Bilateral Communication Between the Unity® Game Engine and Max

Michael Rhoades
David Rodriguez
John Fillwalk

Abstract

In their ongoing research and creative practice, researchers and developers in the Institute for Digital Intermedia Art (IDIA Lab), at Ball State University in Muncie, Indiana, consistently investigate and instantiate multifarious approaches to extended reality (XR). In this paper the authors disseminate their initial investigation into the methodological pipelines required for the perceptual coupling of audio and visual objects in XR. Toward the ultimate goal of enhanced perceptual localization and qualities of immersion, the Unity game engine and Cycling 74's Max were integrated for this project.

Through the use of a three-dimensional Cartesian coordinate system, the authors demonstrate a methodology in which the location of a virtual object is communicated from the Unity game engine, via the Open Sound Control (OSC) protocol, to Max, which then projects a sound sample into the listening space. Using a 3D/360°, 8.1 loudspeaker system configured in a cuboidal pattern, high-order Ambisonics and convolution reverb are implemented to facilitate the perception of the location of the sound object.

Introduction

The Institute for Digital Intermedia Arts (IDIA Lab) is based at Ball State University in Muncie, Indiana. This think tank, incubator, and workshop is composed of expert artists, animators, modelers, programmers, composers, and researchers working in numerous forms of extended reality (XR). As such they commonly extend the boundaries of contemporary interdisciplinary artistic methodologies and ontologies in the areas of augmented, mixed, and virtual reality.

One of numerous highlights in the IDIA lab is the Cave Automatic Virtual Environment (CAVE) (Cruz-Neira et al.). The CAVE consists of an 8.1, three-dimensional, Genelec sound system, designed and installed by Michael Rhoades, and a 4k, 270º, quasi-spherical video projection system. This deeply immersive environment became fully functional in the fall of 2019 and has since inspired numerous projects including the subject matter of the research described herein.

In the winter of 2020, seeing the potential for greater auditory immersion in the CAVE environment, the IDIA Lab director, John Fillwalk, proposed creating a demo that perceptually coupled the movements of visual objects with those of auditory objects employing audio/visual facilities afforded by the CAVE. To facilitate an initial exploration of this idea, a virtual object was envisioned that would provide audio/visual cues as to its location within this virtual space. It was further considered that a standard handheld game controller joystick would control the location of the object. With this basic premise, a team set to work.

The Unity game engine (Unity) was chosen to develop the visual side of this project. Though originally created to democratize game development, today it is used across a wide spectrum of industries to author realtime and rendered 3D applications. From video games, animated movies, and extended reality applications to machine learning, vehicle customization configurators, and immersive occupational training, Unity is an easy choice for high-quality, yet approachable 3D development.

Cycling 74's Max was the platform of choice for the auditory aspect of the project. Max began as a musical programming and compositional environment. Since its inception, it has developed into a powerful interactive multimedia toolset toward that purpose. Further, today it reaches into areas such as auditory spatialization, computer graphics, communication, video creation and incorporation, and numerous other areas of artistic expression.

Though Unity and Max are extremely flexible in areas beyond their originally intended purposes, it is asserted here that a tandem approach, one that capitalizes upon the strengths of each, empowers a union that provides the best of both worlds. Employing Unity for its visual capabilities and Max for its audio capabilities engenders a workflow that is at once broad and deep in the creation of intermedial content.

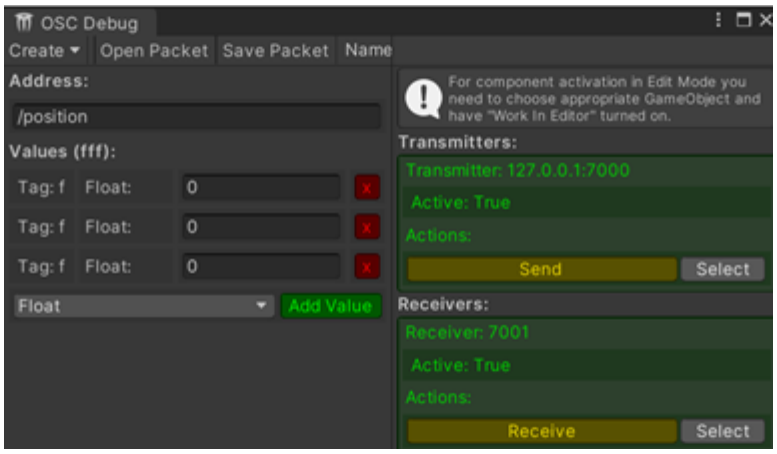
Both the Unity and Max environments provide for the functional implementation of Open Sound Control (OSC), which enables a bilateral communicative pipeline. Using OSC messages to exchange information, in real time with minimal latency, unifies them toward a common purpose that greatly extends the capabilities of each. Passing control data in the form of three-dimensional Cartesian coordinates between them, a visual object in Unity can directly control the perceived location of an audio object in Max. This functionality opens doors to many unique aesthetic and creative opportunities and it also presents novel challenges. Thus the delineation of an initial implementation of this approach is the subject of this paper.

