Computers, Sound and Music: Project Report

Project: Moon Forge

Group: Brian Ginsburg

Project repo: https://github.com/bgins/moon-forge

What was built

For my term project, I built a polyphonic web audio subtractive synthesizer. The sythnesizer includes one oscillator, an amplitude envelope, one filter, a filter envelope, and master gain control.

The synthesizer can be played on your computer keyboard or with a Midi controller. Midi control only works in Chrome or Chromium because they are the only browsers that have implemented WebMidi so far.

The synthesizer can be retuned to arbitrary divisions of the octave. Full control over number of divisions, reference frequency, and Midi note at the reference frequency have been implemented.

Two side efforts arose while working on this project. The original plan included compiling Faust code to web audio, but the current version of this project does not include any Faust. Experiments in compiling and loading Faust to web audio can be found here: https://github.com/bgins/faust-web-examples

I wrote an envelope generator for this project and kept it as a separate module: https://github.com/bgins/env-gen

How it worked

The project has two main parts. The user interface is implemented in Elm with the elm-ui library. Audio and Midi are written in TypeScript and compiled to JavaScript. Instrument settings are maintained by Elm and TypeScript, but can only be changed on the Elm side. This arrangement was chosen to use as much Elm as possible. This stops at audio and Midi because they are not available in a reasonable fashion in Elm at the moment.

Midi controls depend on webmidi module and are implemented in `src/controllers/midi.ts`. Computer keyboard controls use mousetrap-ts and are implemented in `src/controllers/keyboard.ts`.

The audio uses standardized-audio-context, an excellent project that aims to fill in gaps in browser implementations of web audio and to provide a TypeScript interface. The synthesizer audio is implemented in `src/audio/luna.ts` and with some types in `audio.ts`. The envelopes are generated in `src/audio/env-gen.ts`.

Overall, I am fairly happy with how it all works. A fair bit of work went into user interface and setting up controllers, but I think it was worthwhile since it looks nice and it's fun to play it with my Midi keyboard.

What doesn't work

Luna still has some clipping when the gain is turned up high. I added a limiter and lowered the overall gain, both of which helped, but were not enough to get rid of all the clicks and pops. Web audio provides a compressor that can be set to act like a limiter, but it may not be sufficient. This may be a good place to use a Faust module.

The envelope generator is not able to update a note with new settings on retrigger. This can be annoying with a long release because you may think a note is done, go to change the settings and not hear any changes. I think it is possible to patch in the new settings, and I intend to fix this.

The types can be strengthened a bit on the TypeScript side. For example, `flags` really should have a type to give us some guarantees.

The user interface is a bit small on some displays. I am alright with Luna being small because it will need to fit into a larger DAW eventually, but the side panel can definitely be bigger.

One of the goals of this project was compiling Faust code to web audio. As I was developing the synthesizer, I couldn't see a good place to include Faust code. One possibility was to use it to implement some effect such as reverb, but alas, I didn't quite get that far.

What lessons were learned

I learned quite a bit about instrument design and web audio working on this project. Web audio provides a fairly high level API for many tasks, but some parts demand that you think at a lower level.

A monophonic synthesizer could be implemented with an `OscillatorNode` and a `GainNode` connected to `AudioContext` destination (your speakers or headphones). Adding polyphony calls for some means to track active notes (in my implementation an array) and a way to gracefully handle notes that are retriggered. Adding envelopes and filters means adding more structures grouped together for each note event. Taking each of these one step at a time was valuable for debugging.

I probably learned the most about audio implementing my own envelope generator. The tricky bit is handling retriggers. What you want with a retrigger is that the note will start the attack phase at its current value and avoid jumping to zero. It is not ideal to query the current value in web audio because it is running on another thread, and by the time you get the value it will have already changed. This means that you calculate what the value should be given the current time. Then you calculate the when the attack phase would have started to determine the correct linear ramp. Finally, all that math you were doing takes time and you need to account for that too, but you really can't, so you calculate all of that stuff far enough into the future (but not too far!) so that it doesn't matter. At least that is where I got with it, but I can't claim I have all the answers!

I also learned non-audio things, mostly around integrating Elm and TypeScript. It all works out pretty well, though there is a fair amount of boilerplate making sure both sides are happy with the data and how it is typed.