Music 220D Report

Mayank Sanganeria
Fall 2012,CCRMA
Stanford University
Advisor: Julius O. Smith

December 17, 2012

1 Introduction

For my 220D this quarter, I wanted to explore Web Audio and Faust. To this end, I started learning JavaScript and Faust in parallel. While exploring Web Audio, I also came across the NaCl system built by Google for Chrome and explored that as well.

2 Faust

Faust (Function AUdio STream) [5] is an audio signal processing language. Building audio processing units straight from a block diagram form instead of having to translate everything into traditional code was extremely exciting and enjoyable. The fact that Faust exports to almost all conceivable platforms makes it especially useful. I tried out the WebKit version and was very impressed by its performance. It is very exciting that Faust could be present in a future version of WebKit.

Since granular synthesis lends itself more easily to an implementation in code , I thought it would be an interesting exercise to build a granular synthesis unit in Faust. I couldnt find any granular synthesizers built in Faust online which was encouraging.

2.1 Granular Synthesizer

I implemented a real time granulator, as defined by Curtis Roads [7]. The granular synthesizer can be accessed at Github [8].

The most challenging aspect of getting the granular synthesizer to work in Faust was getting an uncorrelated random number generators working for each grain line. I was unable to simply use a single RNG function, as one would do in C code, since a new random number was generated only once every sampling period where n random numbers were required. I tried creating functions that depended on i for the i^{th} grain but they formed determinstic patterns and provided a repeating rhythmic pattern which was undesired. I settled on using the multi-channel noiser as given in the Faust tutorial [6] to get n different white noise generators and used a sequence macro to extract the i^{th} channel from the noiser to use for the i^{th} grain line.

A sample-and-hold was used to randomly select grain positions. With noise as input, and a trigger that was activated once the grain had finished playing, the sample-and-hold provided a new start position for the grains at the right time.

The number of grains was made controllable by defining a maximum number of grain lines and having all of them run in parallel but multiplying them with a conditional to silence all the unwanted grain lines.

3 JavaScript

On Myles Borins advice, I read Douglas Crockfords book [3]. I had read a few other JavaScript books earlier but they were all just reference books and never addressed good style. Crockford's book was really useful in terms of dealing with JavaScripts quirks and leveraging those to write solid code.

Having done that, I thought that it would be useful working on a JavaScript library to get experience in JavaScript architecture and decided to work on Colin Clarks Flocking library [2] and contribute a granulator to the library, since he was looking for a granular synthesis ugen and I had experience in building granulators before.

I made a granulator ugen [9] for his library that Colin Clark was happy with and will be incorporating into Flocking. The pull request is currently pending code review.

In his article on granular synthesis architectures [1], Ross Bencina dis-

cusses a lot of different architectures for implementing real time granular synthesizers. The architecture where the grains are triggered using a separate process (mathematical function or random function) was particularly interesting and I am excited about implementing and listening to it.

4 NaCl

Based on my interest in Web Audio, Julius Smith suggested exploring Google s Native Client (NaCl) [4] as an alternative method to having audio on the web. NaCl allows Chrome to run pre-compiled C++ code straight out of the browser. The NaCl SDK compiles C/C++ code to an *.nexe file format which can be included on an HTML page and communicates with the HTML file using messages. This allows access to computation that runs at native speed on web pages. The advantage to having this system is that it allows high speed computation for all purposes audio, graphics and anything coded in C/C++. The downside, though, is that it is supported only in Chrome and will not be supported by any other browsers in the conceivable future. Even the support in Chrome is experimental currently and requires a flag to be manually set before NaCl applications can be run.

I ran some very simple examples with GUIs on NaCl. The system works really well but the lack of support on other browsers takes away from the utility of everything it enables. The support on the iOS version of Chrome is also lacking at the moment.

5 Future Work

Having experience with Faust will help tremendously while taking Physical Audio Signal Processing next quarter. I have also been implementing some other audio effects in Faust independently and then comparing with the included code to increase my fluency in the language.

I will also continue to contribute different architectures of granular Synthesis to Flocking as well as Faust.

References

[1] R. Bencina. Implementing real-time granular synthesis. http://www.rossbencina.com/static/code/granular-synthesis/BencinaAudioAnecdotes310801.pdf, August 2001.

[2] C. Clark. Flocking. https://github.com/colinbdclark/Flocking.

[3] D. Crockford. *JavaScript: The Good Parts*. Yahoo Press, May 2008.

[4] Google. Native client. https://developers.google.com/native-client/.

[5] Grame. Faust. http://faust.grame.fr.

[6] Grame. Faust tutorial 2. http://faust.grame.fr/images/faust-doc/Faust_tutorial2.pdf.

[7] C. Roads. *Microsound*. MIT Press, January 2004.

[8] M. Sanganeria. Faust granulator. https://github.com/e7mac/faust-code/blob/master/granulator.dsp, December 2012.

[9] M. Sanganeria. Flocking granulator. http://github.com/e7mac/Flocking/blob/master/flocking/flocking-ugens.js#L1716-L1827, December 2012.