Thirty Years of Sound Hacking: From freeware to Eurorack

TOM ERBE

University of California San Diego, United States
Email: tre@ucsd.edu

For slightly more than 30 years I have been developing audio software and hardware under the moniker *soundhack*. Through these years I have programmed applications, plugins, externals, hardware and Eurorack modules – usually focusing on signal processing techniques and applications that are not easily available or offered by mainstream software companies. In this article, I would like to share my point of view and relate the rationale behind the development of these tools, my evolving sonic and design aesthetic, and some of the advantages, disadvantages and other differences between the the various hardware and software contexts.

1. SOUNDHACK

1.1. Robert Ashley and the beginning of SoundHack

In the spring and summer of 1989 I was engaged as a recording engineer and electronic musician (along with David Rosenboom, Tom Hamilton and Sam Ashley) for the recording of Robert Ashley's opera *Improvement* (Ashley 1992). We worked at the recording and electronic music studios of the Center for Contemporary Music at Mills College. One of my main tasks was to develop unique sonic identities for each of the characters in the opera. With our 'studio as an instrument' (a Lexicon PCM70, Drawmer dynamics processor, Alesis Quadraverb, mixer, 24-track and Opcode Vision to automate parameters), David Rosenboom and I were to develop vocal treatments and backgrounds specific to each scene and to each performer. For instance, the protagonist Linda's voice (played by Jacqueline Humbert) was recorded with multiple layers: spoken, sung, echoed and spatialised. This represented both her constant internal dialog and her mnemonic system in 'The Contents of Her Purse'. The character Don (played by Thomas Buckner) was a tap dancer, hence a layer of Tom's vocals containing only plosives and fricatives that I filtered and gated for more percussive effect. When developing the treatments for the Greek chorus and scene 2, 'The Airline Ticket Counter', I needed to create specific spaces and resonances that followed the score and conveyed the location, the relation of the characters and any subtext. I realised that to sculpt this sound I really would have liked to use sound-file convolution for cross-synthesis. Unfortunately:, the CCM did not own an expensive Unix workstation to run the CARL software distribution (Loy 2002), and my attempts to quickly port *Mark Dolson's convolvesf* application to a Macintosh System 6 were too difficult to complete while following our 60 hour a week recording schedule. Instead, the airport was created with reverb, live equalisation and panning, and the Greek chorus modulated a gated Buchla Touché drone providing the pitched resonance. When production moved from Oakland to NYC in the fall, and Tom Hamilton took over my role, I finally had time to create the tool I no longer needed for *Improvement*, but wanted regardless. I bought a book on programming with Think C and the Mac Toolbox, and started work on *SoundHack* (Figure 1).

I released the first version, *SoundHack 0.1*, two years later in October 1991. I named it *SoundHack* as I felt electronic music-making on small home computers had some affinity to the then-present hacker culture. Version *0.6* included sound file conversion, convolution and the phase vocoder. From this version, the software became used widely by a diverse group of musicians, educators and sound designers: from Alvin Curran to Richard D. James, from the Sal Soul Orchestra to the sound designers for *The Matrix*.

1.2. Motivations

My main motivation for programming *SoundHack* was to make new computer music techniques easily available to the experimental musician community. I became aware of much of the interesting research in computer music during my time as a research assistant at the Center for Music Experiment at UC San Diego (1984–87). Not only was I excited by the sonic possibilities of these techniques, but I was also excited at what sonic creations experimental and electronic musicians might make with them.

There was also a very practical motivation for the development of *SoundHack*. At the time, the tools that we used all had their own sound file formats. For example, *Csound* used the IRCAM file format, *Sound Designer II* used the SD2 format, the AIFF format had just been announced (Apple Computer 1989).

