Didactic Platform to Immersive Sound Decoding for Binaural or Multichannel Speakers

Almeida, A.[1], Preto Paulo, J.[12]

***Abstract*—** The concept of immersive sound/3D sound has gained great importance with the advent of VR/AR and also with the transmission of live television programs. Reproduction of this type of material requires coding/decoding algorithms for headphones or loudspeaker arrays with different geometries. Listening to spatial audio through headphones is very convenient, only requiring the spatialization codec and headphones. However, listening through a multi-channel speaker system has far greater realism. Thus, it is common practice to build listening rooms for immersive sound using dozens of speakers that allow research & development work to be carried out. Perceptual tests are carried out with a sample of subjects to fine-tune the entire system and choose the codec that best adapts to each space. The work presented consists of an intuitive application to perform auditory perception tests in order to evaluate the different codecs based on the Unity3D game platform to create the visual environment and on the audio production application Reaper to support spatialization audio codecs. The tested codecs are VBAP (Vector Based Amplitude Panning) and Ambisonic. The use of this application greatly facilitates all the procedures/logistics of perceptual tests, greatly reducing the necessary preparation time. It also allows the use of several simultaneous sound sources. The application is being updated to use VR oculus, in order to be used in therapeutic health plans using sound for patients with cognitive problems and stressful situations, temporary or permanent, in the Immersive Sound Room of the Audio and Acoustics of ISEL (https://acusticaudiolab.isel.pt).

***Keywords*—** Unity3D platform, Auralization, Binaural, Multichannel Speaker, VBAP, Ambisonic.

I. Introduction

Surround sound has quickly become a consumer 'must have' in the audio world, due, in the main part, to the advent of the Augmented Reality (AR)/Virtual Reality (VR) products like the Oculus Rift and Playstation VR, the computer gaming industry and 3D movies (Dolby Atmos). It is generally taken to mean a system that creates a sound field that surrounds the listener. Or, to be put another way, it is trying to recreate the illusion of the 'you are there' experience. This is in contrast to the stereophonic reproduction that has been the standard for many years, which creates a 'they are here' illusion (Glasgal, 2003c). The direction that the surround sound industry has taken, when referring to format and speaker layout, has depended, to some extent, on which system the technology has been used for.

Soundfield recording and reproduction adopts physically-based models to analyze and synthesize acoustic wave fields [1–4]. It is normally designed to work with many microphones and a multi-channel speaker setup. With conventional stereo techniques, creating an illusion of location for a sound source is limited to the space between the left and right speakers. However, with more channels included and advanced signal processing techniques adopted, current soundfield reproduction systems can create a 3D (full-sphere) sound field within an extended region of space.

Dolby Atmos (Hollywood, CA, USA) [5] and Auro 3D (Mol, Belgium) [6] are two well-known examples in this area, mainly used in commercial cinema and home theater applications.

II. Metodology

This project was conceived to allow the handling of the DAW (Digital Audio Workstation) Reaper through the Unity game engine. On this platform, it will be possible to control the Reaper through several commands with the OSC protocol, without having to interact directly with the DAW. The data/command flow of this application is based on the following architecture presented below.

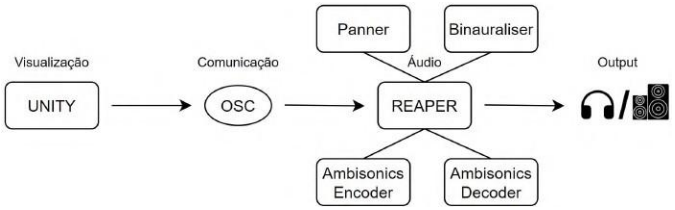


Fig. 1 Data flow

It is seen, in the previous figure, that the communication will be unidirectional, without the Reaper being able to send messages to Unity. Note that Reaper has communication for four plugins that will be assigned to several audio tracks, in order to process the audio spatially. After this processed

André Almeida, ISEL-Instituto Superior de Engenharia de Lisboa, Portugal, Email: a45120@alunos.isel.pt

Joel Preto Paulo, ISEL-Instituto Superior de Engenharia de Lisboa, LAA-Audio and Acoustics Laboratory, Portugal, Email: , joel.paulo@isel.pt, acusticaudiolab@isel.pt

audio, the output is transmitted either to headphones or sound monitors (depending on the mode used in the application).

