for the project Myosotis [2], three further requirements were added: firstly, the plugin had to support iOS. Secondly (because the target platform is not a VR system), a connection to a standalone head tracker had to be implemented, because the game principle relies on a well-functioning binaural synthesis. And third, early reflections for less reflecting outdoor environments as described in [3] should be simulated to make the binaural synthesis more plausible.

Therefore, a new version of the plugin was developed with iOS and OSX bindings to an inexpensive, custom-made Bluetooth head tracker, based on an Adafruit Huzzah32 development board. The sensor fusion was realized with a Sparkfun BNO080 inertial measurement unit (IMU) because according to its datasheets, the BNO080 is supposed to perform an outstanding sensor fusion and has not been used in any open source projects yet.

This paper starts with a brief discussion of related work in section 2. Section 3 describes the setup of the native Unity plugin, the corresponding C# scripts, the head tracker and its communication with Unity. Section 4 outlines the implementation details of all components. Section 5 deals with measurement results regarding latency and CPU usage.

2. Related Work

Several binaural spatializer plugins are available for Unity. The Oculus plugin [4] supports Android, OSX and Windows. It cannot handle custom IR sets. Microsofts plugin is neither able to do this, nor is it compatible with systems other than Windows [5]. Resonance Audio by Google supports all platforms, but it is not documented how custom HRTFs can be used [6]. Steam Audio provides an SDK and a spatializer for Unity [7]. It supports the Sofa file format [8] and can thus load custom HRTFs and render dynamic binaural synthesis. There is no support for iOS yet and only one IR set can be loaded globally for all plugin instances. The SOFAlizer for Unity is capable of loading up to 10 different HRTFs, but the impulse responses are always shortened to 256 samples. According to the developers there is only support for Windows [9]. The Soundscape Renderer (SSR) [10] [11] is a C++ software capable of rendering dynamic binaural synthesis. In combination with a virtual sound device such as Jack it can be used in conjunction with Unity. However, compiling the SSR for iOS or Android is not documented. EVERTims [12] [13] is a framework for the auralization of 3D models with raytracing for OSX, Windows and Linux. It's based on the Accelerated Beam Tracing Algorithm by Lane, Siltanen, Lokki and Savioja [14]. While it looked promising, the Binaural Synthesis Kit [15] is not yet available for download. Open source head tracker projects such as the Hedrot by Alexis Baskind [16], the EDTracker [17], the open headtracker [18] or the MrHeadTracker by Romanov, Berghold, Rudrich, Zaunschirm, Frank, Zotter [19] all have a wired transmission only. Robert Twomey's bluetooth-headtracker [20] comes with Bluetooth but the used sensor board is no longer available. The very advanced project DIY-low-cost-head-tracker with sensor fusion, BLE- and serial connection by Sascha Spors [21] uses the MPU9250 which will be deprecated soon.

3. Setup and availability

The presented solution consists of four components: the plugin, the scripts for plugin configuration, the head tracker and a small Bluetooth app with two corresponding Unity scripts which enable the communication between head tracker and Unity. In its simplest configuration, the plugin renders a dynamic binaural synthesis with the possibility to apply a directional pattern to the sound emitter. In order to take full advantage of the Unity environment, the plugin can be configured to calculate up to 20 reflections. All components are available as source code at the project repository [22] including precompiled plugin binaries for iOS and OSX, a sample scene for Unity and detailed installation instructions. Head tracker firmware, circuit diagram and additional information for building, programming and configuration are also available there. Detailed calibration instructions for the BNO080 are provided by the manufacturer [23].

3.1. Unity

The plugin binary must be placed in the Unity project folder in Assets/Plugins/(TARGET_PLATFORM) and the VAS_Unity_Spatializer must be chosen as *Spatializer Plugin* under *Project Settings/Audio*. Unity will search automatically for the version appropriate for the respective target platform. IRs must be placed within the *Assets/StreamingAssets* folder to ensure cross-platform file access. In the settings of any used audio source the checkbox *Spatialize* must be activated and *Spatialize Blend* should (usually) be set to 1. In order to load an IR set into a plugin instance, one of the C# *VasSpatConfig* scripts must be added to the Unity Game Object that contains the audio source. Three different versions are available:

- VasSpatConfigSimple
- VasSpatConfigManual
- VasSpatConfigAuto

All three implement the basic communication between the native audio plugin and C#. The latter one demonstrates the usage of Unity's physics engine in cooperation with the VAS Unity Spatializer for raytracing.