* This paper assumes a basic working knowledge of Unity and Max on the part of the reader.

Establishing the Communication Pipeline

OSC in Unity

Using a Unity C# library called extOSC (Sigalkin), advanced OSC messaging between Unity and Max was implemented for this project. Though several OSC libraries exist for the Unity game engine, extOSC is one of the most modern and robust implementations of the OSC protocol todate, covering much of the 1.1 specification (Freed and Schmeder) including address masks and full data type support. Additional notable features of the Unity package include automatic bundle packing, value mapping, and tight integration with the Unity editor, including built-in transmitter and receiver components and dedicated debug and console windows. As shown in Fig. 1 below, extOSC's dedicated debug window allows developers to create and save OSC messages and test them with configured transmitters and receivers. The dedicated console window, pictured in Fig. 2, provides a separation between OSC messages and general Unity console messages, which simplifies the debugging process.



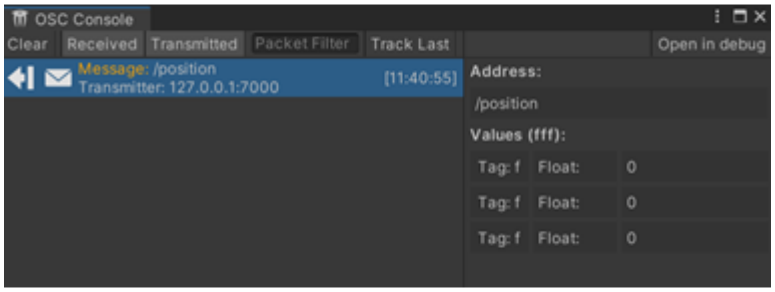
Address:
/position
Values (fff):
Tag: f Float: 0
Tag: f Float: 0
Tag: f Float: 0
Float

For component activation in Edit Mode you need to choose appropriate GameObject and have "Work In Editor" turned on.

Transmitters:
Transmitter: 127.0.0.1:7000
Active: True
Actions:

Receivers:
Receiver: 7001
Active: True
Actions:

Figure 1 - Dedicated Debug Window in extOSC



Message: /position
Transmitter: 127.0.0.1:7000 [11:40:55]

Address:
/position
Values (fff):
Tag: f Float: 0
Tag: f Float: 0
Tag: f Float: 0

Figure 2 - Dedicated Console in extOSC

OSC in Max

Implementing basic OSC communication in Max is relatively straightforward. Since OSC is a UDP networking protocol, instantiating the “udpreceive” object provides the functionality required. For this project, Max and Unity are running on the same machine. Therefore, the udpreceive object, in the “p receive_coordinates” subpatch located in the main patch window, will receive a UDP message via the localhost. The only argument required is the port number that Unity will send its OSC messages to. As can be seen in Fig. 3 below, object_01 is listening on port 7401. Any UDP message sent to that port on the localhost will be accepted by this object and passed along to any objects connected to its output. In this case the next step in the data flow is the “fromSymbol” object, which parses the OSC message from a string to three floating point numbers… one for each of the three Cartesian coordinates, X, Y, and Z. Next the message is sent to the “unpack” object where it is split into three separate outputs, again the three Cartesian coordinates. Lastly, each of the coordinates are sent separately from the p_receive_coordinates subpatch to the “p_set_coordinates” subpatch. There the values sent from Unity are scaled to a range between 0 and 1 and sent out to the ICST Ambisonics plugin object, where they will stipulate, in real time, the position of the auditory object. This will be discussed in detail in a subsequent section.

Also in the p_set_coordinates subpatch, the values of each of the scaled coordinates are averaged and then sent to the “p_convolution_reverb” subpatch where they will be used to designate the relative proximity of a given sound object to the center of the virtual space. This will also be discussed in detail subsequently in this document.