However, regarding the visualization (Unity), it will have a more complex internal environment, so the following class diagram was created, where the connections and dependencies used in the application are presented. It can be seen that the Source and Speaker classes are the only ones that were developed and that are not MonoBehaviour, since each of these represents a sound source or an audio monitor, merely containing the pertinent information of each of the objects. Note that the OSCTransmitter, VideoPlayer and MonoBehaviour classes were imported for application development.

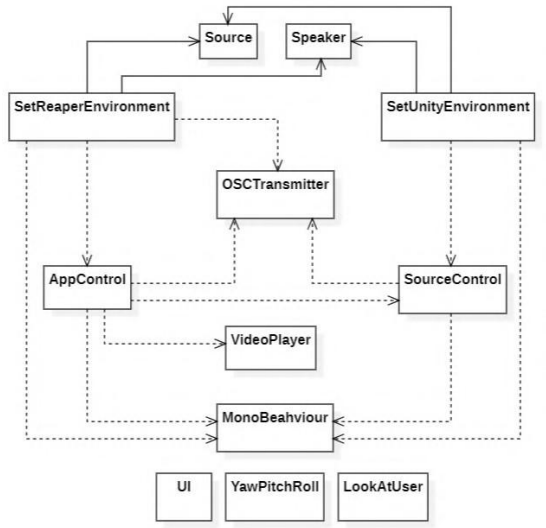


Fig. 2 Class diagram

The interaction between Unity and Reaper aims to test different application modes. These modes can be of two types:

• Vector Base Amplitude Panning (VBAP): Method of positioning virtual sources in arbitrary directions using a multi-output configuration (speakers). This number of outputs can be arbitrary, as can their positions in 2D or 3D configurations.

• Ambisonics: Format of surround sound in a sphere (in addition to the horizontal plane, it covers the sound sources above and below, ie the vertical plane). It is a method for recording, mixing and playing 360º three-dimensional audio.

To test these two modes there will be two scenarios in which the application can run:

• Virtual scenario: The user can use a scenario created in Unity, for example a studio, to test different combinations of inputs and outputs;

• 360º Video: The user will be able to watch an immersive 360º video, which will be complemented by the audio in question, for example a recording of a concert (image and audio);

It is also available to the user the option to turn on or off the binaural mode (which can be useful both in VBAP and in ambisonics). This mode aims to simulate human hearing.

Binaural hearing, together with frequency filtering, makes it possible to determine the direction of sound origin. It is a sound recording and reproduction technique in which, with only two microphones, it is possible to create an ambient sound effect.

III. Development

A. Open Sound Control (OSC)

Open Sound Control (OSC) is a protocol for networked sound synthesizers, computers and other multimedia devices. It is based on a language that performs pattern matching, specifying the various recipients of a single message.

Open Sound Control includes a pattern matching language for specifying multiple recipients of a single message. This protocol was developed enabling maximum interoperability and therefore provides a solution for communication between applications and devices, locally or on a network.

This protocol was then used to allow communication between both applications, sending messages from Unity that will control Reaper parameters. To send these messages, the file "MyOSC.ReaperOSC" was used, which contains the actions and addresses of the different commands that can be executed in Reaper through the aforementioned communication. Each line is a description of the action in capital letters, followed by various OSC message patterns (addresses). You can add, remove, or change patterns, but not action descriptions. Addresses are the messages that Reaper will be able to send and receive with the OSC protocol.