3.1.1. VasSpatConfigSimple

This script configures the plugin as a simple binaural renderer. Audio sources can be provided with a directional pattern. The script exposes seven variables in the Unity editor view:

- *IR set* has to be set to the IR filename including its extension but without its path. Supported file types are .txt files in the VAS format and Sofa files [8].
- *Global* denotes whether the IR set should be used as a global filter for all instances of the plugin.
- *Directivity damping* defines, whether the signal is damped linearly or logarithmically outside its full sound pressure area.
- *Horizontal source width* sets the area in degrees in the horizontal plane where the source is audible.
- *Horizontal full sound pressure* defines the area where the signal is emitted with full sound pressure in the horizontal plane.
- *Vertical source width* sets the directivity in degrees in the vertical plane where the source is audible.
- *Vertical full sound pressure* defines the area where the signal is emitted with full sound pressure in the vertical plane.

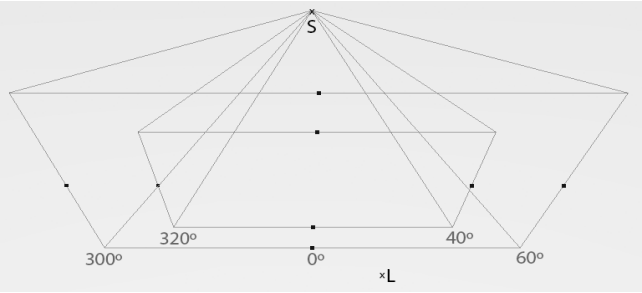

Fig. 1: Directional pattern example with a horizontal source width of 120° and full sound pressure level of 80°.

If the four latter parameters are set to 360°, the source behaves as an omnidirectional emitter. If the horizontal width parameter is, for example, set to 120° and the full horizontal

sound pressure to 80° (fig 1.), the source emits from 0° to 40° and from 320° to 0° with full energy. Within 40° to 60° and 330° to 340° the signal is gradually lowpass-filtered and attenuated. This is done either linearly or logarithmically (depending on the Directivity damping parameter). Bi-directional patterns can be achieved with two sources, emitting into opposite directions. Distance related damping is realized with Unity's build-in audio source features.

3.1.2. VasSpatConfigManual

With the VasSpatConfigManual script it is possible to add five binaural reflections for an Audio Source. The user has to manually create and place game objects in the Unity scene and drag them onto the public variable slots of the script. They determine the locations of the corresponding reflections. Public variables in addition to those of the first script are:

- *Reflection 1–5* are public variable slots for arbitrary game objects representing the position of the reflections.
- *Material stiffness* selects a material characteristic. Possible settings are low, middle and high.

3.1.3. VasSpatConfigAuto

This script uses raytracing to determine the locations of the reflections which are updated in real time. The number of rays is currently hardcoded to five rays. It has one additional variable:

- *Reflection order* defines, how many reflections per ray shall be calculated.

