RoomZ: Spatial panning plugin for dynamic RIR convolution auralisations

David Poirier-Quinot, Peter Stitt, Brian F. G. Katz

Audio Engineering Society

Conference Paper 19

Presented at the International Conference on Spatial and Immersive Audio
2023 August 23–25, Huddersfield, UK

RoomZ: Spatial panning plugin for dynamic auralisations based on RIR convolution

David Poirier-Quinot, Peter Stitt, and Brian F.G. Katz

Sorbonne Université, CNRS, Institut Jean Le Rond ∂'Alembert, UMR 7190, Paris, France

Correspondence should be addressed to David Poirier-Quinot
(david.poirier-quinot@sorbonne-universite.fr)

ABSTRACT

RoomZ is a plugin designed to create dynamic auralisations. Given a spatial grid of room impulse responses, it allows auditory source or receiver movement within that grid. The plugin supports fast multi-core partitioned convolution, custom navigation scenario creation and multi-channel auralisation (monaural, binaural, Ambisonic, etc.). Made available as a freeware for commercial and non-commercial uses, RoomZ is part of an ongoing research effort to make accessible the fruits of acoustic research to audio engineers working with digital audio workstations. This paper gives a general overview of how the plugin works as well as a detailed description of its main components.

1 Introduction

Auralisation is a technique to create a realistic auditory experience of a virtual environment. The technique has been widely used in a variety of applications, including architectural and environmental acoustics, noise control, and sound system design [1]. Auralisation can be achieved by convolving an anechoic sound with a Room Impulse Response (RIR). A RIR encodes the acoustic signature of a room, capturing the way sound waves travel from an acoustic source to a receiver microphone in a room.

Dynamic auralisations, in which the source or the receiver is moving, can be achieved by interpolating between RIRs during the rendering. Panning between two RIRs measured in the same room, for the same receiver position and two different source positions, can create the illusion that the source is moving between those positions [2]. Few tools exist to create such dynamic auralisation, which is very useful to create realistic virtual and augmented reality environments [3]. The RoomZ plugin is a tool designed to assist researchers and sound engineers alike in creating dynamic auralisation renderings.

RoomZ is released as a VST3 plugin, compatible with most Digital Audio Workstations (DAW), in a manner similar to the Anaglyph binaural rendering plugin developed by the same team [4]. Its primary function is to create dynamic auralisations, performing real-time convolution and interpolating between 1D or 2D grids

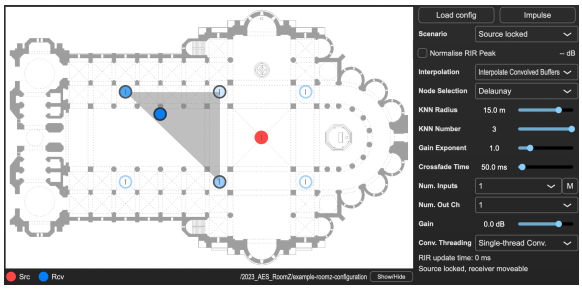

Fig. 1: The graphical user interface of the RoomZ plugin.

of RIRs to produce high-fidelity impressions of a moving source or receiver in a room. RoomZ is designed as a bridge between research and audio production so that the latter can easily exploit the RIRs acquired or generated by the former, regardless of the RIR simulation or recording method used. The RoomZ plugin is made available as freeware for commercial and non-commercial uses and can be downloaded from the project website[1]. The plugin's graphical interface is illustrated in Figure 1.

The paper is organized as follows: Section 2 presents existing convolution plugins, Section 3 gives an overview of the plugin, Section 4 further detail how its constituents work, Section 5 discusses future developments and uses, and Section 6 concludes this paper.

2 State of the art

Several plugins exist for auralisation based on real-time RIR convolution. Few of them are designed so that users can import their own RIRs [5, 6], even fewer support RIR panning for auralization of moving source or receiver. At present, the only other plugin identified that has been released that supports dynamic auralisation is the 6DoFconv plugin [7] part of the Sparta Suite [6].

The 6DoFconv plugin allows users to create auralisations of moving receivers using RIRs loaded in a SOFA file [8]. This plugin targets Ambisonic RIRs, including an Ambisonic rotation module after convolution. It also supports OSC messages to control receiver position and orientation.