Organised Sound **27**(1): 20–25 © The Author(s), 2022. Published by Cambridge University Press. doi:10.1017/S1355771822000176
This is an Open Access article, distributed under the terms of the Creative Commons Attribution licence (http://creativecommons.org/licenses/by/4.0/), which permits unrestricted re-use, distribution and reproduction, provided the original article is properly cited.

https://doi.org/10.1017/S1355771822000176 Published online by Cambridge University Press

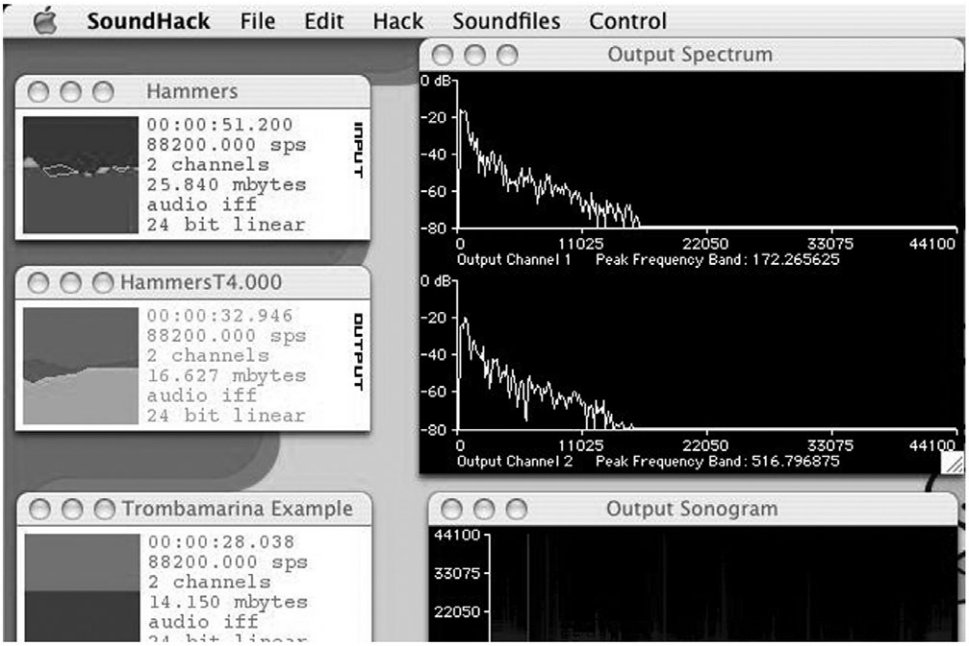

Figure 1. *SoundHack* application: 1991–2021.

SoundHack could fill these gaps and allow one to use multiple music software as well as hardware samplers. We were far from applying 'studio as an instrument' techniques to the computer, but it was a start.

There were no design or research motivations to *SoundHack* at this point. The focus was simply sound, synthesis and music.

1.3. Collaboration and community

The development of *SoundHack* led to a number of interesting collaborations. The binaural filtering processor was the result of working with Durand Begault at NASA-Ames Research Center. We incorporated one HRTF oriented towards accurate spatialisation, and another oriented towards better music fidelity as well as developing a method for position interpolation. My collaboration with Larry Polansky applied his morphological mutation functions to spectral data (Polansky and Erbe 1996), and a conversation with Zack Settle led to the Spectral Extractor module – a method of separating pitched and unpitched sound.

SoundHack was also increasingly used by the academic music community. I found it used at any school with a curriculum in electroacoustic music, and I took many suggestions from my peers at other institutions. Ideas from fellow academics included a spectral file format with examples on how to use it, support of *Csound* file formats, variable parameter functions and the PEAK chunk in AIFF files. As mentioned earlier, I also was hearing *SoundHack* in many commercial contexts: time stretching on the Hafler Trio's *An Utterance of the Supreme Ventriloquist* and Duran Duran's *Medazzaland*, or convolution as a skyscraper froze on *The Day After Tomorrow*. Hearing my work in these places was satisfying, but what really became valuable to me was the community of experimental musicians that emerged around *SoundHack* and other music freeware such as *Thonk, Mammut, nato.0+55+3d, Metasynth* and *Argeïphontes Lyre*.