Assuming that it is necessary to deactivate a plugin for a certain track, one of the addresses given in the file is "b/track/@/fx/@/bypass", where "b" symbolizes that the value of that action is a boolean and where "@" symbolizes the number that corresponds to the intended track or plugin. Therefore, to deactivate the first plugin of the first track, the address will be "/track/1/fx/1/bypass" and the code to implement will be the following.

```
// Desativar o plugin
OSCMessage message = new OSCMessage("/track/1/fx/1/bypass");
message.AddValue(OSCValue.Float(0));
transmitter.Send(message);


// Ativar o plugin
OSCMessage message = new OSCMessage("/track/1/fx/1/bypass");
message.AddValue(OSCValue.Float(1));
transmitter.Send(message);
```

To send a message to multiple plugins at the same time, use the code below, where the first two are activated and the last two are deactivated.

```
OSCMessage message = new OSCMessage("/track/1/fx/1,2,3,4/bypass");


message.AddValue(OSCValue.Float(1));
message.AddValue(OSCValue.Float(1));
message.AddValue(OSCValue.Float(0));
message.AddValue(OSCValue.Float(0));


transmitter.Send(message);
```

For more complete information about the addresses, read the beginning of the "MyOSC.ReaperOSC" file.

It should be noted that, although it is possible to send and receive messages from Unity, the messages received are not through a request for information (such as calling a function). To receive information from Reaper, it is necessary to bind a function to the address where the information is to be received. When the parameter in question is changed in Reaper, a message will be sent to the linked address with the new parameter value.

```
private OSCReceiver receiver;


void CreateReceiver()
{
    receiver = gameObject.AddComponent<OSCReceiver>();
    receiver.LocalPort = 7001;
    receiver.Bind("actionAddress", functionToBind);
}
```

B. User Interface (UI)

Canvas

To present the application to the user, a UI canvas was created, containing information such as the number of inputs/sources, outputs/speakers, the application's audio mode and possible commands.

Two buttons were created, one in each upper corner, so that when "minimizing" the information text, the user can have a clear view of the application. The button for information was implemented in the upper left corner in the shape of an i, so the button for the application's commands is present in the upper right corner with the symbol of a list (hamburger menu).

It should be noted that, to represent the inputs, a treble clef was used and, for the outputs, an audio monitor was used.

Fig. 3 Menus: Information and Commands

Scenes

To represent the virtual scenario, a model of the audio and acoustic laboratory of the Instituto Superior de Engenharia de Lisboa was used, where several tests can be carried out with different arrangements of inputs and outputs. This model may also have acoustic panels, which can be moved in order to diversify the tests carried out in this scenario.

Fig. 4 Model of the audio and acoustics laboratory. (a) Inside and (b) Outside

On the other hand, in order to reproduce the 360º video, a sphere was created where the normals of its material were inverted (), that is, when the Video Player component is added, the video given as source is reproduced inside the sphere, producing the illusion that the user is inside the video scenario.

When applying the new material, despite having inverted normals, the outside of the sphere continues to play the video as if the material were the standard.

Fig. 5 Sphere for 360º video playback. (a) Inside and (b) Outside

Both in one scenario and in the other, it is possible to present the inputs and outputs within them (of the model and of the sphere), as it is possible to notice in the image (a) of the previous figure.

C. Creating the Environment

When the application is started, the necessary conditions are created so that the user can use it properly. To this end, using Unity and the code implemented in it, the environment

in which the user will interact is defined. It is necessary that this environment be the same within both platforms, so that when the user performs any action in Unity, it is reflected in Reaper.

To certify this synchronization, whenever the application is started, Unity will send a series of OSC messages to Reaper, defining the plugin parameters based on two JSON files, "sources. json" and "outputs.json", which store information about sound sources (inputs/sources) and columns (outputs/speakers). These files can be imported or exported from Reaper plugins.

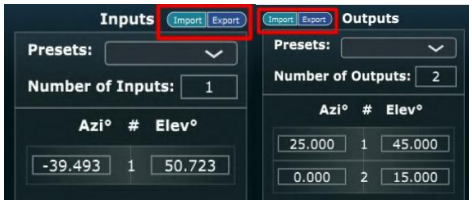


Fig. 6 Inport/Export JSON. (a) Inputs (b) Outputs

A JSON file that stores information about, for example, sound sources, will have the following architecture shown below, where within the Elements node all sound sources are presented with their attributes, highlighting Azimuth and Elevation.

```
{
  "Name": "SPARTA Panner source/loudspeaker directions.",
  "Description": "This configuration file was created with the SPARTA Panner v1.5.8 plug-in. 17
      Feb 2023 10:52:15am",
  "GenericLayout": {
    "Name": "Source/Loudspeaker Directions",
    "Elements": [
      {
        "Azimuth": -1.5,
        "Elevation": 0.0,
        "Radius": 1.0,
        "IsImaginary": false,
        "Channel": 1,
        "Gain": 1.0
      },
    ]
  }
}
```

Unity

To define the environment in Unity, the two JSON files mentioned above are analyzed, saving the inputs and outputs in two lists. To facilitate the creation of these lists, two classes were created, Source and Speaker, which have three public attributes, Azimuth, Elevation and Radius. With both lists created and their elements added, it is possible to create them in the scene.