3.2. Head tracker

Precision and latency of the head tracking are among the most important quality assurance factors for an immersive auralization of virtual acoustic scenes, eg. dynamic binaural synthesis. For the entire audio path, refresh rates of 60 Hz and total delay times of 50 ms are considered acceptable [24]. Because head tracking is only the first part of this audio path, less latency at this point leaves more time for subsequent audio processing and is therefore desirable.
The practical aim of the presented solution is an interaction in which both source and listener are moving. Therefore, the minimal audible movement angles (MAMA) [25] are relevant factors. Strybel, Manligas and Perrott found a sensitive area for movement detection with 1° to 2° at a position of +40° and -40° azimuth and elevations below 80°. Outside of this area the MAMA increased to 3° to 10° [26]. Keeping this in mind the head tracker device should offer a minimum accuracy of less than 3°. The presented do-it-yourself (DIY) low cost head tracker device is made of an Host MCU - Adafruit Huzzah32 development board (ESP32), which supports Wifi and Bluetooth Low Energy (BLE) 4.2, and a Sparkfun BNO080 IMU sensor board connected via I2C. Apart from receiving the IMU data, the ESP32 handles the wireless communication and device management. For mobile use, the device has its own power supply, in the form of a 3.7 V LiPo battery, and a hardware on/off switch. The BNO080 provides orientation data with and without inclusion of the magnetometer. The internal sensor fusion uses the magnetometer for drift correction of the gyroscope. Thus, a smooth output (e. g. for games) or the most accurate output can be selected. The former setup can lead to the typical drift in long-term applications, the second setup to possible jumps during the correction process. However, with the help of a stabilization function (AR/VR stabilization), these jumps can be gradually corrected so that this sensor board is well suited for AR/VR tracking applications. When using the *Gaming Rotation Vector*, which does not use the magnetometer, a static/dynamic error of 1.5°/2.5° is specified. This complies with the Strybel et. al. [26] condition for the MAMA. In this setup, the drift of 0.5°/min can be balanced if AR/VR stabilization of this vector is selected [27].

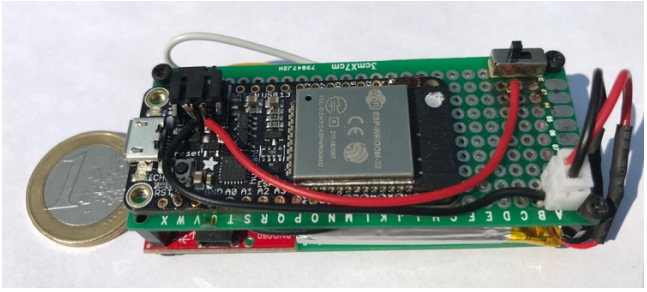

Fig. 2: VAS head tracker consisting of an Adafruit Huzzah32 development board, a Sparkfun BNO080 IMU sensor board and a mobile power supply, (LiPo battery, 3.7 V).

Azimuth and elevation do not describe the exact head position of the listener, as a possible lateral tilt of the head is not included. There is currently no HRTF or BRIR dataset that also shows lateral tilt of the head on a straight torso. The advanced Head-Above-Torso (HATO) HRTF database created by Brinkmann et. al. [28] also uses only azimuth and elevation. In the future, if there are data sets that support a lateral head tilt (Euler angle: roll), this angle can easily be provided by the BNO080. However, such data sets would either be very large, since for every possible head tilt a complete 360° data set would have to be present or would have very high computational costs due to the interpolation required for reduced data sets. Both cases are rather unfavorable in terms of resource allocation for mobile use and require further development work, both hardware and software.
Since both, the BNO080 and Unity work internally with quaternions with the y-axis (here elevation) is limited to ±90° [27, 29], no problems with the gimbal-lock are known.

3.3. Head tracker connection to Unity

VAS Head Tracker Connect is a standalone software, currently available for iOS and OSX, that serves as an intermediary between Unity and head tracker. It connects to the head tracker via Bluetooth and sends azimuth and elevation as open sound control (OSC) UDP packets to Unity. Two values can be set in the user interface:

- *OSC port number* must be set to match the receiver port in Unity
- *Headtracker ID* must be set to match the head tracker's name which is currently hardcoded to the head trackers firmware.

In Unity the package uOSC [30] has to be installed. The scripts uOSCServer and ReceiveHeadtrackerData have to be attached to the *Audio Listener* object.

4. Implementation details

The plugin performs a uniformly partitioned overlap add convolution. Length of the IRs is not limited (only by hardware resources). For implementation details about the underlying rendering engine, please refer to the publication and documentation about the VAS library [22] [31] .