1.4. Winding down

My work on *SoundHack* slowed down for several reasons. At the end of the 1990s, I found that the program had grown too large to maintain as it had little overall internal design or growable structure. I was contemplating incorporating an internal scripting language, but soon found that to do this most everything in *SoundHack* would simply need to be reprogrammed. I learned the hard way that any such scriptable or patchable structure needs to be designed at the beginning of a project. Second, before Steve Jobs returned, there was a lot of speculation that Apple might go out of business. I was looking for a new platform for audio processing. I spent two years programming a new Java-based *SoundHack* DAW, but had to abandon this as I found that the application grew less responsive the larger it became (likely this would no longer be the case with current versions of Java). I was getting lost in the weeds of programming options, but thought I should move to something more modular and fixed.

https://doi.org/10.1017/S1355771822000176 Published online by Cambridge University Press

Hence, the idea of programming plugins became quite attractive. Steinberg had published the VST spec (Johnson and Poyser 1996), which seemed well thought out and I could contain each idea in an individual project and experiment without affecting other work. This was the fresh path.

SoundHack became dormant. I kept it up to date through the transition to OS X and released the last version in 2007 (using Apple's 32-bit CarbonLib to keep System 9 code working under OS X). Sadly, with the elimination of 32-bit support in Big Sur, it is no longer runnable. On the other hand, 29 years is not a bad lifespan for a piece of music shareware.

2. SOUNDHACK PLUGINS

My immediate intention was to recreate all the features of SoundHack as VST plugins for both Mac and Windows. I soon found that would be difficult. Many of these processes do not have a one-to-one relation between input and output time and cannot run in real time. Second, not all hosts could take side chain inputs (inputs other than the main audio input), so cross-synthesis could not easily be supported. So I set aside the convolution, phase vocoder, mutation and varispeed processes, and went ahead with the remainder: spectral dynamics, spectral extraction and the binaural filter. I was confident that offline processing and side chain inputs would be well supported in the future.

My first collection of plugins, the *Spectral Shapers*, was released in 2003, and contained *binaural*, *spectralgate*, *spectralcompand* and *morphfilter*. Dividing the processes into separate plugins really allowed me to focus on each individually. This resulted in a general improvement of sound, a better interface design, and a better and more functional parameterisation. You can see this clearly in the evolution of the binaural interface from *SoundHack* to *Spectral Shapers 1.0* to *Spectral Shapers 2.0* (Figure 2).

2.1. Studio needs, distortion and delay

Concurrent with the development of *Spectral Shapers*, I developed plugins to address what I thought was missing in the digital studio. In the late 1990s and early 2000s, I found many of the plugins available too clean and controlled. I created *decimate* in 2000 to add bitcrushing, and *chebyshev* to add variable harmonic enhancement to instruments without alerting the musician that I am adding distortion to their track. I created *matrix* and *compand* to do mid-side compression and expansion, and thus help me in my role as a mastering engineer. I started working on delay plugins in 2004, as I wanted to show my recording class some effects with more character. This was the motivation behind the delay trio, to create three delay effects (modulated delay, pitch shifting delay and granular delay), each with a very distinct personality. I found that by adding additional processing elements in an effect it is easy to achieve a wide range of sound: clean to dirty, bright to dark, straight-ahead to weird. Once the structure is set, much of the work is to tune the parameters so that the extremes are balanced, and to make sure the combination of all the changing parameters contain many sweet spots. Specific effects can be given personality or colour by adding specific elements. In delay algorithms, adding the right mix of filters, compression and saturation in the feedback path can have a dramatic effect, and certain applications such as comb filtering, flanging or chorusing need different internal adjustments of these submodules to sound their best. With so many parameters it is difficult to build a usable user interface. One could expose every internal control of every module to the user, but when faced with over 10 parameters to adjust, most musicians will have a hard time finding the sound they want. After I finished programming *bubbler*, my granular delay, I felt I had reached the limit of usability with 12 controls and even more buttons (Figure 3). In present designs, I try to combine internal parameters to create more powerful meta-parameters and to minimise the number of controls that only have a subtle effect.

