polaris~

Project Presentation: TEI '21 Student Consortium

Project Outline: TEI '21 Student Consortium Paper

⟵ Software: Running Demos Software: Musical AR Instruments in PureData ⟶

Software: Planning Musical AR Instruments (July - August 2021)

Current state of the ARt

At this stage, I have a working visual AR headset... what about audio? As mentioned in the inspiration for this project I am using wireless bone conduction headphones, after trialling their use for audio AR in the area~ project. I use these because they serve as an aural equivalent to the method of visual mediation being used with the headset ('see-through' = 'hear-through'), that is to say, just as the headset does not occlude the sight of real physical objects in the participant's environment by using a half-mirror to combine virtual objects into the participants field of view, bone conduction headphones sit on the temple of the participant, rather than in the ear canal, and so they do not diminish the ability to hear sounds originating from real (not virtual) objects or processes in their environment.

The content for this headset and headphones is visually and auditorily assembled in three dimensions by the Unity engine in conjunction with real-time head and hand tracking data! What a sentence, and what a feat in itself -- the credit for which lays soley with those genius minds that have enabled this open-source North Star project in the first place!

"So what are you looking (listening) for?"

I am interested in creating expressive musical instruments that are situated in AR; that is to say, they are spatially and contextually relevant to their real world environment, not only the virtual environment they are necessarily spawned into.

"Ok, but how will you make musical instruments in a 3D game engine?"

At this point, I would prefer to draw from an audio engine or audio-focused programming language that I already have some experience in, not only because it would be more familiar, but also because there are a few technical difficulties in trying to realise everything in Unity (more on that later). Sticking to one engine/language for prototyping audio interactions from the outset would be preferable in order to maintain progress and proficiency

"How will you choose from Supercollider, Max MSP, PureData, Wwise, and FMOD amongst others?"

In order to plan for the long-term relevance of this project in creating interactable, tangible, and expressive AR musical instruments, the features that this audio engine or audio-focused programming language would need are:

1. Uses In-Built Unity Audio Spatialisation

Unity is a great 3D engine, with native (in-built) spatialisation of creatable elements called "GameObjects", that can have:

(1) visual attributes such as three-dimensional meshes, material colours, and textural properties

(2) physical attributes such as edges, position, mass, velocity

(3) auditory attributes such as a sound source with pitch and volume dependent on the physical attributes of position and velocity (things further away from the AudioListener component, which is assigned to the position of the headset, are quieter for example)

For these reasons, keeping audio "inside" Unity, and using its native audio spatialisation based on physical transform (x,y,z, velocity etc) values of the GameObjects is a feature that the project would benefit massively from in terms of labour. If instead I chose to

have a separate audio engine/server running in the background, it would have to have its own "knowledge" of the physical attributes of *all* of the GameObjects in Unity, and then spatialise everything itself. This (whilst possible) is probably outside of the scope of this project as long as an alternative is available that can exist "inside" Unity.

2. Cheap Instantiable Audio

The chosen audio engine / language needs to have the functionality of being able to create simple templates of audio algorithms that are then called into existence or "instantiated" tens to hundreds of times as the auditory output of certain GameObjects in Unity.

3. Extended Audio Techniques Available

It's not enough for the chosen audio engine/language to only have access to sample-based audio techniques (e.g. pressing a button triggers a pre-recorded sound file). It is definitely a feature I will need, as it is in cases more computationally "cheaper" to do so versus some synthesis techniques. However, due to the demands of real-time interaction in this project, and the aims of fine-grained expressive adaptation of musical textures, real-time synthesis will be one of the more often used techniques. More broadly and in the long-run, this project would benefit from being able to use as many audio techniques rather than having a cramped palette of 2 or 3 -- drawing from this early choice the ability to create a wide variety of AR musical instruments - each of them having their own nuanced interaction methods, creative expressivities, and modes sensory perception modulation.