Reaper

With the environment defined in Unity, the environment is now defined in Reaper. To do so, the JSON files are analysed, as before, and lists of inputs and outputs are constructed. As in Unity, after defining the lists, it is possible to start defining the environment. But, firstly, it is a priority to mention which plugins are used in this project, which address VBAP, ambisonics and binaural modes:

- VBAP: Sparta Panner;
- Ambisonics: Sparta AmbiENC e Sparta AmbiDEC;
- Binaural:
- Sparta Binauraliser ;

All the necessary plugin parameters are then defined when starting the application.

Since it will be allowed to change between VBAP and ambisonics modes and to enable/disable the binaural mode, it is necessary to define the parameters of the four plugins right from the beginning, so that the application does not need to define them every time the mode is changed. Two functions were implemented to achieve this goal:

- setVBAP();
- setAmbisonics();

Both functions specify the parameter values of the two plugins that correspond to each mode.

To create the environment in Reaper, three value conversion functions were also implemented. These functions are necessary since each plugin parameter receives a value in the range [0, 1], while the values used in Unity have different ranges. Therefore, before sending OSC messages with the new values, it is necessary to convert them to the requested range.

After defining the parameters of the inputs, those of the outputs are now defined, maintaining the same logic, where first the desired number of columns is sent and only then the azimuth and elevation values for each one. The only difference, other than columns are used instead of fonts, are the IDs entered in the address.

At the end of establishing the initial values of the fonts and columns, the script where this processing is carried out is destroyed.

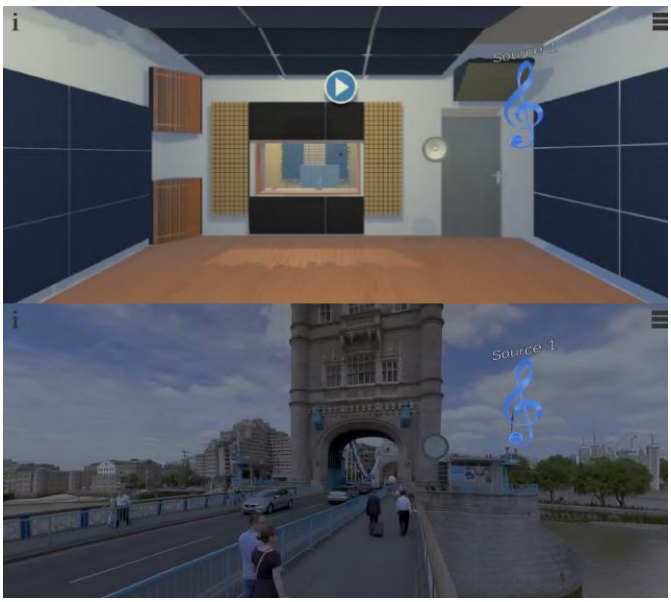
D. System Control

Sound Sources (Inputs)

To control each sound source, a new script, "SourceControl.cs", was created and assigned to the prefab of the sources (treble clef), that is, whenever a new source is instantiated, it is already created with the controller. In this script there is the public attribute id, which has its value changed when a new instance is created (createSources() function in the "SetUnityEnvironment.cs" and "AppControl.cs" scripts, so that it is possible to have several sources with different ids , allowing you to control one source at a time without influencing the others.

At the end, two functions that were implemented in this script are used - calculateAngles() and sendSourcePosition(). In the sendSourcePosition() function, a message is sent via the OSC protocol with the azimuth and elevation values to control the sound source that is being dragged. It should be noted that each sound source has two parameters to which a value must be assigned (azimuth and elevation) in the three plugins in question, VBAP, ambisonics and binaural.