4.1. Unity

The plugin is implemented with Unity's native audio plugin SDK in C++. Azimuth and elevation are automatically accessible in C++ for the direct path from source to listener. To be able to calculate angles and delay times for the reflections, their positions must be transmitted manually as float values from C#, along with the parameters for material characteristics. The maximum number of reflections is limited by the native plugin parameters which have to be declared and initialized in advance in the data structure of the plugin. The current implementation uses five rays. Reflections are calculated up to the 4th order, resulting in twenty reflections in total. The complete signal processing pipeline is illustrated in figure 3 below.

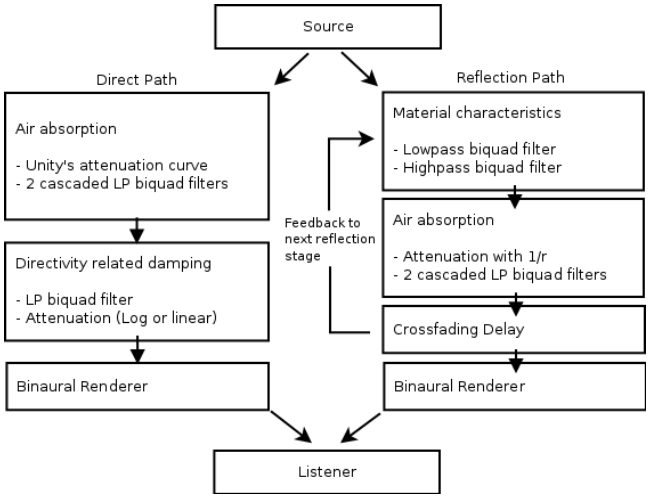

Fig. 3: The complete signal processing pipeline from source to listener.

Frequency related air absorption is approximated for 20° C and 20 % humidity with two cascaded biquad filters, similar to the illustrated filter curves in [32]. The directional pattern (and the corresponding damping) is applied to the direct path from sound emitter to receiver only. Reflections are considered to be omnidirectional. Their positions are calculated with the basic raytracing technique as described in [33]. As recommended by the authors, a predefined distribution pattern is used due to the small number of rays. Schröder suggests distributing them evenly across the source's surface [34]. In the presented solution, sound sources are considered as points without volume. Therefore, rays are evenly distributed within the source's directional pattern in the horizontal plane as shown in figure 4. Material characteristics are currently modelled in a simplified manner using a lowpass and a highpass biquad filter.

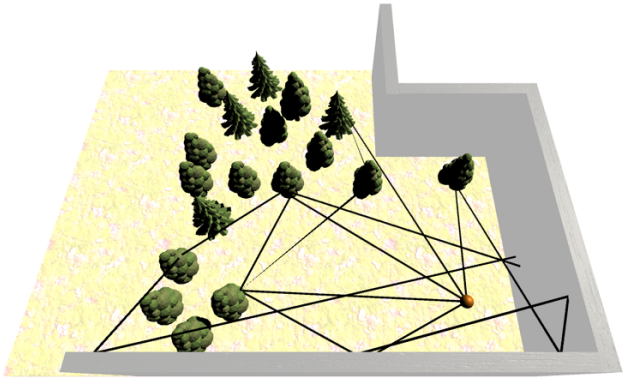

Fig. 4: Five evenly distributed rays with a directional width of 120° in a Unity scene with 4th order reflections.

Since outdoor environments usually have no ceiling, reflections of a higher order would no longer reach the listener if elevation angles are too large. Therefore, they are randomly varied by ±2° only. Only specular reflections are calculated. The possibility of diffuse reflections is currently ignored. The delay line is realized with two delays. In the moment the delay time changes, both the current and the target delay are performed and a crossfade with a length of 1024 samples is calculated from current to target delay. This makes large jumps possible without artefacts and prevents the typical pitch shifting effects of interpolated delays.

4.2. Head tracker

Overall latency for head tracking consists of the packet delivery time τ_{total}, which is the sum of packet transmission time τ_{trans} and the sensor device latency τ_{IMU}. With a polling time τ_{poll} at a sample rate of 200 Hz and a propagation delay τ_{prop} of 3.7 ms [23], τ_{IMU} is about 8.7 ms. The transmission time τ_{trans} is determined by the Bluetooth LE (BLE) transmission speed of the connection between the ESP32 and the mobile device. BLE uses channel hopping and so its communication consists of a consecutive number of connection events, organized at a specific connection interval τ_{ci}. After one τ_{ci} the frequency channel will be switched. The connection parameters are initially determined when a connection is established.

