Enhancing Digital Device Comunication Using Csound

Shirly Gurten and Bradley Ferringo*

[1] Magic Leap
sgurten@magicleap.com
[2] bferringo@magicleap.com

Abstract. UX sound designers are limited by the capabilities of pre-rendered audio interactions. Using Csound to develop a real time generated signal allows work in a spatial environment and produce better and faster result, with more accuracy and context. In this paper, we will discuss how we developed a custom synth for solving the limitations in composing for AR.

Keywords: UX , UI, Interaction Design, Augmented Reality, Cabbage, CsoundUnity, Unity, adaptive audio, reactive audio, edge compute.

1 Introduction

Digital devices have become an increasingly important part of human lives over the past 100 years. Along with the march of progress in technology has evolved an expectation of interaction schemes with these devices. Interaction schema would suited to their use, and feedback from said devices. Initial feedback systems were "dumb", meaning they were a mechanical bell or klaxon. As devices have grown more sophisticated, we began expecting devices to be "smart". This means that these devices are expected to understand context and communicate with us on a more sophisticated level. Through this progression, sound designers had to develop a language of interaction, a synthesized language that gives information about those devices. What began as simple tones from piezo buzzers developed into more advance skeuomorphic (an imitation of natural accruing sounds like a camera shutter) copies of the original "dumb" noises, evolved into a new wholly synthetic unidirectional language to convey information from machine to user [1] .The digital language evolved from pagers (that gave information about certain messages) Operative Systems (with affirmative or negative sounds) and monitoring devices, and created a standard to which users grew accustomed. Users developed an anticipation to receive certain information in a certain tone, with certain parameters[2] [3].

* The authors would like to extend a special apriciation and acknowledgement to Brett Shipes, Dr Richard Boulanger, Rory Walsh and Magic Leap for affording us the oportunity to write this paper.

Sound designers working on AR are tasked with designing new interactions constantly. Using synthesizers to create interactions is a well understood technique, as demonstrated in video games, phones and computers[4]. However, this technique has several key limitations when applied to AR. Firstly, it requires designers to work in a DAW, to render sound to a file, and to evaluate them again after implementation on an AR headset. The sounds are static, and no matter the context in which a user press a button, they will always sound exactly the same, because the played sound is produced from a static file. Taking the speaker output as a continual signal, the only options are "play" or "don't play".

2 What We Propose

Using *live* synthesis allows us to change the signal based on more parameters than just the aforementioned boolean. Interaction language may shift to give more information and more accurate information through our devices. This allows us to adapt the audio to gain more context and clarity. When using live synthesis directly through an AR headset, one is able to control the parameters of sound creation wile hearing the sounds in context at all time. It allows us to apply subtle shifts in timbre, frequency, waveform, and more- all of which create a more accurate and nuanced representation of the information being conveyed.

2.1 Csound Basis

Using Csound, we have developed a synthesizer that is designed specifically for dynamic and adaptive UI interactions. The purpose is to have a tool that will allow the creation of short, communicative sounds that devices can use to transfer information to users in more dimension than preceding systems. The first challenge was that these sounds must be short and precise- part of the expectation of UI sounds in the established UI language.This synth can to not only generate skeuomorphic audio, but extends those sounds to make more informative indications of UI behavior. The second challenge was the creation of a system that can transfer more information than just the existence of the interaction itself, and define better operate in the implicit language of adaptive UI audio. For example: when designing 5 buttons, each of which effects a menu differently, we want to be able to adjust the envelope to be longer or shorter based on the magnitude of change produced in the menu, such as a small click to change highlighted option, and a longer click to switch menu panes. Thirdly, the system must allow for different sound creation, with vastly different aesthetic qualities, that can serve many products and would be versatile for any UI system, such as those themed around elements like wood, glass, plastic, metal and any combination thereof[7]. Since different products will have different aesthetics, the sounds should reflect that. Lastly, we needed the ability to manipulate and shape sounds directly from an AR headset, to get a realistic auditory image with spatialization settings and context[8].

The adaptive Csound synthesizer strips the need to pre-render large sound files, effectively shrinking the compute and memory load dramatically. This is also in theory possible with integration of other COTS synthesizer packages, however, these packages are typically designed to run in DAWs and have their own large compute, dependency, and resource requirements, which far outstrip the small resource footprint of Csound. This style of extremely small footprint allows for the same interaction language scheme to be utilized everywhere from powerful desktops to computing at the edge, as Csound piggybacks on C libraries, which are common in edge processors

2.2 Envelope

The chosen envelope consists of only attack and release with sliders to manipulate A and R curves. When designing short interactions, precision is key. Creating short interaction require small, nearly indiscernible sound variations, and the precision of the envelopes needs to be high, especially for short time scales- on the order of milliseconds. The envelope sliders go from 0.1ms with a logarithmic curve to 1 seconds.Since Csound allows to call the envelope in a-rate, we could achieve a large variety of microscopic envelopes for immediate UI interactions.

