OSC-XR: A Toolkit for Extended Reality Immersive Music Interfaces

David Johnson
Department of Computer Science
University of Victoria
davidjo@uvic.ca

Daniela Damian
Department of Computer Science
University of Victoria
danielad@uvic.ca

George Tzanetakis
Department of Computer Science
University of Victoria
gtzan@ieee.org

ABSTRACT

Currently, developing immersive music environments for extended reality (XR) can be a tedious process requiring designers to build 3D audio controllers from scratch. OSC-XR is a toolkit for Unity intended to speed up this process through rapid prototyping, enabling research in this emerging field. Designed with multi-touch OSC controllers in mind, OSC-XR simplifies the process of designing immersive music environments by providing prebuilt OSC controllers and Unity scripts for designing custom ones. In this work, we describe the toolkit's infrastructure and perform an evaluation of the controllers to validate the generated control data. In addition to OSC-XR, we present UnityOscLib, a simplified OSC library for Unity utilized by OSC-XR. We implemented three use cases, using OSC-XR, to inform its design and demonstrate its capabilities. The Sonic Playground is an immersive environment for controlling audio patches. Hyperemin is an XR hyperinstrument environment in which we augment a physical theremin with OSC-XR controllers for real-time control of audio processing. Lastly, we add OSC-XR controllers to an immersive T-SNE visualization of music genre data for enhanced exploration and sonification of the data. Through these use cases, we explore and discuss the affordances of OSC-XR and immersive music interfaces.

1. INTRODUCTION

In 1992 Jaron Lanier performed *The Sound of One Hand*, a live improvisation using the three instruments designed for the EyePhone (an early virtual reality headset) [1]. A remarkable aspect of his performance (aside from the technologies) was that Lanier was able to simultaneously play multiple instruments to perform music that could not easily have been performed with traditional instruments. Lanier's work showed the potential for immersive musical performances, but since then there has been limited research exploring the musical interactions afforded by virtual reality (VR) and related extended reality (XR) technologies. When Serafin et al. [2] recently surveyed the state of art in virtual reality music instruments (VRMIs) in 2016, the number of interfaces available was fairly small. The capabilities and relatively few design constraints of XR create the potential for a wide array of immersive interfaces based on any of Miranda and Wanderley's four categories of music interfaces: *Augmented Musical Instruments*, *Instrument Like Controllers*, *Instrument Inspired Controllers*, and *Alternate Controllers* [3]. With such broad possibilities, more research is needed to increase our understanding of the affordances of immersive environments and interaction techniques best suited for music control.

To support further research into immersive interfaces for music, we present OSC-XR, a toolkit for rapidly prototyping immersive musical environments in XR using Open Sound Control (OSC), a communication protocol widely used in audio software [4]. Influenced by multi-touch OSC controllers, OSC-XR provides developers with a wide range of readily available components in order to make designing immersive environment more accessible to researchers and sound designers. In this paper, we discuss the infrastructure of OSC-XR, validate its generated data by comparing with a popular multi-touch OSC controller, and present three environments developed to demonstrate its capabilities for immersive interface design.

2. RELATED WORK

2.1 XR Music Interfaces

In one of the first research studies on virtual music performance, Mulder, Fels and Mase [5] designed virtual 3D instruments that users interacted with using CyberGloves and motion tracking sensors. While the instruments were not displayed in an immersive environment, they did explore interactions in 3D desired for immersive performances. Using a fully immersive environment, Mäki-Patola et al. [6] developed and analyzed four immersive music interfaces based on physical models. In their findings, they reported that because VR is a different medium compared to the real world, mimicking traditional instruments in immersive environments may not result in better instruments unless it is used to augment real instruments with additional control. Rather than mimicking existing instruments, Berthaut et al. [7] proposed 3D reactive widgets for musical performance with interactions that went beyond what is possible in the real world. The reactive widgets represented complex multi-process sounds with many parameters that would be difficult to interact with in the real world. Using VR with carefully designed gestures and audiovisual mappings allowed the user to easily interact with multiple widgets to generate an expressive musical interaction. In these examples, music performance was controlled using

ISBN 978-84-09-08518-7 ISSN 2518-3672

self contained virtual objects tightly coupled with sound generation limiting customization or extensibility.

A immersive interface proposed by Moore et al. [8], called The Wedge, allowed users to not only perform music in VR but also build and customize their performance environment. With this interface a user could build a customized performance environment by selecting and combining note objects from a palette to form musical chords and sequences. The interface used two simple gestures for interaction, one gesture for playing notes and another gesture for building the interface by placing notes within the environment. The interface had limited capabilities for generating sounds as complex as the previously discussed reactive widgets but showed how XR can be used to quickly build customized interfaces for musical performance. Our work takes inspiration from this previous research while also allowing users to build more complex sound environments through customizable OSC controllers.