Compared to 6DoFconv, RoomZ has a more advanced scenario definition, supporting moving receiver as well as multiple moving sources for complex scenes. Not based on the SOFA format, creating a new scenario only requires one to provide a set of WAV file RIRs and edit an XML file using a common text editor. The plugin's Graphical User Interface (GUI) shows the scene components (sources, receiver, etc.), drawn on top of a picture of the room in which the scenario takes place for improved readability. The plugin is not restricted to Ambisonic convolution and can be adapted to any type of channel layout. RoomZ also supports various nearest neighbour search algorithms, multi-thread convolution, and provides access to many adjustable parameters to support custom research and audio production scenarios better. Those features are described and illustrated in the following sections.

[1]RoomZ website: roomz.dalembert.upmc.fr.

3 Plugin overview

RoomZ is fundamentally a panning interface based on a 2D graphical mapping of source and receiver positions, where the weights are applied based on those geometrical distributions. Options for different panning laws are available to best suit the type of distribution and desired behaviour.

Using the plugin requires a RoomZ configuration XML file, comprising principally a description of source and receiver positions, with its accompanying set of WAV RIRs associated with each source/receiver combination defined (see Section 3.1). A room image JPG file is needed, providing the user with a visual reference, though no actual information in the image is used by the plugin. Once loaded, the RoomZ GUI shows the reference image with an overlay of source and receiver positions for the currently active **scenario** from the configuration file (see Section 3.2). Users can then move either sources or receivers depending on the selected scenario (see Section 3.3). From there, the plugin convolves the input stream with the relevant subset RIRs for dynamic auralisation.

3.1 Create a configuration

A RoomZ full configuration contains a picture, most typically a plan view of the room, a set of RIRs, and an XML configuration file providing information about how to select and map those RIRs for auralisation in the reference frame of the room image. The structure of the "example-configuration" folder used in the remainder of this paper is shown in Figure 2.

```
example-configuration
├── room.jpg
├── config.xml
└── rirs
    ├── 1.wav
    ├── 2.wav
    ├── ...
    └── 16.wav
```

Fig. 2: Example configuration folder structure.

`config.xml` is a file which provides information about the spatial relationship between source and receiver positions for different auralisation scenarios. The file used in the example-configuration is shown in the Appendix, where the "Source locked" scenario is the one loaded in Figure 1. For a given scenario, either the source or the receiver is marked as "locked", meaning its position is fixed. The other component can freely be moved in the room, *i.e.* within the grid of RIRs. The plugin supports multiple scenarios with multiple locked sources (playing simultaneously from different positions) within a scenario. Each will be matched to one of the channels of the plugin input provided by the DAW. For example, auralisation of a receiver moving around a duet singing in a room can be achieved by sending a stereo input to the plugin with the scenario "Multi-sources locked" in Section 7.

RIRs are expected as WAV files. There is no limitation on the number of channels of the RIRs, but for that imposed by the user's DAW. This allows the creation of monaural, binaural, Ambisonic, etc. auralisations. RIR sampling rate is expected to match the DAW's as the plugin does not perform re-sampling.

3.2 Load a configuration and a scenario

Loading a configuration can be achieved either via a dialog box opened from the plugin GUI or by dragging a configuration folder or file onto said GUI. The plugin will then verify the configuration's components, update the room displayed, and show the various components of the first scenario defined in the configuration file. As seen in Figure 1, potential positions of the moving component (receiver in this example), *i.e.* those positions for which a source-to-receiver RIR has been defined in the selected scenario, are depicted as empty circles. The current source and receiver positions (*i.e.* active RIRs) are indicated with filled circles.

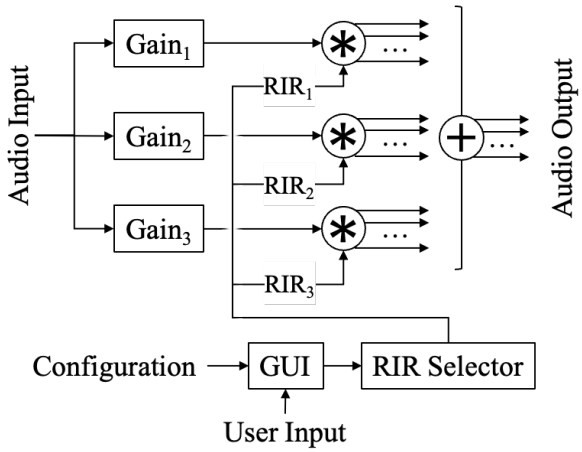

Fig. 3: Illustration of the RoomZ plugin architecture. The number of audio channels out of the convolutions is defined by the number of channels in the RIRs. The audio graph displayed is that of the "post RIR interpolation mode" described in Section 4.2. In the "pre RIR interpolation mode", the processing tracks associated to $Gain_2$ and $Gain_3$ do not exist, the weighted sum of the 3 RIRs happens before the convolution.