2.2. Bypassing the real-time time stretching issue

My most recently released plugin bundle is a set of phase vocoder based plugins called the *Pvoc Kit* (Erbe 2011). I was hoping that offline processing would become more universally supported among plugin hosts, but in the meantime I decided to work around the issue. In the *Pvoc Kit* there is a time stretcher, *spiralstretch*, that runs in real time by time compressing first, that is, it captures small grains of the incoming sound and overlaps the time stretched version of each grain on the output. The result sounds time stretched, but is the same length. The second time stretcher in this bundle simply holds the sound file internally with full control of playback rate. These work-arounds point to one of the problems in plugin development. One does not have control of the plugin software development kit (SDK) or the plugin host, and because of this there will always be certain types of sound processing or synthesis that one cannot do.

2.3. Supporting plugin updates

The constant problem with plugin support is the perpetually changing computing environment. The plugin host will likely change every year, and often this will break your plugin. The computer OS or OS

https://doi.org/10.1017/S1355771822000176 Published online by Cambridge University Press

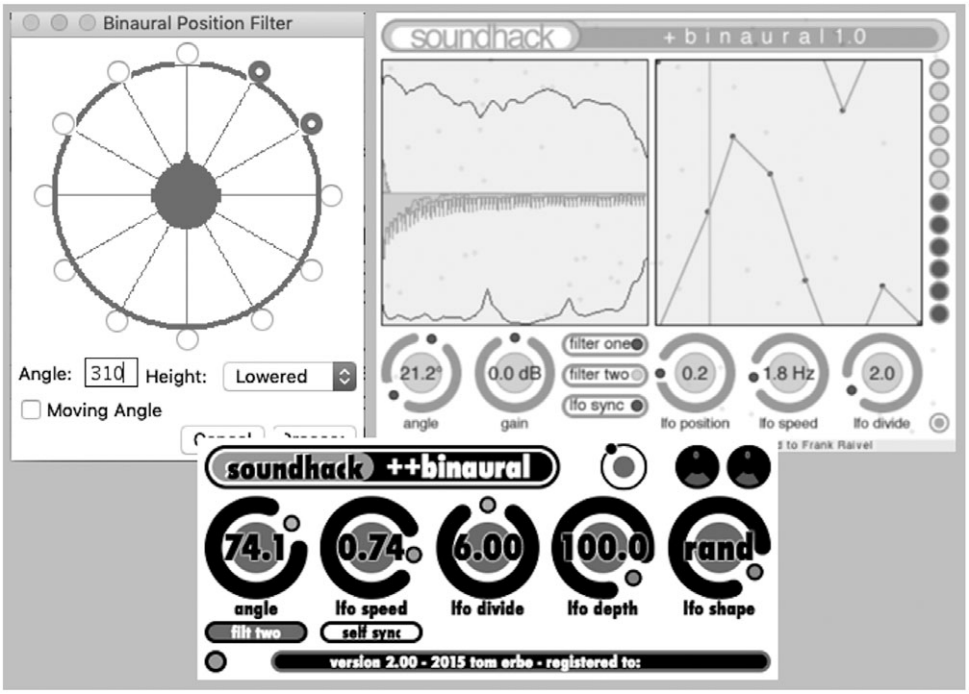

Figure 2. *SoundHack* binaural (top left), *Spectral Shapers 1.0 +binaural* (top right), *Spectral Shapers 2.0 ++binaural* (bottom).

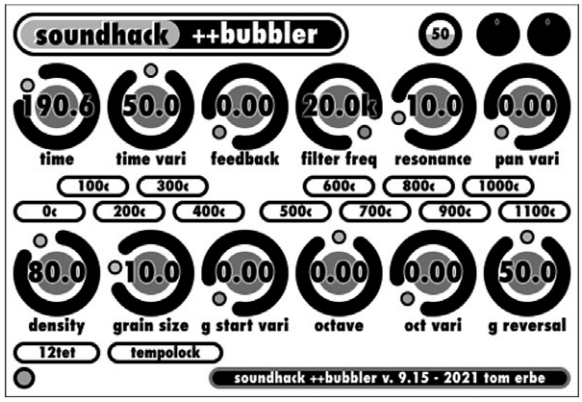

Figure 3. *SoundHack ++bubbler*: my upper limit for control density.