The previous works all rely on interaction with virtual objects through standard input devices, like hand controllers or data gloves. Bottcher et al. [9] instead proposed a VRMI for interacting with tangible music controllers. In this work, the authors built physical flute and drum like controllers which were represented in the virtual environment as 3D objects. Interaction with the controllers was mapped to the parameters of a physical model. By moving the controllers, the user was able to change the dimensions of the physical model, and it's virtual representation in real-time, while simultaneously using it for a musical control. Using tangible interfaces as controllers for virtual music performance provided users with a clear understanding of the affordances and constraints for interaction.

The presented works provide interesting use cases and examples of immersive environments for musical performance that demonstrate the potential of using XR for musical expression. For a more extensive overview of recent VRMI see the survey on the current state of the the art by Serafin et al. [2]. The systems presented, however, are standalone and have limited capabilities for designing new environments. OSC-XR provides designers with a more general toolkit to make building and prototyping new musical environments more accessible.

2.2 OSC Controllers

During their research on OSC, Wessel and Wright [4] discussed the affordances of using digitized tablets for musical control as well as potential mapping strategies for gestural control of music. This work has inspired a number of multi-touch OSC based control surfaces. TouchOSC [1] is one of the most popular multi-touch controller applications. It provides users with prebuilt layouts using TouchOSC's standard control widgets. In addition to the set of existing control interfaces, users may also use the TouchOSC Editor to build their own interfaces from the prebuilt widgets. The authors of two other multi-touch toolkits, Argos [10] and Control [11], cited the influence of TouchOSC on their flexible design.

[1] https://hexler.net/software/touchosc

Argos was an application for building multi-touch interfaces for musical control using OSC [10]. Using Argos users were able to design control interfaces from a library of prebuilt widgets, such as knobs, sliders and buttons. Additionally, Argos provided developers a set of C++ classes, built on openFrameworks, for creating their own widgets. Similarly, Control, let users design custom interfaces from a set of prebuilt widgets using JSON to define the interface structure [11]. Control was set apart from from other interfaces by giving users the ability to add customized functions to their widgets using JavaScript. The popularity of multi-touch OSC controllers, especially TouchOSC, show that OSC based applications with flexible design support needs of designers. While these applications were all designed for multi-touch surfaces, OSC-XR is inspired by the underlying theme of flexible design through prebuilt objects and customized scripting.

Multi-touch devices are not the only place OSC has been used to build control environments. Hamilton [12] used OSC in the design of UDKOSC, a immersive musical performance environment for the Unreal Development Kit (UDK). With this system Hamilton was able to perform in an immersive environment using avatars that interacted with objects in the virtual environment. Our work differs from Hamilton's in that OSC-XR uses a design metaphor based on standard music control idioms rather than the game like metaphor seen in Hamilton's work.

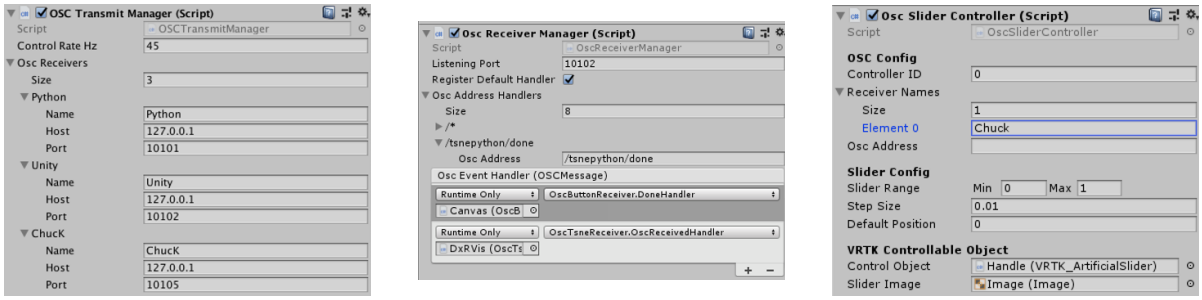
2.3 Audio Programming in XR

Immersive environments for XR are typically developed using dedicated game engines, such as Unity [2] or Unreal Engine [3], which are designed to simplify the process of developing 3D environments through a suite of tools that include advanced graphics rendering pipelines and physics engines. They also include sound engines for playback of sound files with mixing, added effects and sound spatialization. There is, however, minimal support in game engines for audio synthesis capabilities desired by sound designers. Unreal Engine has an experimental package for sound synthesis [4] but the limited features of the environment may not provide sound designers with the full tool set provided existing audio programing languages. To support the design of immersive music environments there is a need for more robust audio synthesis capabilities.

Currently there are a some audio programming languages that support audio synthesis and processing with Unity. Faust, for example, can compile to a C library for use as a Unity Plugin [13] and LibPD has a C# wrapper that can be integrated with Unity [14]. Most recently, a plugin to support the use of ChucK within the Unity development environment, called Chunity, was developed [15]. While these systems all add support for audio synthesis to Unity, designers must use Unity scripting to setup parametric control of the patches. Because OSC-XR uses OSC for control, it allows designers to directly integrate their favorite audio synthesis tools into the design process.