3.3 Moving source or receiver

Moving the non-locked element will trigger an update of the RIRs used for the convolution. Which and how many RIRs are used for the convolution depends on the current node selection and interpolation method, further detailed in Section 4. Those RIRs will be highlighted on the GUI as illustrated in Figure 1, with a color varying as a function of their contribution to the final auralisation.

4 Detailed components

RoomZ is implemented in C++, using the JUCE framework[2] for cross-platform compilation, GUI design, audio classes (buffers, filters, etc.), and standard VST features (automations, presets, I/O management, etc.). The architecture of the plugin's internal workings is illustrated in Figure 3.

[2]JUCE framework: juce.com.

4.1 Neighbouring RIRs selection

The RoomZ plugin supports two types of panning schemes to determine neighbouring RIRs and associated weights.

K Nearest neighbours selection: The K Nearest Neighbours (KNN) selection method consists in finding the K RIRs nearest to that of the moving component. The search can be limited to a circle around said component of radius R. Both K and R can be adjusted via the plugin GUI. The gain applied to each RIR selected is calculated using the inverse of the distance between the position of the moving component and the RIR's reference position. An exponent is applied to this gain that can be adjusted via the plugin GUI. The sum of the K gains is normalised to $= 1$ to avoid varying RIR grid spacings having an influence on output level. The method is illustrated in Figure 4(left). The KNN selection method is suitable for irregular, incomplete, and 1D grids (*i.e.* a linear arrangement). When $K = 1$, this method acts as a simple RIR selector, without any panning applied.

Delaunay triangle selection: The Delaunay selection method consists in using the 3 RIRs that correspond to the vertices of the Delaunay triangle in which the moving component is currently located. When using this method, the plugin will perform a spatial Delaunay triangulation on the RIR grid positions of the current scenario. If the triangulation cannot be performed (*e.g.* for 1D grids), the plugin defaults to the KNN selection method. The gain applied to each RIR is calculated using the **barycentric coordinates** of the current moving component within the Delaunay triangle. The contribution of each RIR is determined with a gain proportional to the area of the triangle formed between the two other RIRs in the Delaunay triangle and the current moving component position. The sum of the 3 gains is normalised to one. The method and how it behaves when the non-locked component moves is illustrated in Figure 4(right).

The main benefit of the Delaunay selection is that, by construction, it will reduce the contribution of any given RIR to zero before removing it from the auralisation. For most grids, this will reduce artefacts resulting from RIR interpolation during navigation. The KNN selection method, in addition to being intuitive to use, serves as a fallback when the Delaunay selection cannot apply.

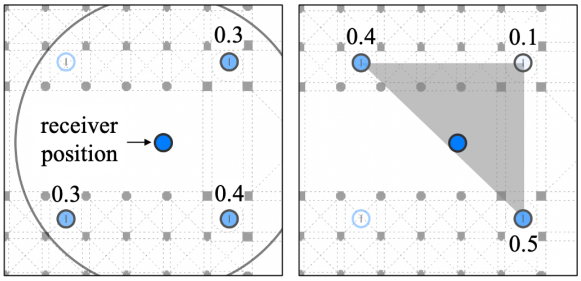

Fig. 4: Illustration of the neighbouring RIRs selection methods. (left) KNN: $K = 3$, the large grey circle corresponds to the KNN selection radius. (right) Delaunay: the grey triangle represents the active Delaunay triangle. Shown gain values are figurative.

4.2 Pre vs. post-convolution interpolation

This parameter allows one to choose if the interpolation between neighbouring RIRs is applied before or after the convolution. If the pre-convolution mode is selected, a new RIR will be generated from the selected neighbouring RIRs and loaded into a single convolver node. The benefit of this method is that it is less CPU demanding, requiring a single N channel RIR convolution as long as the moving component is static, or $2 \times N$ channel convolutions when it moves due to the crossfade mechanism. The drawback of this mode is that the interpolated RIR will have to be re-calculated and re-loaded into the convolver each time the moving component changes position. This can slow down the plugin and create zipper noise during the crossfade. This mode is best suited to static auralisations where neither source nor receiver is moving.