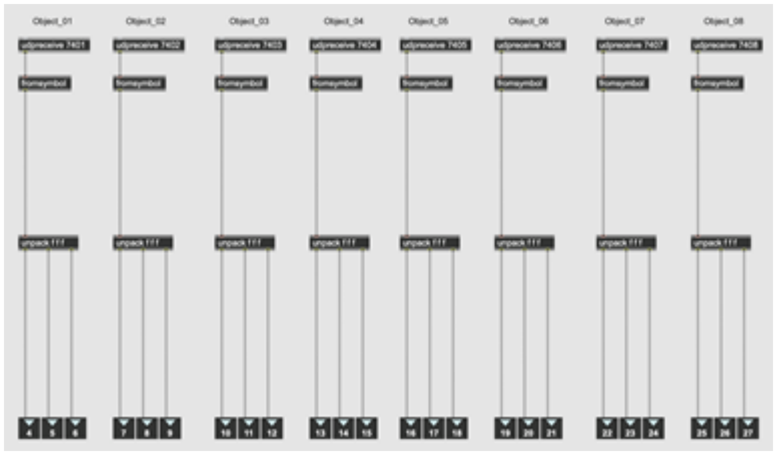


Figure 3 - OSC Listening Ports

Implementation

The Unity Project

Communicating the position of a three-dimensional object in a Unity scene is accomplished by using the extOSC library and the provided OSCTransmitter, OSCMessage, and OSCValue types. First, as seen in Fig. 4, an integer is declared to uniquely identify the object. Though a unique identifier is not strictly required when sending the coordinates of a single object, it is necessary if the application consists of multiple objects. In subsequent iterations of this project, the Max patch was expanded to integrate this feature.

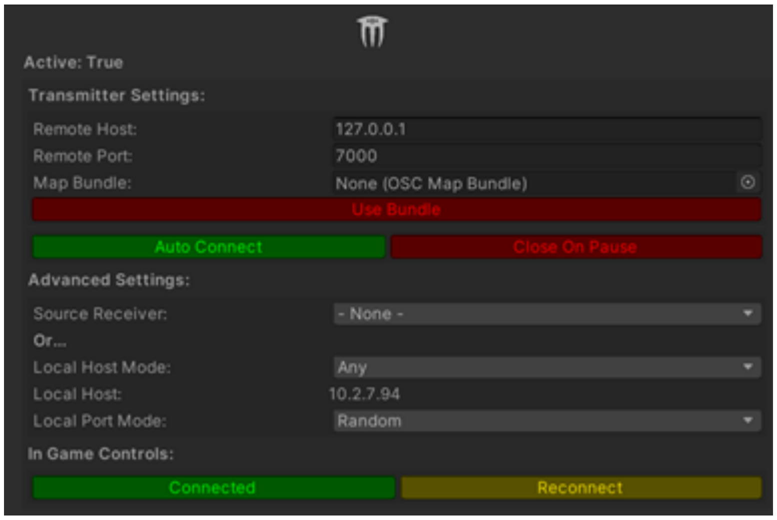
```
    [SerializeField] private int id;
```

Figure 4 - Declaring a unique object identifier

Next, an OSCTransmitter component is referenced. Though it is possible to instantiate and configure a transmitter through code, it is more straightforward to attach an OSCTransmitter component to a game object and configure it in the inspector at edit time. This is shown in Fig. 5. Note that the built-in transmitter component in extOSC allows for a remote server configuration that is separate from the simulation logic. An example of this is demonstrated in Fig. 6 below.

```
    [SerializeField] private OSCTransmitter _transmitter;
```

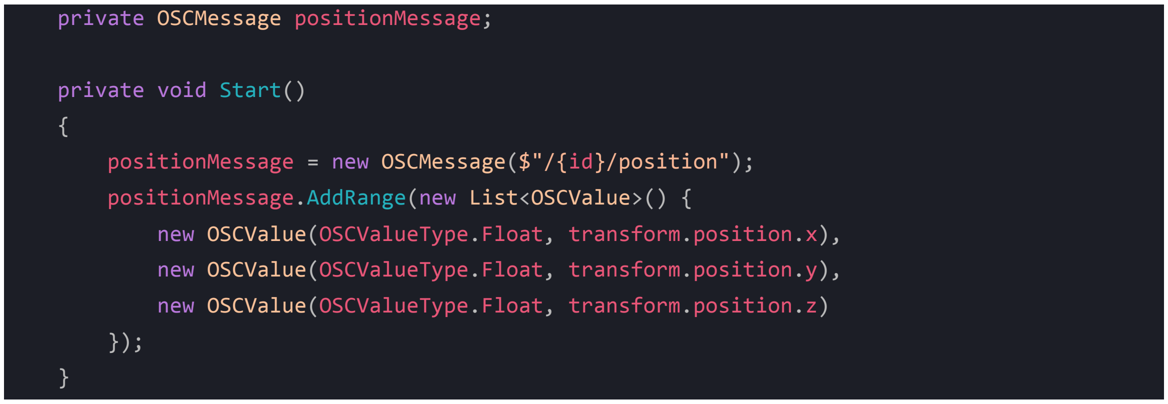
Figure 5 - Reference to an OSCTransmitter Component