With these two built-in functions and with the sending of the position values of each sound source through OSC messages, the results shown in the figure below are obtained, where a source was positioned in the position where azimuth = −35 and elevation = 10 through the Unity controls and the same happened to the font position in the Reaper plugins.



Application

To control the application, whether in Unity or in Reaper through Unity, a new script was created, "AppControl.cs". This script will control the following features: • Audio played by Reaper; • Used modes (VBAP, Ambisonics and binaural); • Virtual reality; • Visualization; • Yaw, Pitch and Roll;

Fig. 7 Sound source position. (a) Unity - Audio and Acoustics Laboratory, (b) Unity - 360º Video and (c) Reaper

Audio

To control the audio reproduced by Reaper there are two basic controls:

• P key: Play/Pause. Controls Reaper audio playback. If the audio is being reproduced and the key is pressed again, the reproduction is stopped, if it is stopped, the audio is reproduced. In addition to controlling the audio, the video used in the 360º scenario is also controlled, where the video starts playing at the same time as the audio and, if you want to pause the audio, the video will also pause.

If the user is equipped with virtual reality glasses or uses the camera controls,

he will be able to look at a button with the image shown below in order to play the audio.

In figure 8 you can see a white circular progress bar, which moves clockwise, and when it reaches 0, the circle is complete and disappears, just like the button. This implementation is carried out in the startApp() function, which is only carried out if the hasStarted variable is False.

Fig. 7 Location of play button and timer

The key: Stop. Stops the 360º audio and video, returning to the beginning of both when the P key is pressed.

Number keys 1-9: Mute. When one of the numbers 1 to 9 is clicked, the track corresponding to that number is muted. If the selected track is already muted, then it is active again.

To know the state of the track in question, the tracksMuted dictionary presented in the previous code excerpt was implemented. This dictionary contains 9 elements, where the keys are integers from 1 to 9 and the value of each is a boolean to indicate the status of the track, true if it is muted and false if it is playing audio.

One sound source will have one instance of the plugins while other sources will have other instances of the same plugins. In this way, to show which sound sources are active or deactivated, the material of the treble clef was changed, changing it to gray if it is deactivated or blue if it is active.

Fig. 8 Source muted (left)– Unity

When viewing the effect that this action had on Reaper, it can be seen that track 2 was also muted (active red button with the letter "M").

Fig. 9 Source muted - Reaper

Note that in this example only two fonts are instantiated in Unity, while in Reaper there are 9. If a numerical key is clicked and it corresponds to one of the fonts that is not instantiated, this action will have no effect. For this, it is necessary to instantiate more sound sources.

Application Mode

To change the application's operating mode between VBAP and ambisonics, press the M key,

changing which plugins will be active. For this, two functions were created:

• ChangeToVBAP(), ChangeToAmbisonics()

To activate or deactivate the binaural mode, the B key was defined. In this way, the fourth plugin is activated or deactivated, depending on its current status. In addition to this status change, the information text and the isBinaural variable are also updated. The code snippet below shows only the binaural mode activation function.

Yaw, Pitch e Roll

To control yaw, pitch and roll values in plugins that use these parameters, the sendYawPitchRoll() function was created, used at each frame (Update() method). In this function, values are assigned to the yaw, pitch and roll variables through the rotation of the user's camera.

This rotation takes place by manual camera control or by rotating the VR device. The three variables will correspond to the three axes (x, y, and z) as follows: • Yaw: Y axis; • Pitch: X axis; • Roll : Z axis;

After defining the variables, an OSC message is sent to change the values of the parameters in question. For this message, three value conversion functions are used, convertYaw(float value), convertPitch(float value) and convertRoll(float value). These functions apply a conversion from the range in which the rotation in angles of the camera on each axis is expressed to the range [0, 1].

IV. Conclusions and Future Work

This project focused on the development of a virtual reality application for testing and audio control through Unity and Reaper platforms. With the use of these it was possible to use several scenarios and modes (active plugins) to test different factors.

From the data flow presented in the introduction (figure 1.1), it is concluded that there is no communication from Reaper to Unity, defining it as a unilateral communication. Therefore, the general procedure of the application involves the user performing certain actions that will have an impact on the audio control platform.