If the post-convolution mode is selected, the interpolation will be applied after the convolution, *i.e.* on the convolved audio streams resulting from each active neighbouring RIR. The benefit of this mode is that it only requires updating the RIRs loaded into the convolver nodes when one or more of the active neighbouring RIRs change. The drawback of this method is that it requires as many simultaneous convolutions as there are RIRs currently selected, multiplied by the number of channels of these RIRs. At the expense of CPU for audio fidelity, this mode is best suited to dynamic auralisations with moving components.

4.3 Additional component descriptions

Convolver: The internal convolution module builds upon the fast partitioned overlap-add convolution integrated to the JUCE framework (version 6). Compared to the original, it allows adjusting the cross-fade time allotted to swap between two RIRs within a given convolution node. When crossfading, it also copies the reverb tail of the current convolution engine to the new one to ensure continuity of the signal[3].

Crossfade time: Defines how long it takes to swap between two RIRs loaded in a given convolver node. It can be adjusted on the plugin GUI to balance responsiveness and rendering quality. Small values result in fast RIR updates, accommodating fast-moving sources and receivers. RIRs with longer decay times may require a larger cross-fade time to avoid zipper noises.

Number of inputs: This parameter defines the number of audio channels sent to the plugin for convolution. Available values range from 1 to the maximum number of locked components defined in the current scenario. Input channels can be independently muted via the plugin GUI to examine each track / locked component behaviour independently.

Number of outputs: Defines the number of audio channels of the RIR used for convolution. Available values range from 1 to the number of channels in the RIR set of the present scenario. This allows the user to adjust the CPU consumption of the auralisation, *e.g.* working with fewer channels for a more responsive workflow during preliminary test and setup.

Convolution threading: Allows the use of single or multiple threads for convolution processing. For machines that support it, multi-threaded convolution will run the processing for later RIR partitions on background threads. The first partitions are always rendered on the real-time audio thread. This increases the maximum auralisation complexity (number of sources, channels, etc.) the plugin can handle before CPU saturation.

[3]This is not the case with the default JUCE convolution, which is intended for use with IRs that are generally shorter than the cross-fade, allowing time for the whole reverb tail to build up before finishing the cross-fade to the new engine.

5 Future work

Future work will include improving the ease-of-use and flexibility of RoomZ for both research and creative work. The most obvious extension is to allow both source and receiver to be moved simultaneously. This requires significantly more data and quickly increases the complexity of the user interface if the source and receiver grids are not homogeneous.

A “preview” rendering mode is planned that will allow the user to render a truncated version of the RIR when mixing, reducing the CPU load. Truncated reverberation tails could be complemented with an algorithmic reverberator (*e.g.* feedback delay network) matched to the estimated reverberation time of the RIR. Upon final export processing, the full RIR would be used.

The plugin export could also be improved to provide maximum fidelity for off-line auralisations, without CPU saturation limitations. The current rendering is performed as a “live" render in real-time. A future option could be given to the user to block the rendering until all background tasks (RIR loading/interpolation) is completed: an **off-line** rendering mode where latency is not an issue.

To further improve ease of uptake, a web-based interface for creating the configuration XML scripts is planned. This will be complemented by the addition of SOFA support to avoid data duplication. Furthermore, it is intended to create a database of configurations that can be downloaded by users spanning a large variety of architectural and outdoor spaces. The intent is to allow sound engineers to be able to find an acoustic space that closely matches their needs without the necessity of individually making large numbers of impulse response recordings or high-definition simulations.

The loading of configurations is currently only possible through the GUI. Future updates could allow for configuration loading to be controlled via OSC, without direct user action. This could be particularly useful for scientists intending to run experiments or for virtual reality auralisations.

At present, RoomZ is intended to give an accurate reproduction of “real” spaces through the use of their measured/simulated RIRs. There may, however, be instances where it may be creatively desirable to be able to modify/alter the decay of the RIR dataset, to better

adapt to the artistic needs of the rendering, while maintaining the fine temporal reflection structure details of the room's acoustic signature. A first approximation for such manipulation would be a frequency-band reverberation slope adjustment [9], extending RoomZ to be a more creative sound design tool.