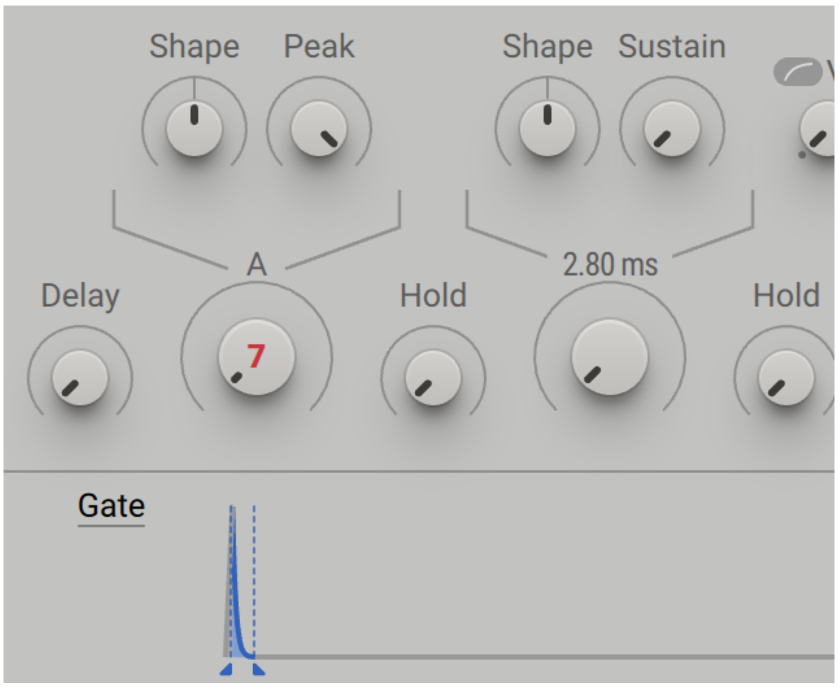

Fig. 1. Fastest Attack in Massive X is 0.0005 seconds, fastest release/decay : 0.0028

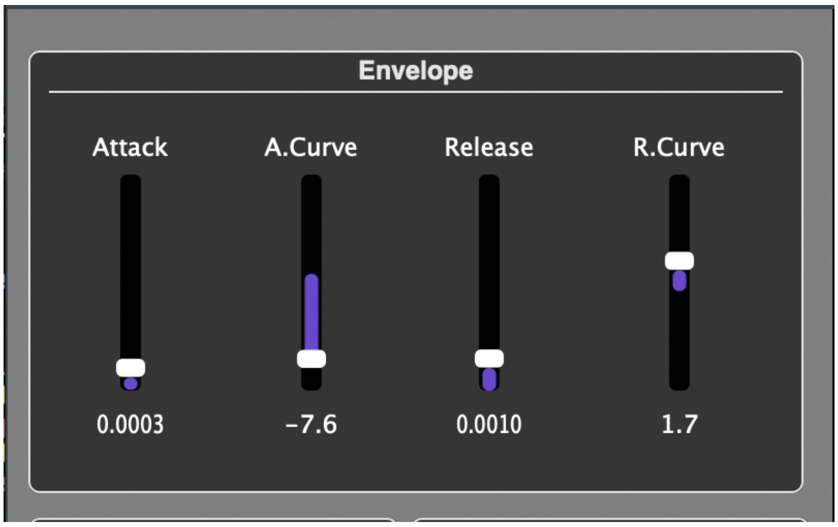

Fig. 2. Csound allows high accuracy in envelopes

Precision envelope allows the creation of tactile button interaction sounds. Those sounds are quick and impactful, short enough so users understand that this sound is a reaction to a behavior. Using a live synthesizer, build with Cabbage and CsoundUnity, we can change the envelope to range between extremely short to very short based on parameters imported from Unity. The result is an adaptive envelope based on layout/importance/distance elements.

2.3 Waveform

A combination of 2 oscillators, each one with a variety of options of different waveforms allow a variety of timbres of different interactions, from woody and round sounds to metallic and glassy sounds. To make the process easy and approachable, we added a dropdown menu with 6 different waveforms on each oscillator. With 15 different possible combinations, we can also control the level of each waveform and the frequency of them separately.