[2] https://unity3d.com
[3] https://www.unrealengine.com
[4] http://bit.ly/UnrealSynth

ISBN 978-84-09-08518-7 ISSN 2518-3672



(a) The OSC Transmitter Interface

(b) The OSC Receiver Interface

(c) The OSC-XR Slider Interface

Figure 1: Example Unity Inspector Interfaces for OSC-XR

3. UNITY OSC LIBRARY

To implement OSC in Unity, many projects have used Jorge Garcia's UnityOSC library [5] . We have found this library to be somewhat difficult to integrate in new projects. To simplify the process of OSC configuration we present a new library, UnityOscLib, that builds on Garcia's core OSC classes and integrates them into the Unity development work flow. The new library simplifies configuration by implementing separate `MonoBehaviour` (a Unity base class from which all Unity scripts must be derived) classes for receiving and transmitting OSC messages. The configuration process has also been simplified by exposing OSC properties in the Unity Inspector as well as through Unity scripting. At the time of writing, we have not exposed OSC bundles or timestamps through the UnityOscLib API (but plan to do so in a later release). This section briefly introduces the new library while complete details, including examples, can be found on the project's Github repository [6] .

The `OscTransmitManager` is a Unity `MonoBehaviour` that handles all aspects of transmitting OSC messages. To send OSC messages from a Unity application, add the OSC transmit manager to one `GameObject` and configure connection information for one or more OSC receivers. OSC receiver configuration details are exposed through the Unity Inspector, as shown in Fig. 1a, in addition to the scripting interface using the `AddReceiver` method. Once configured, the environment is ready to transmit OSC messages using `SendOscMessage` or `SendOscMessageAll`.

The OSC transmit manager also implements an optional control rate feature, to configure the frequency of OSC message transmission. Transmitting OSC messages may be triggered by specific events that only occur periodically, such as collision events, but they may also be triggered continuously, for example when an objects position is changing. This type of continuous data is generally calculated at a rate specified by Unity's `Update` or `FixedUpdate` messages. With XR these messages typically occur at around 90 frames per second (FPS) or faster as technologies improve. Audio applications may not be able to handle incoming messages at this rate. The UnityOscLib control rate feature is implemented to limit the rate OSC messages are transmitted. To use this feature, developers should register a method that transmits OSC messages with the `OnSendOsc` event of the `OscTransmitManager`. Any methods registered with`OnSendOsc` will be called at the control rate specified in the Unity Inspector.

The `OscReceiverManger` is a Unity `MonoBehaviour` class that manages the routing and handling of incoming OSC messages. To receive OSC messages in a Unity application, add the OSC receiver manager to one `GameObject` in the scene and configure the receiver with the port to listen on, see Fig. 1b. OSC address routing is implemented using Unity Events for configuration in the Unity Inspector as well as using delegate events for C# scripting. To route messages based on in the inspector, UnityOscLib exposes an interface in the inspector to add any number of OSC addresses and one or more handler methods for each address, see Fig. 1b. Additionally, the receiver manager's `RegisterOscAddress` method is used to add OSC addresses and event handlers through Unity scripting. All OSC event handler methods used should accept a UnityOscLib `OscMessage` as an argument. This implementation provides flexible implementation for adding OSC handling during environment design or at runtime.

4. OSC-XR

The main contribution of this work is the OSC-XR toolkit for designing immersive XR environments for music control. It is developed using Unity and UnityOscLib to provide sound designers a simple interface for prototyping interactions in immersive environments. The OSC-XR toolkit contains two main components for building environments, 1) a set of scripts that can be attached to any Unity `GameObject` to transmit the object's state via OSC and 2) a set of prebuilt music controller, called controller prefabs, for transmitting control data via OSC, similar in concept to widgets in TouchOSC. With this infrastructure, developers with limited Unity experience can quickly design immersive music environments through the use of the controller prefabs. Furthermore, more experienced developers can easily extend custom `GameObjects` with OSC capabilities through the scripting interface. Finally, the robust Unity platform affords customization and extension of any OSC-XR components to those familiar with Unity and C#. The flexible design of OSC-XR, combined with the power of Unity, supports rapid prototyping to make designing im-

[5] https://github.com/jorgegarcia/UnityOSC
[6] https://github.com/fortjohnson/UnityOscLib

ISBN 978-84-09-08518-7 ISSN 2518-3672

Name	Description	Example OSC Message
OscSlider	A slider prefab with position mapped to a configurable range, see Figs. 1c and 3a	`/slider/value 1 4.5`
OscPad	A drum prefab with pressed and released events including an optional velocity, see Fig. 3a	`/pad/pressed 1 1.5`
OscGyro	A virtual gyroscope prefab for sending angular velocities normalized to a range of 0 to 1	`/gyro/velocities 1 .9 .7 .5`
OscTransform	A script for sending transform data via OSC	`/trans/local/pos 1 0.5 1.3 2.0`
OscTrigger	A script for sending Unity Trigger events; includes an ID and position information for the triggering object	`/trigger/enter 1 0.5 0.4 1.0 2`

Table 1: Examples of available OSC-XR controller prefabs and scripts. Refer to our GitHub repository for a complete list.

mersive environments quicker more accessible.