Active: True
Transmitter Settings:
Remote Host: 127.0.0.1
Remote Port: 7000
Map Bundle: None (OSC Map Bundle)

Advanced Settings:
Source Receiver: - None -
Or...
Local Host Mode: Any
Local Host: 10.2.7.94
Local Port Mode: Random
In Game Controls:

Figure 6 - Built-in Transmitter Component in extOSC

Finally, an OSCMessage is constructed with the address /{id}/position, stipulating that the data is the position of the {id} object. Because Unity's transform position is represented as a Vector3 struct and Vector3 is not a part of the OSC specification, the position is instead stored as three float components; the x, y, and z values. As demonstrated in Fig. 7, an OSCMessage is declared and initialized containing the object position as floating point numerical components. Note the string interpolation of the id in the message address, which ensures the position of each object is routed to the pertinent identifier.

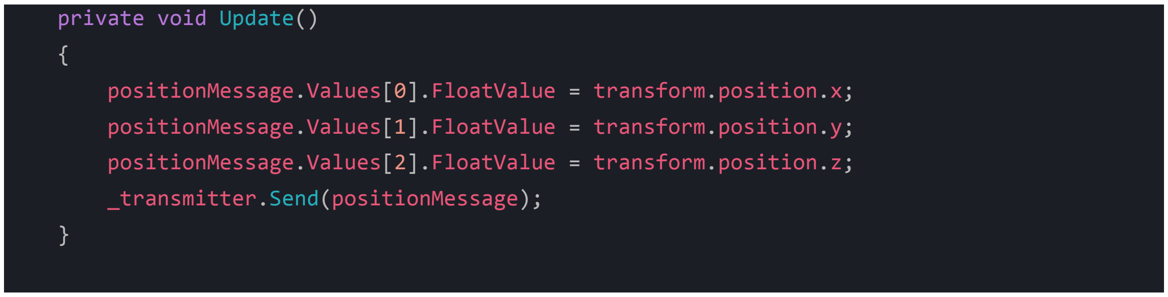


```
private OSCMessage positionMessage;

private void Start()
{
    positionMessage = new OSCMessage($"/{id}/position");
    positionMessage.AddRange(new List<OSCValue>() {
        new OSCValue(OSCValueType.Float, transform.position.x),
        new OSCValue(OSCValueType.Float, transform.position.y),
        new OSCValue(OSCValueType.Float, transform.position.z)
    });
}
```

Figure 7 - Declaration and Initialization of OSCMessage

The initialization of the OSCMessage variable is performed within the Start method. The current position of the object is updated in every frame and a message consisting of its coordinate values is broadcast via the OSCTransmitter. See Fig. 8 below.



```
private void Update()
{
    positionMessage.Values[0].FloatValue = transform.position.x;
    positionMessage.Values[1].FloatValue = transform.position.y;
    positionMessage.Values[2].FloatValue = transform.position.z;
    _transmitter.Send(positionMessage);
}
```

Figure 8 - Updating and sending the OSCMessage

The Max Patch

Background:

In order to produce a real-world analogy to the perceived location of the virtual visual object in the Unity environment, two audio paradigms are primarily employed in Max; High-order Ambisonics (HOA) and Convolution Reverb (CR). Through wavefield interference patterns, HOA utilizes a weighted sum of each loudspeaker in a venue to contribute toward the creation of sound fields that can be positioned in specified locations. The CAVE's 8.1 sound system, using 8 - Genelec 8030s and a 7050 sub, is configured in a 3D cuboidal arrangement. This means there are four loudspeakers in a square plane above the 270º projection screen and four below it as depicted in Fig. 9 below.