In the presented soft- and hardware solution the computer or smartphone acts as the master and finally defines τ_{ci} to the peripheral slave head tracker device. However, the peripheral device may ask for certain connection parameters. In case the suggested parameters do not meet the specifications of the central device, the request will be rejected. Depending on the operating system and the device generation, this minimum connection interval ranges from 7.5 ms to 30 ms and the maximum number of packets per connection interval N_{ci} may be 4, 6 or 7. Because BLE is a shared resource on mobile devices, the operating system can scale down N_{ci} as needed and increase τ_{ci} as needed. In central mode the connection parameters are determined by iOS. On the iPhone 8 test device, a τ_{ci} of 15 ms [35] and a N_{ci} of 7 is supported.

The data (azimuth and elevation as CSV) takes 8 bytes per orientation event, which fits easily into the default maximum transfer unit (MTU) size of 23 bytes which has a possible payload of 20 bytes, so only one packet per orientation event is needed. In line with our requirement of transmitting small data sizes within a strict time limit we need the smallest possible effective connection interval τ_{eci} and the highest N_{ci}. Therefore, the slave latency, the number of skipped connection intervals, N_{sl} is set to zero with $\tau_{\mathrm{eci}} = \tau_{\mathrm{ci}}$ [36] and the peripheral will send an update request to the central to ask for the smallest τ_{ci}.

5. Results

5.1. Latency

The predicted latency τ_{total} of the head tracking device in conjunction with the iPhone 8 test device should be:

$$\tau_{\mathrm{total}} = \tau_{\mathrm{trans}} + \tau_{\mathrm{IMU}}$$

$$\tau_{\mathrm{trans}} = \frac{\tau_{\mathrm{eci}}}{N_{\mathrm{pack}}} \qquad \tau_{\mathrm{eci}} = \tau_{\mathrm{ci}}(1 + N_{\mathrm{sl}})$$

$$\tau_{\mathrm{IMU}} = \tau_{\mathrm{poll}} + \tau_{\mathrm{prop}}$$

$$\tau_{\mathrm{total}} = \frac{15\ \mathrm{ms}}{7\ \mathrm{packets}} + 5\ \mathrm{ms} + 3.7\ \mathrm{ms} = \frac{10.8\ \mathrm{ms}}{\mathrm{packet}}$$

with $\tau_{\mathrm{ce}} < \frac{\tau_{\mathrm{eci}}}{N_{\mathrm{pack}}}$ to be able to use the maximum allowed number of packets.

With the capability of BLE to transmit 1 symbol in 1 µs [37], the time for a single connection event consisting of one communication cycle can be estimated as follows [36]:

1. Receive packet with a payload of 8 bytes and a maximum protocol overhead of 14 bytes [37] ($22 \times 8\ \mathrm{bit} \times 1\ \mu\mathrm{s}$),
2. Mandatory interframe space (150 µs),
3. Send acknowledge packet ($80\ \mathrm{bit} \times 1\ \mu\mathrm{s}$),
4. Mandatory interframe space (150 µs).

The sum meets the above condition for τ_{ce}: $0.536\ \mathrm{ms} < 2.14\ \mathrm{ms}$.

A theoretical number of possible N_{pack} per τ_{ci} can be calculated with:

$$N_{\mathrm{pack}} = \frac{\tau_{\mathrm{ci}}}{\tau_{\mathrm{ce}}}$$

Under real world conditions the influence of the bit error rate (BER), as demonstrated by Gomez, Demirkol and Paradells [38], the interference with other devices using the 2.4 GHz band, packet loss [36] and the restrictions of the central operating system leads to a much lower transmission rate and varying latencies. Especially the limitation of N_{pack} per τ_{ci} by the operating system means that no further data exchange takes place after reaching the maximum N_{pack} until the end of the τ_{ci}.