Regarding the implementation, the scripts were divided in a logical way, similar to the logical architecture used in the chapters of this document (see figure 1.2), each one covering a specific area of the total implementation. This way, when it is necessary to change the implementation of a goal, this document can serve as a guide, providing a better idea of the code's architecture.

However, it is necessary to mention an objective that was not implemented completely correctly with regard to the Rotation of Sources Around the User. Indeed, by dragging the sources with the mouse, the user should be able to move them freely in a hypothetical sphere around him. This sphere is not visible, but when a source is dragged, it will move as if it were attached to the user with a string, spinning around them. To do so, it is necessary to calculate the angle that the source will move, as discussed earlier. This angle calculation is efficient until the source is shifted 90° or -90° in x. The source, when moved close to these angles, starts to behave as if there were a barrier, preventing it from going beyond that limit. This error occurs in the OnMouseDrag() function, more specifically when using the transform.RotateAround() and calculateAngles() functions, and is due to the calculation of the angle through the user's rotation of the source, since the distance vector is projected between the two in one plane.

This error is not only visible when dragging the source in Unity, but also in the Reaper plugin which is receiving information about the azimuth and elevation coordinates of the source in question.

We intend to continue this project for use with VR glasses. In this way, we can create complete immersiveness in terms of audio and image (video).

Acknowledgment

This work is being supported by the Audio and Acoustics Laboratory of the Instituto Superior de Engenharia de Lisboa, ISEL, LAA (https://acusticaaudiolab.isel.pt).

References

[1] Abhayapala, T.D.; Ward, D.B. Theory and design of high order sound field microphones using spherical microphone array. In Proceedings of the IEEE International Conference on Acoustics, Speech, and Signal Processing (ICASSP), Orlando, FL, USA; 2002; pp. 1949–1952. https://doi.org/10.1109/ICASSP.2002.1006151

[2] Meyer, J.; Elko, G. A highly scalable spherical microphone array based on an orthonormal decomposition of the soundfield. In Proceedings of the IEEE International Conference on Acoustics, Speech, and Signal Processing (ICASSP), Orlando, FL, USA; 2002; pp. 1781–1784. https://doi.org/10.1109/ICASSP.2002.1006109

[3] Poletti, M.A. Three-dimensional surround sound systems based on spherical harmonics. J. Audio Eng. Soc. 2005, 53, 1004–1025.

[4] Samarasinghe, P.N.; Abhayapala, T.D.; Poletti, M.A. Spatial soundfield recording over a large area using distributed higher order microphones. In Proceedings of the IEEE Workshop on Applications of Signal Processing to Audio and Acoustics (WASPAA), New Paltz, NY, USA; 2011; pp. 221–224.

[5] Dolby Atmos Audio Technology. Available online: https://www.dolby.com/us/en/brands/dolby-atmos. html.

[6] Auro-3D/Auro Technologies: Three-dimensional Sound. Available online: http://www.auro-3d.com/.

[7] Pulkki, V., "Communication Acoustics: An Introduction to Speech, Audio and Psychoacoustics", Wiley, 2015. https://doi.org/10.1002/9781119825449

[8] Pulkki, V., "Virtual Sound Source Positioning Using Vector Base Amplitude Panning", The Audio Engineering Society, 1997.

[9] Pulkki, V., "Spatial Sound Generation and Perception by Amplitude Panning Techniques", Helsinki University of Technology, 2001.

[10] Gerzon, M. A., "Periphony: With-height sound reproduction", Journal of the Audio Engineering Society, 1973.

[11] Daniel, J., "Further investigations of High-Order Ambisonics and Wavefield synthesis for Holophonic sound Imaging", Journal of the Audio Engineering Society, 2003.

[12] Morse, P., "Theoretical Acoustics", Princeton University Press, 1986.

[13] Daniel, J., "Spatial Sound Encoding Including Near Field Effect: Introducing Distance Coding Filters and a Viable, New Ambisonic Format", Journal of the Audio Engineering Society, 2003.

[14] Marruffo, A. "A real-time encoding tool for Higher Order Ambisonics", 2014.