OSC-XR was developed using Unity and tested using the Samsung Odyssey Windows Mixed Reality Headset [16] with SteamVR [17]. By making use of the well known Virtual Reality Toolkit (VRTK) [18], OSC-XR should work with any of VRTK's supported platforms and hardware, affording multi-platform support. The remainder of this section discusses the OSC-XR infrastructure. The details we provide here are intended to give the reader high level understanding of how the toolkit is structured but we encourage the reader to visit the project's Github repository [7] for complete details, including video examples.

4.1 OSC Controller Prefabs and Scripts

Adding OSC controller prefabs to a Unity scene is the quickest way to get started with OSC-XR. To implement a controller simply add the prefab from the `OSCXR/Prefabs` folder to the Unity game hierarchy. Once added to the scene, modify the object's transform as desired. At this point the object is ready to use in the environment. For additional configuration each controller exposes a set of properties in the Unity Inspector, see Fig. 1c. Table 1 lists the descriptions of a few of the available OSC controller prefabs, including an example OSC message for each. Developers can further customize the controller prefabs using Unity tools. For example, the visual aspects of any of the controller prefabs can be modified by configuring the Unity components that comprise each object, such as the meshes or materials.

The OSC-XR scripting interface allows developers to quickly add OSC capabilities to any `GameObject` by attaching any of the readily available controller scripts to the object. Each of the scripts models a predefined behaviour for triggering and sending OSC messages. By default adding an OSC controller script to a `GameObject` uses that object's state for creating and transmitting OSC messages. This can be overridden on most scripts by updating the `Control Object` property of the script with a different `GameObject`, in which case, the state of the configured `Control Object` will be used instead. This is useful when building a composite object where the tracked object is not the top level object. For example, the slider controller prefab implements this design in which case the state of prefab's handle is used for control data, as seen in Fig. 1c. Table 1 lists the descriptions of a few of the available OSC-XR controller scripts, including an example OSC message for each.

Designers wishing to build their own OSC controller scripts should extend OSC-XR's `BaseOscController`. This class includes a number of base properties for OSC configuration, the controller ID and the OSC address, as well as methods for sending OSC messages. Furthermore, the class automatically registers the method, `ControlRateUpdate` to support transmitting OSC messages at the control rate specified in the OSC transmit manager. Any controller script that needs to send data at the configured control rate should override `ControlRateUpdate` with a method that generates and transmitting OSC data. Each custom script should extend these options as needed to achieve the behaviour being modeled.

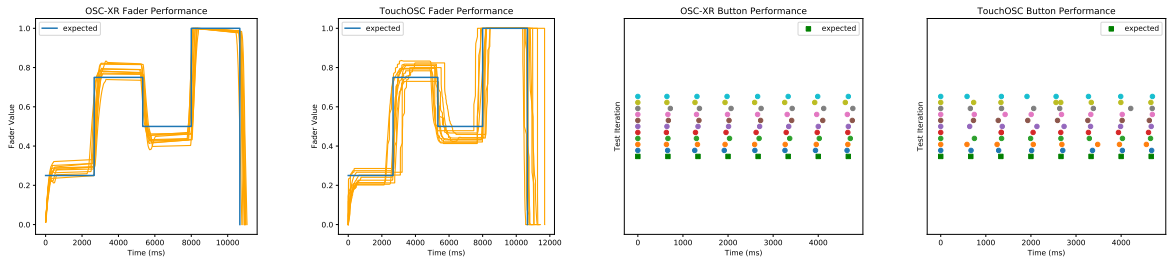
4.2 Control Data Validation

To ensure that data generated by OSC-XR is consistent with users' expectations, we employ two simple user tasks for comparing OSC-XR with TouchOSC. An OSC receiver is implemented to log data generated by each task for an analysis of user performance. One task utilizes a slider controller (or fader widget in TouchOSC) to validate the control precision of the different applications. The second task utilizes a pad controller (or button widget in TouchOSC) to evaluate rhythmic control of the different interfaces. One of the authors, who has intermediate musical skills, performed the tests to validate that the data sent from OSC-XR is consistent with existing systems.