For the measurement 10,000 numbered packets were sent in ten iterations from the peripheral to the central device at a distance of 1 m and with a received signal strength indicator (RSSI) of -70 ± 5 dB. By using offline logging and a subsequent evaluation of timestamps and quantity of sent and received packets, an average packet loss of < 1.5 % was measured.

Packet loss leads to sporadically occurring higher latencies, which directly affects the update rates of the central device. Therefore, only an averaged update rate of appr. 5 ms at a IMU refresh rate of 200 Hz can be considered. With this setup, we estimate an average latency of the head tracking system of appr. 11 ms.

Using the *Best Latency* setting in Unity's audio preferences leads to a vector size of 256 samples under both operating systems (iOS in conjunction with an iPhone 8, and OSX) which corresponds to 5.8 ms (assuming a sample rate of 44.1 kHz). The dynamic filter change and the resulting crossfade between current and new angle causes an additional latency of 11.6 ms. The average OSC transmission time from the VAS Head Tracker Connect software to Unity was measured on a Macbook Pro 2018 (14.4 ms) and an iPhone 8 (18 ms).

At a refresh rate of 200 Hz this results in a total system latency of 42.2 ms on OSX and 46.2 ms on iOS, which meets the above-mentioned criterion [24]. The default latency setting, which leads to a vector size of 1024 samples on iOS, leads to a total latency of 63.6 ms. This value could still be considered acceptable, but savings in terms of CPU load are almost negligible (see table 1).

5.2. CPU load

CPU load was measured with Xcode Instruments on an iPhone 8. Partition and FFT size for the convolution were set to match the vector size. The percentage value in the right column is the CPU load for one core for one voice.

Vector size	CPU load (iPhone 8, one core)
256	9 %
1024	8 %

Tab. 1: CPU load on an iPhone 8.

A voice includes the playback of the audio source, Unity internal DSP (distance attenuation, doppler effect, mixer) and the complete signal processing of the plugin with 20 reflections. The head tracker was turned constantly, so that the convolution for the binaural synthesis (with an HRTF length of 256 samples) had to be carried out continuously for the current and the target angle.

6. Conclusion and outlook

The presented soft- and hardware is a powerful and easily configurable engine for rendering dynamic binaural synthesis in Unity. Besides real time calculation of HRTF based

binaural synthesis including up to 20 reflections it can process BRIRs of arbitrary length (only limited by hardware resources). The possibility to load an individual IR set for every plugin instance makes the VAS Unity Spatializer unique for the time being. This enables the user to, for example, equip different rooms with different BRIRs, preload several IR sets for listening tests or allow multiple users to experience one scene with different (for instance individualized) HRTFs simultaneously.

In Unity, the audio vector size is not as finely adjustable as in other environments, especially those focused primarily on audio (such as Pure Data or Max/MSP) where sizes as small as 16 samples are achievable. However, due to the low latency of the presented head tracker, the overall system latency is well within the requirements for dynamic binaural synthesis.

The Adafruit board can be configured as a Wifi access point. With the next firmware version, it will be possible to set all parameters (data format: e. g. euler angles or quaternions, connection type, sensor fusion method) via a static page hosted on the board. In order to enable use in environments without Bluetooth, data transmission via Wifi and OSC will be implemented. For the presented raytracing solution, physical principles have been simplified to ensure good usability and not to overuse hardware resources on the iOS platform. The focus was on outdoor environments with little reflections. A future release will enable the user to use more natural directional patterns, different material characteristics and a much larger number of reflections.

7. References

[1] HZK, "Auralisierung archäologischer Räume", [Online]. Available: https://www.interdisciplinary-laboratory.hu-berlin.de/de/content/analogspeicher-ii-auralisierung-archaologischer-raume. [Accessed 29. 08. 2019].

[2] FHNW, "FHNW Mysotis Garden," [Online]. Available: https://www.fhnw.ch/de/die-fhnw/hochschulen/ht/institute/institut-fuer-data-science/fhnw-myosotis-garden. [Accessed 22. 06. 2019].