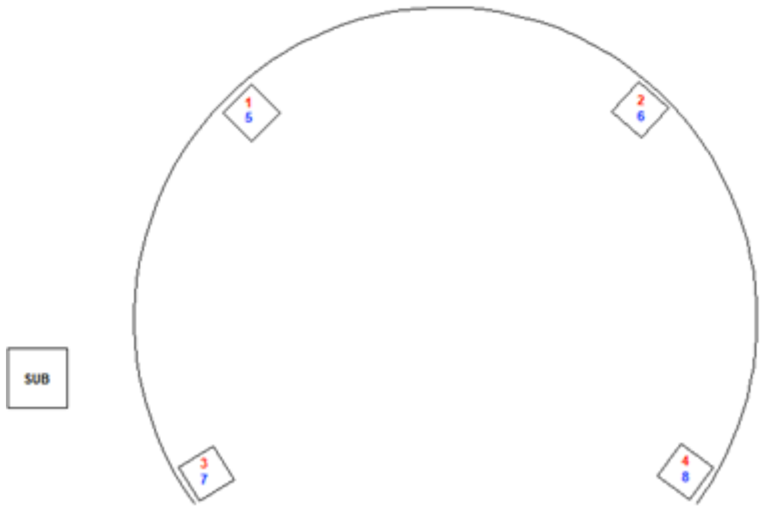


Figure 9 - 8.1 Loudspeaker Configuration in the CAVE

The sound field can, theoretically, be projected to any position within the venue. As demonstrated in Fig. 10 a cuboidal three-dimensional Cartesian coordinate system is implemented to specify the location of the sound object. Notice that the loudspeaker locations in the CAVE are directly correlated to the Cartesian coordinates.

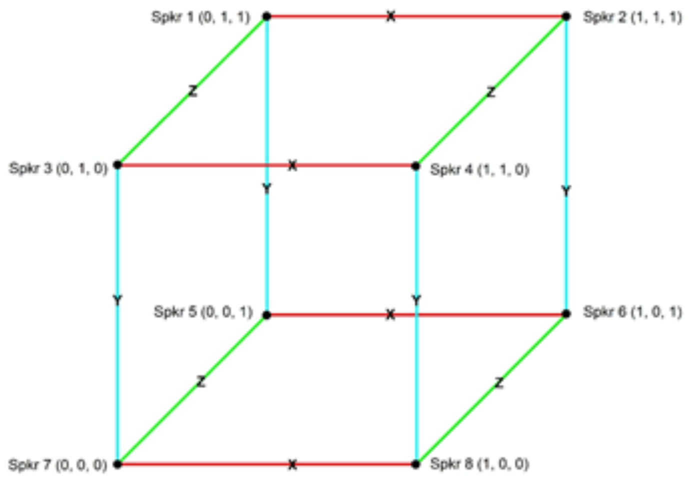


Figure 10 - Loudspeaker Cartesian Coordinate System (Rhoades)

Though HOA is quite effective in stipulating the perception of height, width, and depth of a sound object within the virtual space, it is not thoroughly convincing regarding distance cues (Rhoades). To enhance the perception of distance, reverberation was integrated into the process.

“Convolution is a mathematical procedure whereby one function is modified by another” (Heintz). Convolution reverb is an ideal solution for producing distance cues in audio objects. It consists of mapping the mathematical representation of a reverberation signature, derived from an impulse response (IR) recording, onto another sound or sound event (Rhoades). Generally for this project, a mix between a 100% reverberant (wet) and 100% non-reverberated (dry) sound sample is determined by the proximity of the virtual visual object to the center of the virtual space. For instance, if the visual object is close to the center of the space, there is very little reverberation in the sample playback. Conversely, the opposite is true when the visual object is far from the center… it becomes nearly 100% reverberant. This reverberation characteristic is generally consistent with our everyday real-word perception of distance. When coupled with the previously mentioned approach to HOA, the location of a given sound event can be effectively and consistently perceived.

Implementation:

Figure 11 displays the main window of the Max patch utilized for this project. The basic audio flow begins with the sfplay object in the p_sfplay sub patch. There a monophonic sound sample is played in a constant loop when the “start” button is activated. The output from the subpatch is routed to the CR subpatch, p convolution_reverb shown in Fig. 12.