4.2.1 Slider and Pad Evaluation

To perform the slider evaluation task, a user sets the position of the slider to specific values at regular time intervals. For this work, the task requires setting the values of the sliders to 0.25, 0.75, 0.50, and 1.00 in that order. The user is required to transition the slider to each value on every fourth beat at a tempo of 90 beats per minute (BPM) indicated using a metronome. To perform a baseline analysis of the control data, a user performed the task ten times in both OSC-XR and TouchOSC. The output of the user's performance is then compared to signal representing the expected data, figures 2a and 2b show the results of each run for both applications overlaid with the expected output. The data is compared quantitatively by calculating

[7] http://github.com/fortjohnson/OSC-XR



(a) OSC-XR Slider (b) TouchOSC Slider (c) OSC-XR Pad (d) TouchOSC Pad

Figure 2: The results of control data validation for slider controllers and pad controllers.

the euclidean distance between the actual and the expected output to calculate an error value. This value is averaged over all iterations for a final error metric. Table 2 lists the average errors of this task for both interfaces.

To perform the pad evaluation task, a user presses a pad controller for eight beats at a tempo of 90 BPM using a metronome to keep time. As in the previous task, baseline analysis of the data is captured with a user that performs the task ten times. Results of each iteration are shown in Figures 2c and 2d, for OSC-XR and TouchOSC respectively. Each iteration is represented as a row of dots where each dot in the row indicates a pad pressed event. For comparison the expected beat times are shown with the green squares in the bottom row. The error for each iteration is calculated as

$$\frac{\sum_{n=1}^{N} \mid t_{exp} - t_{act} \mid}{N} \quad (1)$$

where N is the number of beats per iteration, t_{exp} is the expected time of the beat, t_{act} is the actual time of the pressed event from the user. The errors are averaged over all iterations for the final error metric. The error results for both interfaces are listed in Table 2.

4.2.2 Discussion

Results of the slider evaluation provide a baseline comparison of OSC-XR with TouchOSC. Initial analysis of the data shows similar performance between both applications even though the interactions are slightly different. To move slider in TouchOSC, a user slides their finger across the surface to the new location. Whereas, the OSC-XR slider requires an additional grab interaction to take control of the slider handle before moving it towards its destination. Overall, the OSC-XR slider error is slightly greater than that of TouchOSC. We can compensate for this in OSC-XR by adding a display prefab to the slider for additional feedback. While the interactions required for manipulating sliders are different, this evaluation shows that OSC-XR sliders may perform as well as multi-touch sliders and generate data that is consistent with an application sound designers may already familiar with.

OSC-XR also requires a different technique for interacting with pads due to a lack of haptic feedback. When pressing a pad in OSC-XR users are not provided the same haptic response naturally afforded through interaction with physical objects. Instead users must rely on wrist action and hand controller momentum to control rhythm. Initial evaluation of the pad controller indicates this may not adversely affect rhythmic performance. Results show that the user was able to perform slightly more accurately with OSC-XR. This may be a result of the user relying on wrist action for control rather than pressing a pad with a single finger. Although a larger study is needed to confirm any hypotheses, users may expect rhythmic control from OSC-XR that is consistent with TouchOSC.

	OSC-XR	**TouchOSC**
Slider	3.43	2.98
Pad (ms)	35.2	52.0

Table 2: Average errors for each evaluation task

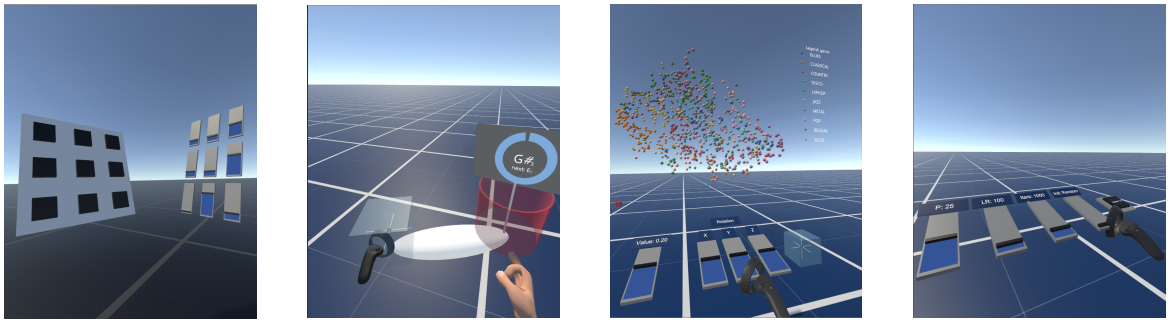
5. OSC-XR USE CASES

In this section, we discuss three prototype use cases for immersive environments developed with OSC-XR. Prototyping the environments helped inform the design OSC-XR. Furthermore, the use cases demonstrate the capabilities of the toolkit in different scenarios providing readers ideas on how OSC-XR might be used for their own projects.

5.1 The Sonic Playground

The Sonic Playground is an immersive environment that explores a variety of OSC-XR controllers. The playground is composed of multiple zones each with a different performance environment. Users are able to navigate between the zones using teleportation, providing the ability to quickly move between different performance environments. The Sonic Playground is designed to explore and demonstrate musical interaction with OSC-XR controllers that communicate with an external audio programming environment.