[3] F. Stevens, D. T. Murphy, L. Savioja and V. Välimäki, "Modeling Sparsely Reflecting Outdoor Acoustic Scenes Using the Waveguide Web," *IEEE/ACM Transactions On Audio, Speech, and Language Processing,* p. pp. 1566–1578, 08. 2017.

[4] Oculus, "Oculus Spatializer," [Online]. Available: https://developer.oculus.com/downloads/package/oculus -spatializer-unity/. [Accessed 21. 06. 2019].

[5] Microsoft, "Microsoft Mixed Reality Documentation," [Online]. Available: https://docs.microsoft.com/en-us/windows/mixed-reality/spatial-sound-in-unity. [Accessed 21. 06. 2019].

[6] Google, "Resonance Audio," [Online]. Available: https://resonance-audio.github.io/resonance-audio/. [Accessed 21. 06. 2019].

[7] Steam Audio, "Git Repository Steam Audio," [Online]. Available: https://valvesoftware.github.io/steam-audio/downloads.html. [Accessed 27. 02. 2019].

[8] P. Majdak, Y. Iwaya, T. Carpentier, R. Nicol, M. Parmentier, A. Roginska, Y. Suzuki, K. Watanabe, H. Wierstorf, H. Ziegelwanger und M. Noisternig, „Spatially Oriented Format for Acoustics: A Data Exchange Format Representing Head-Related Transfer Functions," in *Proceedings of the 134th Convention of the Audio Engineering Society*, Rom, 2013.

[9] M. P. Jenny C. and C. Reuter, "SOFA Native Spatializer Plugin for Unity - Exchangeable HRTFs in Virtual Reality," in *Proceedings of the 144th Convention of the Audio Engineering Society*, Milan, 2018.

[10] M. Geier, J. Ahrens and S. Spors, "SoundScape Renderer," [Online]. Available: http://spatialaudio.net/ssr/. [Accessed 10. 02. 2019].

[11] M. Geier, J. Ahrens und S. Spors, „The SoundScape Renderer, A unified spatial audio reproduction framework for arbitrary rendering methods," in *124th AES Convention*, Amsterdam, 2008.

[12] LIMSI/CNRS, TKK/Department of Media Technology, IRCAM, "EVERTims," [Online]. Available: https://evertims.github.io. [Accessed 27. 02. 2019].

[13] M. Noisternig, B. Katz, S. Siltanen and L. Savioja, "Framework for real-time auralization in architectural acoustics," *Acta Acustica United with Acustica,* vol. 94, no. 6, p. 1000–1015, 2008.

[14] S. Laine, S. Siltanen, T. Lokki und L. Savioja, „Accelerated beam tracing algorithm," *Applied Acoustics,* Bd. 70, Nr. 1, p. 172–181, 2009.

[15] A. Franck, G. Costantini, C. Pike and F. M. Fazi, "An Open Realtime Binaural Synthesis Toolkit for Audio Research," in *Audio Eng. Soc. 144th Conv*, Milano, 2018.

[16] A. Baskind, "Hedrot," [Online]. Available: https://abaskind.github.io/hedrot/. [Accessed 21. 06. 2019].

[17] V. Manoukian, "EDTracker2," [Online]. Available: http://www.edtracker.org.uk. [Accessed 21. 06. 2019].

[18] D. Frie, "DIY Headtracker (Easy build, No drift, OpenSource)," [Online]. Available:

http://www.rcgroups.com/forums/showthread.php?t=1677559. [Accessed 21. 06. 2019].

[19] M. Romanov, P. Berghold, D. Rudrich, M. Zaunschirm, M. Frank and F. Zotter, "Implementation and Evaluation of a Low-cost Head-tracker for Binaural Synthesis.," in *142th AES Convention*, Berlin, 2017.

[20] R. Twomey, "bluetooth-headtracker," [Online]. Available: https://github.com/roberttwomey/bluetooth-headtracker/tree/d3df1d65b69e2e189bb189d9948c26a76d16ca1a. [Accessed 21. 06. 2019].