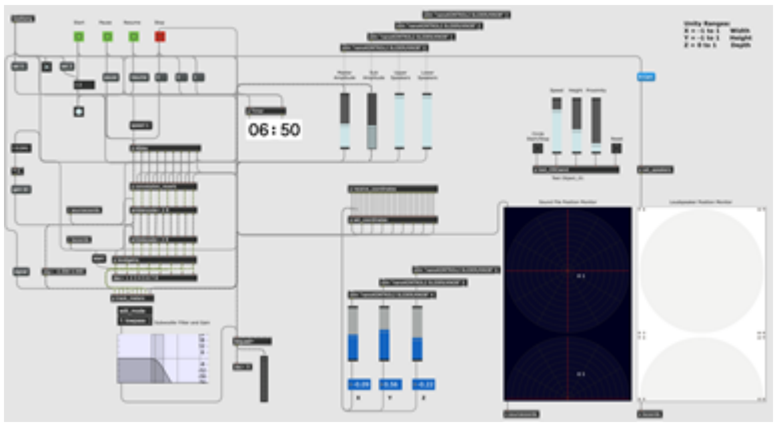


Figure 11 – Main Max Patch Layout

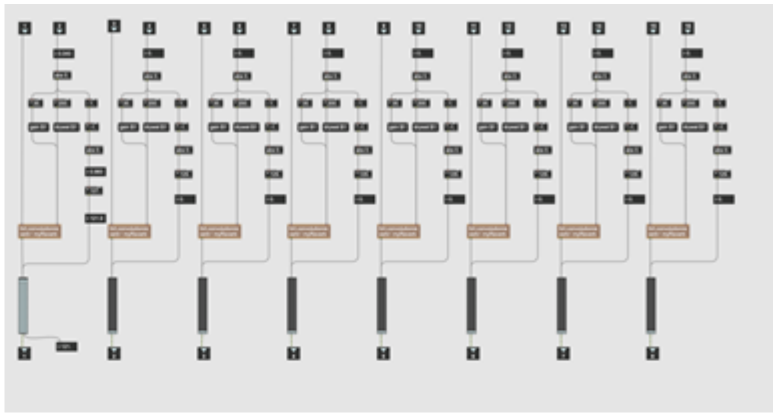


Figure 12 – Subpatch Implementing Convolution Reverberation

The hirt.convolutionreverb plugin was created as a part of the Huddersfield Immersive Sound System, which finds its origin at Huddersfield University (Harker, Tremblay). It is an excellent convolution reverb choice for Max offering several key features required for this project. Of particular interest are the implementation of the wet/dry mix and the amplitude parameters. The intention in the former was to determine the reverberation effect, applied to the sound sample, depending upon the virtual visual object's distance from the center of the virtual space. As previously mentioned, the farther from the center of the virtual space the visual object is located, the more reverberation is applied to the sample and the closer it is to the center the less reverberation is applied. This is accomplished by appropriately mixing the dry sample with the wet sample.

The amplitude parameter of the CR plugin controls the loudness of the sound sample depending upon the virtual visual object's distance from the center of the virtual space. The sound sample is played at full amplitude near the center and it becomes inaudible at the extremities. Together, these effects create convincing localization distance cues since it generally replicates two dynamic aspects of real world audio perceptions.

Next, the audio sample is sent to the second-order Ambisonics encoder and decoder where it is output to a weighted sum of the 8 loudspeakers in the CAVE according to the location of the sound field as stipulated by the three-dimensional Cartesian coordinates sent from Unity to Max via OSC. Fig. 13 shows the ICST plugin, which was created by the Zurich University of the Arts (ZHdK). The right half of the image shows the manner in which the loudspeaker locations within the venue are determined for the plugin. These are directly correlated to the speaker locations shown in Fig. 10 above. The left half of the image shows the location of the current coordinates of the sound field being projected. The spherical top half of both images offers a top view of the listening space and the hemispherical bottom half shows a frontal view.