The Sampler Zone, seen in Fig. 3a, is an immersive sampler environment composed of a 3×3 matrix of pad controllers, to trigger sample playback, and a corresponding matrix of sliders, for additional control of the samples. Pads are configured to send the controller ID as well as pressed and released with included velocity for mapping to sample volume. Each slider is configured to send a value ranging from 0.25 to 5.0 mapped in ChucK to the playback rate of the corresponding sample. Pad and slider events



(a) The Sampler Zone (b) Hyperemin (c) T-SNE (d) T-SNE

Figure 3: Three immersive environments built using OSC-XR to explore its affordances

are all mapped to a corresponding sample using the controllers' IDs. This environment was developed as a proof-of-concept to demonstrate and explore the affordances of typical music controllers in immersive environments.

The first thing to notice in this environment is the size of the objects. Using input controllers for interaction requires the use of large objects as users lose the dexterity that is naturally afforded through interactions using the hand. Integrating hand tracking devices, such as the Leap Motion, may allow for designing dexterous interactions. Another challenge of performing in XR is the lack of haptic response to physical actions, such as tapping a pad. Even with these challenges, virtual pads in a musical environment afford their own interaction style with large expressive motions and gestures. The evaluation of the pad controller, discussed in Section 4.2, indicates that rhythmic control may not be severely affected through the lack of haptics and in this environment we learned the lack of haptics affords a expressive playing style.

The Sonic Objects zone is an environment for prototyping interactive sound environments. It is composed of various OSC-XR controllers that are readily available to communicate with an audio programming environment, such as ChucK. The environment affords the rapid prototyping of interactive sound design by combining OSC-XR's ability to easily add new controllers and interactions with the power of ChucK's development environment to quickly iterate on sound design.

One of the interesting affordances of immersive music environments we explore is the combination of real life physics based interactions with "impossible" interactions that ignore physics. For example, using physics we can toss objects around or stack and lean them on each other to create interesting soundscapes with generative audio patches. Sometimes, however, a user may want to have more control over when parameters of an audio patch stop as they reach a desired state. By ignoring the physics of an object we can lock it in space to immediately stop it from sending OSC messages. For example, an OscGyro object will always send angular velocity data as its being moved, but a user may want to lock in the sound parameters before releasing the object. With this in mind, we decided to add an interaction to freeze the OscGyro anywhere in space. Once frozen the object will be suspended in space until the user grabs the object to move it again. Another interesting affordance we discovered through prototyping in this environment is the ability to easily add automation to controllers through Unity components, such as animation or particle systems. For example, the strongly timed behaviour of particle systems allows for particles to collide with an OSC Trigger controller for initiating musical events at rhythmic intervals. Furthermore, the movement of particles within the controller may be mapped to other audio parameters, such as frequency. These examples show how OSC-XR supports rapid prototyping for exploring and creating new musical interaction techniques in XR.

5.2 Virtual Hyperinstruments

In the NIME community it is common to augment a traditional instrument with sensors to extend its capabilities. Machover and Chung [19] first presented work on this concept with their hyperinstruments in 1989. Typically hyperinstruments extend traditional instruments with direct augmentation of an instrument, such as a violin, with physical sensors [20]. Physical modification of an instrument can be invasive to it's design, therefore, non-invasive techniques have also been developed for augmentation without physical modification, through the use of cameras and depth sensors [21]. These techniques use gesture detection and object tracking for added sound control but provide no visual signifiers to indicate the location of control objects. This is seen in the work of Trail et al. [21] in which they augment a vibraphone with virtual faders that are controlled using mallet tips tracked by a Kinect. Because there are no computer generated signifiers, the fader locations are mapped to the vibraphone keys to signify control locations. Integrating XR in their system would have allowed the authors to add a visual layer to enhance visual feedback.

We have previously explored the virtual hyperinstrument concept by augmenting a physical theremin with virtual objects to visualize the pitch space for music tutoring [22]. We extend that work with Hyperemin, a virtual augmented theremin. OSC-XR controllers are added to the Hyperemin environment to provide real-time control of DSP of the theremin audio. Audio from the theremin is routed to a ChucK patch for playback and audio processing. An OSC 3D Grid controller is added to the environment to control the audio processing, allowing a performer to play

ISBN 978-84-09-08518-7 ISSN 2518-3672

the theremin while also controlling audio processing parameters. Currently, the interaction requires a VR headset and controllers, which may be intrusive to performance but the addition of a LeapMotion sensor, or use of a HoloLens with hand tracking, would address this. In addition to adding sensors directly to the instrument, one of the affordances of XR is the ability to place objects anywhere in the space allowing users to create a customizable control interface not limited to pedals, small device displays or other physical input controllers.