[21] S. Spors, "diy-low-cost-head-tracker-2," [Online]. Available: http://spatialaudio.net/diy-low-cost-head-tracker-2/. [Accessed 21. 06. 2019].

[22] T. Resch, „Git Repository VAS Library," [Online]. Available: https://github.com/funkerresch/vas_library. [Zugriff am 22 06 2019].

[23] Hillcrest Labs, "BNO080/BNO085 Sensor Calibration Procedure," [Online]. Available: https://www.hillcrestlabs.com/downloads/bno080-sensor-calibration-procedure. [Accessed 17. 06. 2019].

[24] M. Vorländer, Auralization, Heidelberg: Springer Berlin, 2008.

[25] D. W. Chandler and D. Grantham, "Minimum audible movement angle in the horizontal plane as a function of stimulus frequency and bandwidth, source azimuth, and velocity.," *J. Acoust. Soc. Am., Vol. 91, No. 3,* p. pp. 1624–1636, 03. 03. 1992.

[26] T. Strybel, C. L. Manligas and P. D. R. ., "Minimum audible movement angle as a function of the azimuth and elevation of the source," *Human Factors The Journal of the Human Factors and Ergonomics Society,* p. 267–275, 07. 1992.

[27] Hillcrest Labs, "BNO080/BNO085 Datasheet," [Online]. Available: https://www.hillcrestlabs.com/downloads/bno080-datasheet. [Accessed 17. 06. 2019].

[28] F. Brinkmann, A. Lindau, S. Weinzierl, S. v. d. Par, M. Müller-Trapet, R. Opdam and M. Vorländer, "A High Resolution and Full-Spherical Head-Related Transfer Function Database for Different Head-Above-Torso Orientations," *J. Audio Eng. Soc,* vol. 65, no. 10, p. 841–848, 2017.

[29] Unity Technologies, "Unity Documentation," [Online]. Available: https://docs.unity3d.com/Manual/QuaternionAndEulerRotationsInUnity.html. [Accessed 20. 08. 2019].

[30] Hecomi, „Git Repository uOSC," [Online]. Available: https://github.com/hecomi/uOSC. [Zugriff am 22 06 2019].

[31] C. B. S. W. T. Resch, „VAS – A cross platform C-library for efficient dynamic binaural synthesis on mobile devices," in *AES, International Conference on Headphone Technology*, San Francisco, 2019.

[32] L. S. M. K. J. Huopaniemi, „Modeling of reflections and air absorption in acoustical spaces — A digital filter desing," in *Proceedings of 1997 Workshop on Applications of Signal Processing to Audio and Acoustics*, New Paltz, NY, USA, 1997.

[33] U. P. S. L. Savioja, „Overview of geometrical room acoustic modeling techniques," *The Journal of the Acoustical Society of America 138,* p. 708–730, 2015.

[34] D. Schröder, „PHYSICALLY BASED REAL-TIME AURALIZATION OF INTERACTIVE VIRTUAL ENVIRONMENTS," *Aachener Beiträge zur technischen Akustik 11,* 2011.

[35] Apple, "Technical Q&A QA1931 Using the correct Bluetooth LE Advertising and Connection Parameters for a stable connection," [Online]. Available: https://developer.apple.com/library/archive/qa/qa1931/_index.html. [Accessed 05. 06. 2019].

[36] J. Afonso, A. Maio and R. Simoes, "Performance Evaluation of Bluetooth Low Energy for High Data Rate Body Area Networks," *Wireless Personal Communications,* p. 121–141, 09. 2016.

[37] Bluetooth SIG, Inc., "CS – Core Specification," [Online]. Available: https://www.bluetooth.org/docman/handlers/downloaddoc.ashx?doc_id=441541. [Accessed 23. 06. 2019].

[38] C. Gomez, I. Demirkol and J. Paradells, "Modeling the Maximum Throughput of Bluetooth Low Energy in an Error-Prone Link," *IEEE COMMUNICATIONS LETTERS,* vol. 15, no. 11, p. 1187–1190, 11. 2011.