requirements will also change every year. Even if you have made no improvements to your plugin, it will need to be recompiled and tested at least once a year. Presently I have 17 plugins published that run on macOS as VST, AU, AAX and VST3 and on Windows as VST and VST3. This means for every revision I need to compile, test, package and upload 119 pieces of software. If there was only just one universal plugin format and computer companies made reverse compatibility a priority, individual music software developers could focus more on new sonic techniques. Cross-platform APIs like JUCE (Raw Material Software Limited 2021) help the situation greatly, but plugin support is still a huge time sink. The plugin format has the advantage of cross-platform compatibility and modularity, but the lack of control of the computing environment makes it much less attractive.

3. BACK TO HARDWARE: SOUNDHACK EURORACK MODULES

For the past 10 years, the cost of hardware development, high-quality components and construction has been low enough that custom hardware development has been accessible to many people. There are low-cost development boards from Teensy, beagleboard, ElectroSmith and others that provide easy starting points for hardware programming. One great advantage in programming hardware is that the environment is under your control. However, control comes at a cost. There is often very little in software library support. You will be responsible for setting up all of the hardware components (e.g., RAM, CODEC, DMA) in detail. You will not have many examples to help you. Finally, sharing your work can be complex or costly. A DIY/open-source project requires the user to have a certain amount of expertise in building, and often one has to devote a lot of time to user support. A commercial project obviously carries its share of risk. That said, it is extremely satisfying to

https://doi.org/10.1017/S1355771822000176 Published online by Cambridge University Press

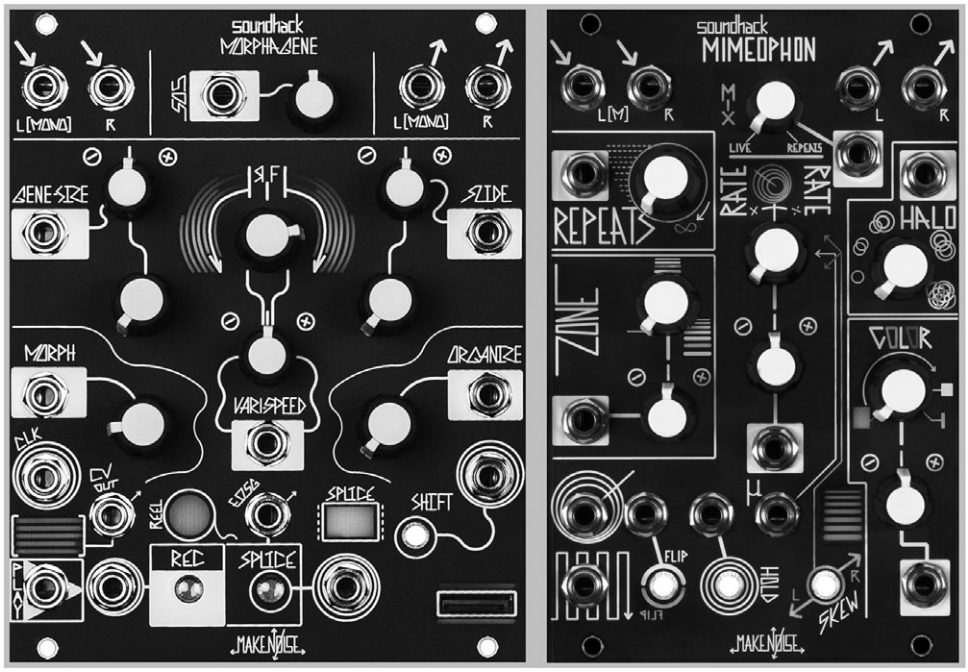

Figure 4. Make Noise/SoundHack: *Morphagene* and *Mimeophon*.

create a device that has a physical existence, that has buttons, dials and indicators with dedicated parameters, and can be played on its own.