The Hyperemin environment explores the capabilities of OSC-XR for augmenting physical instruments with virtual objects. As XR technology improves we expect that augmenting more traditional instruments will become more accessible. For example, with proper tracking technology it would be possible to attach an OSC Gyro object to the head of a violin and a set of pads to the body adding additional control without physical modifications.

5.3 Immersive Vis Control

OSC-XR was designed with music interfaces in mind but its support for rapid prototyping makes it ideal for prototyping other types of immersive environments that require parametric control and distributed communication. With the emergence of XR technologies, there has been trend of research towards immersive environments for information visualization [23]. With this comes the need to rapidly prototype interaction techniques to support the design of immersive interfaces. In this case, we explore the process of prototyping with OSC-XR to build immersive visualization environment with sonic interaction and distributed communication.

To explore interaction needs of immersive analytics environments, we implemented a 3D visualization of the GTZAN music genre dataset [24]. To visualize the high dimensional data in 3D, 52 spectral and timbral features of each song in the dataset are transformed into 3D coordinates using T-SNE [25]. To visualize the data, we integrate OSC-XR with an immersive visualization toolkit, DxR [26]. Using DxR we were able to quickly develop an immersive scatterplot visualization of the T-SNE data. While DxR provides a 3D interface for controlling the visualization, it is limited to basic point and touch based interactions. Integrating OSC-XR into this environment allows us to quickly prototype new interfaces and interactions to control the visualization as well as augment it with with additional functionality.

We prototyped a new interface to manipulate the DxR generated visualization by augmenting the environment with new control interfaces and interactions using OSC-XR objects. The interface is composed of two control panels, the main panel is to manipulate the view of the visualization, as shown in Fig. 3c, and a second panel controls the T-SNE parameters, which was not previously possible using DxR alone, shown in Fig. 3d. The main panel provides users a set of sliders to directly manipulate view parameters such as zoom and rotation. Since this panel affects the visualization in real-time and would be frequently utilized by a user during data analysis, it is oriented such that a user is facing the visualization while interacting with the controller. The T-SNE control panel, oriented to the left of the user, allows users to to adjust T-SNE parameters and rerun the data transformation on a Python server without having to leave the virtual environment. We also take advantage of OSC-XR capabilities to interact with the visualization marks from a distance. Every mark in the visualization is configured as an OSC Pointer Trigger providing users the ability to interact with marks using the pointer from an input controller. Using this interaction technique a user is able to select any mark in the visualization to playback its associated audio file allowing users to explore the data aurally, as we well as visually. Lastly, visualization marks can be filtered using the pointer by select a genre mark from the legend. Using OSC-XR controllers we have been able to quickly prototype new methods for exploring and interacting with an immersive visualization.

While Unity, DxR, and OSC-XR are all used to build the immersive environment, other applications are needed to support it. T-SNE is implemented in Python and audio playback is implemented in ChucK. OSC communication affords us the ability to easily communicate between the distributed applications. In addition, OSC-XR also allows for communication within Unity by attaching OSC receiver methods to Unity `GameObjects` affording flexible and extensible event handling. By using OSC-XR, we are able to rapidly prototype an immersive environment with complex needs, such as toolkit integration and distributed communication.

6. CONCLUSION

We have introduced OSC-XR, a toolkit for prototyping immersive musical environments. By providing developers readily available controllers and scripts enabled with OSC, OSC-XR reduces the need to build control objects from scratch, making the development of immersive environments more accessible to researchers and developers. Combined with the power of Unity for building 3D environments, developers using OSC-XR are able to easily explore the affordances of immersive XR environments to find interactions for music control that would not be possible with other mediums.

The flexibility of OSC-XR creates many opportunities to further research on immersive music environments. First, we plan to implement features to spawn any controller prefab from within an immersive environment. This provides sound designers, with and without Unity development experience, the ability to build and customize immersive performance environments on the fly. Furthermore, to allow designers to take full advantage of the large amounts of data potentially created by such an environment OSC-XR would benefit from gesture learning capabilities, similar to those of the Wekinator [27]. Adding these features to OSC-XR will expand the possibilities of immersive performance environments and make designing them more accessible.

7. REFERENCES

[1] J. Lanier. (2019) Virtual instrumentation. [Online]. Available: http://jaronlanier.com/instruments.html

[2] S. Serafin, C. Erkut, J. Kojs, N. C. Nilsson, and R. Nordahl, "Virtual reality musical instruments: State of the art, design principles, and future directions," *Computer Music Journal*, vol. 41, no. 2, 2016.

[3] E. R. Miranda and M. Wanderley, *New Digital Musical Instruments: Control And Interaction Beyond the Keyboard*. A-R Editions, Inc., 2006.

[4] D. Wessel and M. Wright, "Problems and Prospects for Intimate Musical Control of Computers," *Computer Music Journal*, vol. 26, no. 3, pp. 11–22, 2002.