4. Real-Time Parameter Control

Building on the last two features, the project would call for the ability to change in real-time, parameters of the audio algorithms being used. This would be the main source interactibility with an instrument, as having no parameters to control would likely mean that the instrument would not be instrumental in doing anything. Additionally, it would reduce the extent of co-adaptation of musical textures between participant and their instrument if there were not any parameters of the sound that were effectable in real-time. Therefore, this audio engine/language must have the ability to expose its own parameters to Unity via C# scripting (which is the native language of Unity projects) so that attributes and values inherent to the associated GameObjects can be mapped to have effects on the audio algorithms. This would make interactions such as touching an object (which is an event that happens in Unity) be able to have effects such as increasing the pitch of the sound coming out of the object, randomising the volume, or changing the scale (which would

be values inside the audio engine/language).

5. Allows Rapid Prototyping

In order to iterate quickly and efficiently on the design of instruments, whatever engine or language I am using must support quick changes to the structure and logic of the instrument and its audiovisual properties.

6. Open-Source, Free, and Cross-Platform

It would be preferable that this audio engine/language is open-source, free and cross-platform in order to be as inclusive as possible to a wide range of computational artists, designers, and developers wanting to implement and make their own AR musical instruments.

What to use then?

The following are only some of the available options, but are those that tick the most boxes feature-wise.

Unity's AudioSource Component

Rating	Notes
★★★★★ Uses Unity Spatialisation ★★★★★ Cheap Instantiable Audio ★★☆☆☆ Extended Audio Techniques Available ★★★☆☆ Real-Time Parameter Control ★★★☆☆ Allows Rapid Prototyping ★★★★★ Open-Source, Free, and Cross-Platform	No real-time synthesis means that instruments would be fairly simple sample triggerers with not much parameterisation available.

C# Granulator Synth via Granulator

Rating	Notes

	Granular synthesis only, would be a fairly limited set of options for building parameterised instruments.
★★★★★ Uses Unity Spatialisation	
★★★★★ Cheap Instantiable Audio	
★★★☆☆ Extended Audio Techniques Available	
★★★★☆ Real-Time Parameter Control	
★★★☆☆ Allows Rapid Prototyping	
★★★★★ Open-Source, Free, and Cross-Platform	

Supercollider

Rating	Notes
	No integration with Unity means two engines running (scsynth+Unity), and hence there would be a lot of work associated with attaining the same level of 3D audio believability that Unity provides natively. I would have to send all GameObject object, head, hand transforms to Supercollider and re-create the 3D engine environment before even getting to synthesis.
☆☆☆☆☆ Uses Unity Spatialisation	
★★★★★ Cheap Instantiable Audio	
★★★★☆ Extended Audio Techniques Available	
★★★★★ Real-Time Parameter Control	
★★★★★ Allows Rapid Prototyping	
★★★★★ Open-Source, Free, and Cross-Platform	

MaxMSP

Rating	Notes
☆☆☆☆☆ Uses Unity Spatialisation	
★★★★☆ Cheap Instantiable Audio	
★★★★★ Extended Audio Techniques Available	Same issue as Supercollider, but is also not free nor open-source.
★★★★★ Real-Time Parameter Control	
★★★★★ Allows Rapid Prototyping	
★☆☆☆☆ Open-Source, Free, and Cross-Platform	

Pure Data via Heavy Compiler and Wwise

Rating	Notes
☆☆☆☆☆ Uses Unity Spatialisation	Not being comfortable with Wwise means this is a poor option. It would also require using a separate 3D audio engine such as Resonance or Microsoft 3D Audio. It would be slow to prototype due to the time compiling patches into C for Wwise.
★★★☆☆ Cheap Instantiable Audio	
★★★★★ Extended Audio Techniques Available	
★★★★★ Real-Time Parameter Control	
★☆☆☆☆ Allows Rapid Prototyping	
★☆☆☆☆ Open-Source, Free, and Cross-Platform	

Pure Data via LibPdIntegration

Rating	Notes
★★★★★ Uses Unity Spatialisation	This is the way to go!
★★★★★ Cheap Instantiable Audio	
★★★★★ Extended Audio Techniques Available	
★★★★★ Real-Time Parameter Control	
★★★★★ Allows Rapid Prototyping	
★★★★★ Open-Source, Free, and Cross-Platform	

⟵ Software: Running Demos Software: Musical AR Instruments in PureData ⟶

Resources

Headset Documentation: Project North Star

Community: Project North Star Discord Server

Repository: Project Esky Renderer