For volume control, we implemented 2 sliders, one for each oscillator. These balance each other to a maximum level of 1. So if 1 oscillator is all the way up, and the other is on 0, the volume would be 1, however if both sliders are all the way up - each one is being played in a volume of 0.5, summing to 1 total. Mixing between the 2 separate Oscillator allows the creation of more nuance sounds, especially when using noise generators.

2.4 ”Play!” button

To complete the module, we added a “play” button in the end of the module, to allow for testing before committing to a version. The button simulates the behavior of a UI element, on a standalone app, and can be perfectly translated to real button interaction in Unity. Since this tool is developed for UI sounds, the synth should represent that UI as interaction to get sound out of it. It is trivial for the designer to swap in and out different styles of play button, to better match the in-situ UX.

2.5 Extensibility

What has been described here is a bare-bones implementation to show the reader some of the possibilities of this concept. Csound contains one of the largest libraries of synthesis and signal processing routines, and benefits from a large community that allows for the sound designer to add extensions, effects, DSP, and anything else that the task may required [7]

3 Examples

3.1 Navigation Menus

One use case is a menu with nested sub-menus. With this synthesizer, we can create a condition where the sound becomes quieter and more filtered as a user navigates deep into the menu. The deeper the sub-menu is, the more distant the sound gets. That creates a feeling of depth that allows the user to more intuitively know where they are and how many times they to click “back” in order to get to the main menu. In this case, we adjust the filter level and the volume to imitate the human change in vocal behavior in the situation where a person is nearby vs. far away/distant.

3.2 Virtual Keyboard

When creating a digital keyboard, we rely on the tactile sounds of a mechanical keyboard be the main guideline for sound creation. Keyboards have complex systems with multiple functions beyond just adding a character; Shift, Alt, Return, Switch language, are just few examples for interactions a regular virtual keyboard can perform. While mechanical keyboards have an almost identical trigger below each key, a virtual keyboard would have a sound for each key that insets a character, and a different set of sound for every button that has a different interaction (like shift and caps lock). To avoid sound fatigue, we would add a small frequency randomizer, in a set of pre-defined range, as well as a volume shifter, to create the illusion of a real world mechanical sound with all its natural variations.

4 Benefits

Working with generative audio (opposed to pre-rendered sounds) allowed us more control over the sound creation, easier implementation, and fewer iterations on each sound. Instead of changing the pitch of a prerecorded sound, that can create artifacts in the sound file[3] we can now adjust the frequency or the mix balance without rendering another file, or applying post effects to manipulate the audio. The audio interactions can be richer because they are being updated based on user defined input, and the context give users more information about the sound that is being created. This also dramatically shortens the iterative design loops of "Make-implement-test" by functionally removing the "implement" phase. Designers using this paradigm need only change a slider and click play, as is common in more user-centric DAWs.

We see this as a *leap* towards better communication between devices and their users. Due to memory limitations, slow design processes, and difficulty in implementation, the space of modulation of device sounds for informative feedback has essentially been neglected, and we aim to rectify that, utilizing csounds' low-memory capabilities, live parameter control, and single-site implementation, we seek to open this space so that a button is never just a button.

References

1. Kuang, Cliff, and Robert Fabricant. "Humanity." User Friendly: How the Hidden Rules of Design Are Changing the Way We Live, Work, and Play, Penguin, London, England, 2020, pp. 195–195.
2. Sweet Anticipation: Music and the Psychology of Expectation, MIT Press, Erscheinungsort Nicht Ermittelbar, 2008, p. 324.
3. Dunne, Anthony. "Chapter 1." Hertzian Tales: Electronic Products, Aesthetic Experience, and Critical Design, MIT, Cambridge, MA, 2009, pp. 16–16.
4. Farnell, Andy. Designing Sound. MIT Press, 2010.
5. Avarese, John. "Sound Design ." Post Sound Design: The Art and Craft of Audio Post Production for the Moving Image, Bloomsbury Academic, an Imprint of Bloomsbury Publishing Inc., New York, NY, USA, 2017, pp. 75–75.
6. R . Boulanger. The Csound Book. MIT Press, Cambridge, Mass.
7. Steven YI and Victor LAZZARINI . "Csound for Android" National University of Ireland, Maynooth
8. CsoundUnity Github site, `https://github.com/rorywalsh/CsoundUnity`

[3] ,