[5] A. G. E. Mulder, S. S. Fels, and K. Mase, "Design of virtual 3d instruments for musical interaction," in *Proceedings of the 1999 Conference on Graphics Interface*. San Francisco, CA, USA: Morgan Kaufmann Publishers Inc., 1999, pp. 76–83.

[6] T. Mäki-Patola, J. Laitinen, A. Kanerva, and T. Takala, "Experiments with virtual reality instruments," in *Proceedings of the 2005 Conference on New Interfaces for Musical Expression*, 2005, pp. 11–16.

[7] D. F. Berthaut, M. Desainte-Catherine, and M. Hachet, "Interacting with 3d reactive widgets for musical performance," *Journal of New Music Research*, vol. 40, no. 3, pp. 253–263, 2011.

[8] A. G. Moore, M. J. Howell, A. W. Stiles, N. S. Herrera, and R. P. McMahan, "Wedge: A musical interface for building and playing composition-appropriate immersive environments," in *2015 IEEE Symposium on 3D User Interfaces (3DUI)*, March 2015, pp. 205–206.

[9] N. Böttcher, S. Gelineck, L. Martinussen, and S. Serafin, "Virtual reality instruments capable of changing physical dimensions in real-time," *Proceedings of Enactive 2005*, 2005.

[10] D. Diakopoulos and A. Kapur, "Argos: An Open Source Application for Building Multi-Touch Musical Interfaces," in *Proceedings of the 2010 International Computer Music Conference*, 2010, pp. 88–91.

[11] C. Roberts, "Control: Software for end-user interface programming and interactive performance," in *Proceedings of the 2011 International Computer Music Conference*, 2011, pp. 425–428.

[12] R. Hamilton, "UDKOSC: An immersive musical environment," in *Proceedings of the 2011 International Computer Music Conference*, 2011, pp. 717–720.

[13] Y. Orlarey, D. Fober, and S. Letz, "Faust: an efficient functional approach to dsp programming," *New Computational Paradigms for Computer Music*, vol. 290, p. 14, 2009.

[14] P. Brinkmann, C. Mccormick, P. Kirn, M. Roth, and R. Lawler, "Embedding Pure Data with libpd," in *Proceeding of the Fourth International Pure Data Convention*, 2011, pp. 291–301.

[15] J. Atherton and G. Wang, "Chunity: Integrated Audiovisual Programming in Unity," in *Proceedings of the 2018 Conference on New Interfaces for Musical Expression*, 2018.

[16] Microsoft. (2019) Windows Mixed Reality. [Online]. Available: https://www.microsoft.com/en-ca/windows/windows-mixed-reality

[17] SteamVR. (2019) SteamVR. [Online]. Available: https://developer.valvesoftware.com/wiki/SteamVR

[18] VRTK. (2019) VRTK - Virtual Reality Toolkit. [Online]. Available: https://vrtoolkit.readme.io/

[19] T. Machover and J. T. Chung, "Hyperinstruments: Musically intelligent and interactive performance and creativity systems," in *Proceedings of the 1989 International Computer Music Conference*, 1989.

[20] D. Overholt, "The overtone violin," in *Proceedings of the 2005 Conference on New Interfaces for Musical Expression*, 2005, pp. 34–37.

[21] S. Trail, M. Dean, G. Odowichuck, T. F. Tavares, P. F. Driessen, W. A. Schloss, and G. Tzanetakis, "Non-invasive sensing and gesture control for pitched percussion hyper-instruments using the kinect," in *Proceedings of the 2012 Conference on New Interfaces for Musical Expression*, 2012.

[22] D. Johnson, I. Dufour, G. Tzanetakis, and D. Damien, "Detecting pianist hand posture mistakes for virtual piano tutoring," in *Proceedings of the 2016 International Computer Music Conference*, 2016.

[23] T. Dwyer, K. Marriott, T. Isenberg, K. Klein, N. Riche, F. Schreiber, W. Stuerzlinger, and B. H. Thomas, *Immersive Analytics: An Introduction*. Cham: Springer International Publishing, 2018, pp. 1–23.

[24] G. Tzanetakis and P. Cook, "Musical genre classification of audio signals," *IEEE Transactions on Speech and Audio Processing*, vol. 10, no. 5, pp. 293–302, July 2002.

[25] L. van der Maaten and G. Hinton, "Visualizing Data using t-SNE," *Journal of Machine Learning Research*, 2008.

[26] R. Sicat, J. Li, J. Choi, M. Cordeil, W. Jeong, B. Bach, and H. Pfister, "Dxr: A toolkit for building immersive data visualizations," *IEEE Transactions on Visualization and Computer Graphics*, vol. 25, no. 1, pp. 715–725, Jan 2019.

[27] R. Fiebrink, D. Trueman, and P. R. Cook, "A meta-instrument for interactive, on-the-fly machine learning." in *Proceedings of the 2009 Conference on New Interfaces for Musical Expression*, 2009, pp. 280–285.