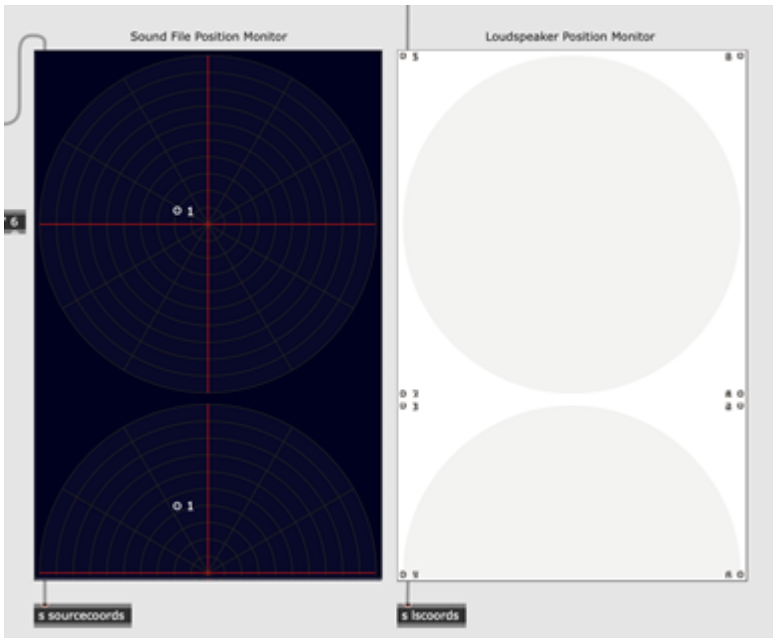


Figure 13 – ISCT Ambisonic Object Plugin

When the ICST plugin has determined the audio signals to be sent to each of the 8 loudspeakers in the venue, the signal has been altered from a monophonic sample to an 8-channel sound source. These eight samples, which are usually in a constant state of flux according to the varying location of the virtual visual object, are then sent to the p levelgains sub patch where the amplitude of the upper level of loudspeakers are mixed with that of the lower level loudspeakers. This enables a precise balance of equal perception between them. This is necessary due to the fact that the height dimension of a three-dimensional sound system is the most difficult to perceive (Hollerweger). Stipulating a precise balance between the two levels of loudspeakers is an important step in projecting the localization of the audio object, and the subsequent coupling of audio and visual objects, in a perceptible manner.

The last step in the audio flow of the Max patch is the digital audio converter object, which sends the audio signal to the audio interface hardware. In the CAVE this means a Focusrite® Scarlett 18i20.

Conclusions

Though not tested empirically, the confluence of these audio/visual processes, uniting Unity and Max, forms a cohesive and powerful determinant toward the perceptual localization of the virtual visual object. While experiencing it in the CAVE one certainly gains a sense of the audio/visual object existing and moving within the three-dimensional virtual space. There appears to remain a vast area of research and implementation to be explored in such an approach, which has powerful implications in several areas of XR.

Though its implementation is highly dependent upon having access to facilities and venues that are conducive to such an interaction, it can nonetheless be generalized to other situations. However, it is easy to imagine a future in which highly immersive venues become increasingly available, thus providing for the implementation of these and other similar techniques.

This project has served as a dynamic proof of concept with regard to the perceptual coupling of audio/visual of objects through the use of Unity and Max as integrated and implemented through OSC communication protocols.

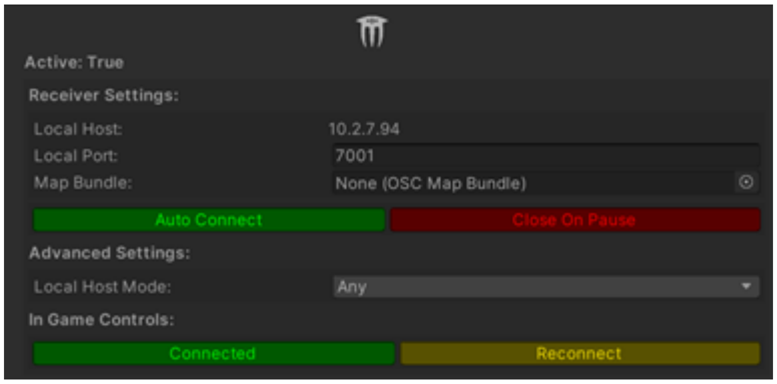
One potential area of future enhancement to this project is grounded within the fact that there is a disparity between the three-dimensionality of the sound system as compared to the two-dimensionality of the projection system. For instance, with the current configuration, as a visual object approaches a close proximity to the viewer it simply becomes larger on the projection screen, which is a visual proximity cue yet one that cannot exist in isolation. Object size is an aspect of visually perceived proximity, however it needs to be accompanied by the perception of an object consuming the space between itself and the viewer to be truly convincing. Conversely, moving a sound field within close proximity to the listener in this three-dimensional venue, closely mimics its corollary in a real space/time. Implementing a stereoscopic visual projection system, which ascribes perceptible three-dimensionality to the visual experience, would alleviate this disparity. This concern can also be addressed by implementing the approach for projection within a head-mounted display, which is often a more financially viable option. Three-dimensional stereoscopic visual projection coupled with the three-dimensional sound system greatly enhances such an immersive experience.

Future Development

This relatively simple project provides far-reaching implications in the future work of the IDIA Lab being a proof-of-concept for more extensive projects. In its present state, up to 8 visual objects can be correlated to 8 auditory objects and this number can easily be expanded. Thus the basic ideas explored in this research can be generalized to more expressive and active environments such as those in gaming, interactive installations, telematic expressions, telecommunication, visual music, and numerous others. Further, this initial foray into such a paradigm enables one to imagine a future in which holographic and holophonic, 3D audio/visual, implementations could assume much more fully immersive trajectories than those currently in place.