6 Conclusion

The RoomZ plugin has been conceived to create realistic dynamic auralisations of acoustic spaces. It is freely available for research, creative, and commercial creation. It uses existing sets of RIRs (either recorded or simulated), presenting users with an interface where they can move either source or receiver positions dynamically and in real-time. It supports the rendering of multiple sources in a single instance of the plugin. The processing is based on partitioned convolution interpolating between a subset of the RIR set, selected by proximity, with a choice of rendering options to balance CPU load and audio fidelity. The plugin is format agnostic, with format-specific elements being implemented either upstream (*i.e.* RIR generation/acquisition) or downstream (*e.g.* Ambisonic rotation and decoding).

RoomZ can be used by both researchers and sound engineers to create either rendered auralisations or live experiences. It is presently being used to create a fictional audio narrative around the soundscapes and acoustics of Notre-Dame de Paris Cathedral across the centuries, accompanying Victor Hugo while he writes his famous novel[4]. It is intended that the plugin fosters discussion between researchers and sound engineers in order to provide insight into future developments of methods for real-time auralisation.

7 Acknowledgments

RoomZ was developed within the European Union's Joint Programming Initiative on Cultural Heritage project PHE (The Past Has Ears, Grant No. 20-JPIC-0002-FS, phe.pasthasears.eu) and the French project PHEND (The Past Has Ears at Notre-Dame, Grant No. ANR-20-CE38-0014, phend.pasthasears.eu), including participation by the Conservatoire National Supérieur de Musique et de Danse de Paris (CNSMDP).

[4]Looking for Notre-Dame: lookingfornotredame.pasthasears.eu.

References

[1] Vorländer, M., **Auralization: Fundamentals of acoustics, modelling, simulation, algorithms and acoustic virtual reality**, Springer Nature, 2020.

[2] Savioja, L., Huopaniemi, J., Lokki, T., and Väänänen, R., "Creating interactive virtual acoustic environments," **J Audio Engineering Society**, 47(9), pp. 675–705, 1999.

[3] McKenzie, T., Schlecht, S. J., and Pulkki, V., "Auralisation of the transition between coupled rooms," in **Immersive and 3D Audio Conf.**, pp. 1–9, 2021.

[4] Poirier-Quinot, D. and Katz, B. F. G., "The Anaglyph binaural audio engine," in **Audio Engineering Society Conf.**, pp. EB431:1–4, Milan, 2018.

[5] Kronlachner, M., "Plug-in suite for mastering the production and playback in surround sound and ambisonics," **Contributions to the AES Student Design Competition**, pp. 1–5, 2014.

[6] McCormack, L. and Politis, A., "SPARTA & COMPASS: Real-time implementations of linear and parametric spatial audio reproduction and processing methods," in **Audio Engineering Society Conf.**, pp. 1–12, 2019.

[7] McKenzie, T., Meyer-Kahlen, N., Daugintis, R., McCormack, L., Schlecht, S. J., Pulkki, V., et al., "Perceptually informed interpolation and rendering of spatial room impulse responses for room transitions," in **Intl Congress on Acoustics**, pp. 1–11, 2022.

[8] Majdak, P., Iwaya, Y., Carpentier, T., Nicol, R., Parmentier, M., Roginska, A., Suzuki, Y., Watanabe, K., Wierstorf, H., Ziegelwanger, H., et al., "Spatially oriented format for acoustics: A data exchange format representing head-related transfer functions," in **Audio Engineering Society Conf.**, pp. 1–11, 2013.

[9] Cabrera, D., Lee, D., Yadav, M., and Martens, W. L., "Decay envelope manipulation of room impulse responses: Techniques for auralization and sonification," in **Acoustics Conf.**, pp. 1–5, 2011.