The past 10 years I have collaborated with Tony Rolando of Make Noise Music in the design of a series of Eurorack synthesis modules. So far our collaboration has resulted in *Echophon*, a pitch-shifting delay, *Erbe-Verb,* a very flexible and complex reverb, *Telharmonic*, an additive synthesis oscillator with minimised timbre controls, *Morphagene*, a looper with extensive granular options and live-splicing, and *Mimeophon*, a very clean and lush delay with complex filtering (Figure 4). I will not go into the specific development of each module, but I will share the general design process and guidelines Make Noise and I have developed during this collaboration.

3.1. Design process

The design process has been consistent throughout most of these projects. First, we have a collaborative brainstorming session in which we relate all our new ideas for projects. Any of the potential projects are then analysed to determine whether they can be implemented in full with current technology, and to see if we can design it to be more interesting or musical than other instruments of the same type. Also, as these are modules, we look at how they may interact with other modules. An echo device may gain an external feedback path to patch into other processors, an oscillator will need to work well with external source of FM, and most all control-voltage parameters will need to respond well to audio and control rate modulation. At this point I conduct research on similar instruments, current and past, in order to gain a thorough knowledge of the sonic effect of their algorithms and implementation. This leads to a synthesis of our initial idea and the subsequent research, and a prototype is built for alpha testing by a group of musicians and engineers. If the prototype is too hard to use, too bizarre, unmusical, one dimensional, or underwhelming, I take this feedback and try to address the critique, while being careful to maintain my vision for the project. Finally, the project undergoes weeks of beta testing as hardware is expected to be bug-free on release, This is unlike software, which will have many subsequent updates and fixes.

3.2. Design guidelines

First, your parameters should affect the sound, not just the algorithm. Find natural combinations of internal parameters that can be mapped onto a single control. For example, a reverb might have a liveness control that actually controls the feedback filtering, diffusion and output filtering. A filter might retune resonance or slope when you change frequency. Give your parameter a balance of beautiful and ugly sounds. The extreme settings should be too much for most cases, 80 per cent of the control should be the settings one uses the most.

https://doi.org/10.1017/S1355771822000176 Published online by Cambridge University Press

Second, complicate the internal structure. An oscillator with a wave-shape control has one dimension. Add a tilt or spectral flux control and you have squared or cubed the timbre-space.

Third, try to avoid modal user interfaces. If you find you have modes, see if there is a way to morph between modes (and hence, one mode). If you have to have modes, do not enter and exit modes with complex button combinations.

Fourth, limit the number of controls to those that make a significant change. Buttons, jacks and potentiometers add a lot to the cost. In addition, keep enough room around them so that it is playable.

Finally, good sound is primary. Every position of every dial should create compelling sound. Overloads, over modulation and so on should be handled in a way that does not detract from the sound. If you reach the point in your design where you think it sounds good, you are not finished. If you reach the point that you cannot stop playing the device, then you are getting close. There is often room in your design for more musical flexibility or an additional sonic twist.

REFERENCES

Apple Computer. 1989. *Audio Interchange File Format, A Standard for Sampled Sound Files, Version 1.3.*

Erbe, T. 2011. PVOC KIT: New Applications of the Phase Vocoder. *Proceedings of the 2011 International Computer Music Conference*, University of Huddersfield, UK, 31 July–5 August, 143–6.

Johnson, D. and Poyser, D. 1996. Steinberg Cubase VST. *Sound On Sound*, July. www.soundonsound.com/reviews/steinberg-cubase-vst (accessed 9 July 2021).

Loy, G. 2002. The CARL System: Premises, History, and Fate. *Computer Music Journal* **26**(2): 52–60.

Polansky, L. and Erbe, T. 1996. Spectral Mutation in SoundHack. *Computer Music Journal* **20**(1): 92–102.

Raw Material Software Limited. 2021. *JUCE 6.0.7*. https://github.com/juce-framework/JUCE/releases/tag/6.0.7 (accessed 9 July 2021).

DISCOGRAPHY

Ashley, R. 1992. *Improvement (Don Leaves Linda)*. Nonesuch, 79289-2.

https://doi.org/10.1017/S1355771822000176 Published online by Cambridge University Press