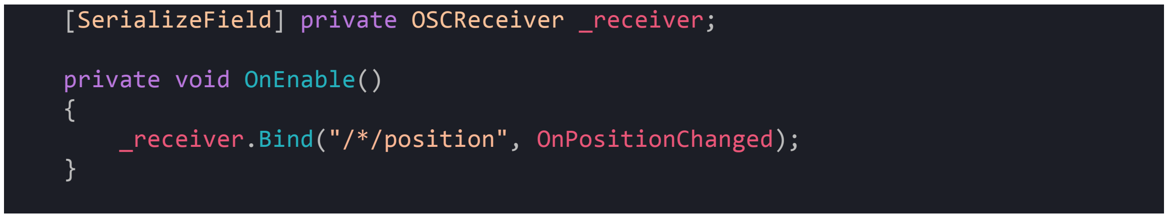
At present, the communication between Unity and Max in this project is unidirectional. The simulation in the Unity game engine transmits object positions to Max via OSC. However, future development could also explore communication in the opposite direction wherein Max could send data to Unity. Imagine a parameter of an auditory object, for instance the amplitude, in Max stipulating the location of a visual object in Unity. Such relationships could provide for very interesting immersive experiences.

This could be accomplished with the extOSC library's OSCReceiver and OSCBind and a reference to an OSCReceiver component, see Fig. 14. As with the transmitter, this component can be configured in the editor and dragged onto the reference. Then an address can bind to a method callback that takes an OSCMessage as a parameter. When the receiver receives a message that matches the binding, the callback is instantiated with the message. Similar to the OSCTransmitter in Fig. 6 above, the built-in receiver component in extOSC allows for a remote server configuration separate from the simulation logic. This is shown in Fig. 14 below. Note that a reference to an OSCReceiver component, Fig. 15, is established and an address is bound to a callback. Also, note the "*" wildcard in the message address. This stipulates that any id will trigger this callback.

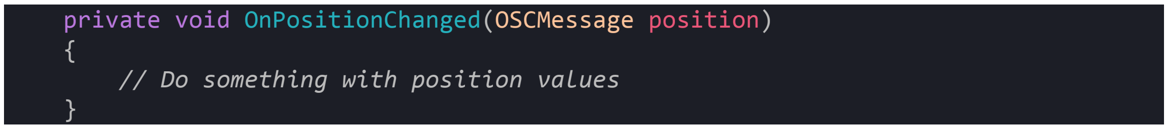

Active: True
Receiver Settings:
Local Host: 10.2.7.94
Local Port: 7001
Map Bundle: None (OSC Map Bundle)

Advanced Settings:
Local Host Mode: Any
In Game Controls:

Figure 14 - Built-in Receiver in extOSC


```
    [SerializeField] private OSCReceiver _receiver;

    private void OnEnable()
    {
        _receiver.Bind("/*/position", OnPositionChanged);
    }
```



```
    private void OnPositionChanged(OSCMessage position)
    {
        // Do something with position values
    }
```

Figure 15 - Reference to an OSCReceiver component

References

Cruz-Neira, Carolina, et al. “The CAVE: audio visual experience automatic virtual environment.” *ACM Digital Library*, 1 June 1992, https://dl.acm.org/doi/10.1145/129888.129892. Accessed 27 July 2021.

Freed, Adrian, and Andy Schmeder. “Features and Future of Open Sound Control version 1.1 for NIME.” *Features and Future of Open Sound Control version 1.1 for NIME*, 2009, http://cnmat.org/files/attachments/Nime09OSCfinal.pdf. Accessed 14 July 2021.

Harker, Alex, and Pierre Alexandre Tremblay. “The Hisstools.” *University of Huddersfield*, https://research.hud.ac.uk/institutes-centres/cerenem/projects/thehisstools/. Accessed 27 July 2021.

Hollerweger, Florian. “An Introduction to Higher-Order Ambisonic.” April 2005, http://decoy.iki.fi/dsound/ambisonic/motherlode/source/HOA_intro.pdf. Accessed 27 July 2021.

Rhoades, Michael. “Composing Holochoric Visual Music: Interdisciplinary Matrices.” *VTechWorks Home*, Virginia Tech, 1 February 2021, vtechworks.lib.vt.edu/handle/10919/102159. Accessed 27 July 2021.

Sigalkin, Vladimir. “GitHub - Iam1337/extOSC: extOSC is a tool dedicated to simplify creation of applications in Unity with OSC protocol usage.” *GitHub*, 20 May 2021, https://github.com/Iam1337/extOSC. Accessed 14 July 2021.

ZHdK - Zürcher Hochschule der Künste. “Downloads: Ambisonics Externals for Maxmsp.” *ZHdK*, www.zhdk.ch/forschung/icst/software-downloads-5379/downloads-ambisonics-externals-for-maxmsp-5381. Accessed 27 July 2021.

Disclaimers