Appendix: Example XML configuration file.

```
<root>

  <!-- The data field is used to descirbe the configuration, displayed on the plugin UI. -->
  <info data="Basilique Sainte-Anne-de-Beaupré"/>

  <!-- The room image width value is used for pixel to scene unit conversion. origin_i indicates the
  ↪  position relative to which element coordinates are defined. -->
  <room width="95" origin_x="0" origin_y="0" origin_z="0"/>

  <!-- In this scenario, the source is locked and the receiver is free to move. -->
  <scenario locked="source" name="Source locked">

    <!-- Defining source position in the room. [src_x, src_y, src_z] corresponds to the Cartesian
    ↪  coordinates of the source. [rot_x, rot_y, rot_z] corresponds to its orientation (Euler angles).
    ↪  Orientation values are only displayed in the UI, not taken into account during DSP processing. -->
    <source src_x="59.7" src_y="30.4" src_z="0" rot_x="0" rot_y="90" rot_z="0"/>

    <!-- Defining receiver's potential positions in the room, along with the associated source-to-receiver
    ↪  RIRs to use for auralisation when the receiver is at that position. -->
    <receivers>
      <receiver file_name="rirs/01.wav" rcv_x="28" rcv_y="20" rcv_z="0" rot_x="0" rot_y="90" rot_z="0"/>
      <receiver file_name="rirs/01.wav" rcv_x="50" rcv_y="20" rcv_z="0" rot_x="0" rot_y="90" rot_z="0"/>
      <receiver file_name="rirs/02.wav" rcv_x="70" rcv_y="20" rcv_z="0" rot_x="0" rot_y="90" rot_z="0"/>
      <receiver file_name="rirs/03.wav" rcv_x="28" rcv_y="41" rcv_z="0" rot_x="0" rot_y="90" rot_z="0"/>
      <receiver file_name="rirs/03.wav" rcv_x="50" rcv_y="41" rcv_z="0" rot_x="0" rot_y="90" rot_z="0"/>
      <receiver file_name="rirs/04.wav" rcv_x="70" rcv_y="41" rcv_z="0" rot_x="0" rot_y="90" rot_z="0"/>
    </receivers>

  </scenario>

  <!-- In this scenario, the receiver is locked and the source is free to move. -->
  <scenario locked="receiver" name="Receiver locked">

    <!-- Defining receiver position in the room -->
    <receiver rcv_x="59.7" rcv_y="30.4" rcv_z="0" rot_x="0" rot_y="90" rot_z="0"/>

    <!-- Defining source's potential positions in the room. -->
    <sources>
      <source file_name="rirs/05.wav" src_x="50" src_y="20" src_z="0" rot_x="0" rot_y="90" rot_z="0"/>
      <source file_name="rirs/06.wav" src_x="70" src_y="20" src_z="0" rot_x="0" rot_y="90" rot_z="0"/>
      <source file_name="rirs/07.wav" src_x="50" src_y="40" src_z="0" rot_x="0" rot_y="90" rot_z="0"/>
      <source file_name="rirs/08.wav" src_x="70" src_y="40" src_z="0" rot_x="0" rot_y="90" rot_z="0"/>
    </sources>

  </scenario>

  <!-- In this scenario, two locked sources are defined alongside one free to move receiver. The plugin
  ↪  expects a 2 channels input, one for each source. -->
  <scenario locked="source" name="Multi-sources locked">

    <!-- Defining source 1 position in the room. -->
    <source_1 src_x="60" src_y="50" src_z="0" rot_x="0" rot_y="90" rot_z="0"/>

    <!-- Defining source 2 position in the room. -->
    <source_2 src_x="60" src_y="12" src_z="0" rot_x="0" rot_y="90" rot_z="0"/>

    <!-- Defining receiver's potential positions in the room. file_name_i corresponds to the
    ↪  source_i-to-receiver RIR. -->
    <receivers>
      <receiver file_name_1="rirs/09.wav" file_name_2="rirs/13.wav" rcv_x="50" rcv_y="20" rcv_z="0"
      ↪  rot_x="0" rot_y="90" rot_z="0"/>
      <receiver file_name_1="rirs/10.wav" file_name_2="rirs/14.wav" rcv_x="70" rcv_y="20" rcv_z="0"
      ↪  rot_x="0" rot_y="90" rot_z="0"/>
      <receiver file_name_1="rirs/11.wav" file_name_2="rirs/15.wav" rcv_x="50" rcv_y="40" rcv_z="0"
      ↪  rot_x="0" rot_y="90" rot_z="0"/>
      <receiver file_name_1="rirs/12.wav" file_name_2="rirs/16.wav" rcv_x="70" rcv_y="40" rcv_z="0"
      ↪  rot_x="0" rot_y="90" rot_z="0"/>
    </receivers>

  </scenario>

</root>
```