BSOUND MANUAL

Welcome to the BSOUND MANUAL. I hope that the graphical user interface (GUI) I introduced in BSOUND 0.2 makes BSOUND fairly self-explanatory, but I hope this can serve as a reference and guide to how BSOUND works, whilst also explaining some fundamental concepts of Digital Signal Processing.

Currently BSOUND sports the following effects:

DELAY

delay tape
pingpong multitap

GRANULAR SYNTHESIS

cloud shimmer
transpose

PHYSICAL MODELING

reverb pedal

MODULATION & DE-MODULATION

moddemod ringmod

SAMPLERATE REDUCTION

crush

GENERAL OPERATION

Before you start BSOUND, please make sure that the appropriate input and output devices have been selected in “System Preferences -> Audio”. These settings are passed on to BSOUND and there is currently no way of changing them reliably once BSOUND is running. Also, BSOUND will the default Terminal window settings, so please go to “Terminal -> Preferences” to change the size and style of the BSOUND window *before* you open BSOUND.

When you open BSOUND you should be prompted about loading a previously saved state. In its save slots, BSOUND remembers all of the parameters and the order of effects, and you should find a couple example FX pre-loaded to get you started.

If you choose not to load from a save slot you should find a blank template. Typing ‘:’ will allow you to add effects and all available effects will be listed on screen. By default, a new effect is added immediately after the currently selected effect. To add effects to the beginning of the signal chain type ‘push’ + ‘FX name’. To delete individual effects type ‘delete’ and then choose the item to be deleted and hit ‘enter’. ‘Clear’ deletes all effects and brings you back to a blank palette.

Beware that there is currently a hard limit to the number of effects you can chain together. This has less to do with processing power and more to do with an aesthetic choice: Long FX-chains tend to be more unpredictable and more prone to degrade into chaotic overdrive or suddenly drop out altogether.

Navigating the effects chain is done with the ‘A’ and ‘D’ keys. The arrow keys are used to edit an effect’s parameters: UP and DOWN choose which parameter you are currently editing, and LEFT and RIGHT increment and decrement the parameter. Fast scrolling in steps of ten is available with ‘SHIFT’. Not all parameters respond equally well to fast changes, and some changes may take a moment to be heard.

The ‘B’ key will toggle ‘BYPASS’. When ‘BYPASS’ is enabled, ’BYPASS’ will appear in the upper right-hand corner, and the effect’s chain is bypassed and BSOUND will ignore incoming audio. Since BSOUND is intended for use with acoustic instruments, the input audio is not available at the output (as it would be for a guitar effect). Instead, BSOUND will continue to process audio that is available internally through feedback or similar mechanisms, and will output this ‘spill-over’.

A similar mechanism is available on a per-effect basis via the parameter ‘skip’. Setting skip to any value other than 0 will pass some of the output of the selected effect directly to the output, bypassing the remaining FX-chain. If ‘skip’ is active, a small ‘icon’ will appear over the effect so that the signal flow remains clear at first glance.

Saving a BSOUND state overwrites the currently selected slot. There is currently no auto-save, so what you don’t save is lost for ever (or until you recreate it).

The preferences page (available by typing ‘:’ and ‘preferences’) allows you to map a mono input to all output channels -- otherwise this signal will simply appear on CHANNEL 1. Furthermore, it allows you to change the BUFFER size. The buffer size changes the minimum amount of lag inherent in the signal chain. Low settings will produce the shortest lag, but might result in input overflow or output underflow. These errors will appear on screen and you may hear audio cutting out momentarily. Changing the buffer size may only take effect after BSOUND has been restarted.

The current settings are automatically saved and recalled on startup.

DELAY OPCODES

BSOUND currently has four delay opcodes: standard (opcode: "delay"), tape delay (opcode: "tape"), pingpong (opcode: "pingpong") and multitap delay (opcode: "multitap"). The main differences between these delays is their delay times and the feedback paths.

In audio processing a delay occurs when the read and write operations to a buffer do not occur simultaneously. The time difference between operations is called the delay time. During this time the signal is saved internally in the buffer. Manipulating the delay time thus manipulates the information stored in the buffer. This effect can be exploited for pitch shifting purposes.

In BSOUND, all delay opcodes additionally feature a feedback path. Feedback is the process of *feeding* the output of some operation back into it’s input. In the case of these opcodes, a proportion of the output signal (controlled by the “feedback” parameter) is added back to the input after some processing is applied to it. The bare minimum processing is volume control, but BSOUND also provides a filter for additional tonal shaping.

"delay" features a standard feedback path and a mid-long delay time. "tape" has a similar sound on the first repeat, but successive repeats become darker as the filter in the feedback path takes effect. This is a classic DSP technique to simulate the degradation vintage tape delays introduce to a signal.

Instead of sending a delayed version of channel 1 to channel 1, ”pingpong" sends the delayed version to channel 2 (and the signal from channel 2 to channel 3 etc.). This equally happens with the feedback, so an input signal bounces around from channel to channel and it is this action that gives a pingpong delay it’s name.

"multitap" is also based around one delay line per channel, but some of the signal heard at the output is the result of “tapping” the delay line. This is achieved by placing multiple “read heads” at different locations in the buffer and it can produce a more rhythmical, denser sound.

All of these opcodes have similar multitap functionality that can be activated by increasing the “spread” control.

GRANULAR SYNTHESIS OPCODES

BSOUND currently has three granular synthesis opcodes: cloud reverb (opcode: "cloud"), shimmer reverb (opcode: "shimmer") and granular transpose (opcode: “transpose").

Instead of manipulating input audio as a whole, granular synthesis turns an input audio stream into many individual grains by applying an envelope function. These grains can be manipulated individually: changing their length produces a more grainy sound and changing their playback speed creates pitch shifting effects.

We can also change the way the grains are played back, collectively: By reducing or increasing the number of grains *seeded*, we can change the overall density of the effect and by delaying the playback of individual grains randomly, the input signal can be spread out in time.

“transpose” uses relatively long grains that are played back at a different speed than the input to effect pitch shifting on the input audio.

“cloud” uses the aforementioned delayed playback and high grain rates to produce a reverb-like effect: late reverberation in large acoustic spaces physically resembles a simple delay, so delaying individual parts of the input audio can, at times, sound similar to this audio bouncing off far away walls and ceilings.

“shimmer” combines the effects of transpose and cloud: some of the audio is randomly delayed and some of it is pitch-shifted (and some is both pitch-shifted and delayed). Different implementations of this effect are very common in popular music, and were probably first heard in the music of Brian Eno.

The transpose control might warrant some further explanation: The value displayed on screen is the ratio between the input and playback speeds. Values greater than 1.000 speed up playback, values less than 1.000 slow playback down. Furthermore, the steps in the control are quantized in such a way that every ten ticks represents moving from one overtone to the next. In this way, the octave between 1.000 and 2.000 is divided into 8 equal spaces. The octave between 1.000 and 0.500 is an inversion of this spacing and represents what is sometimes called the “under-tone” series.

MODULATION & DE-MODULATION

The MOD- & DE-MOD opcodes are largely based on the eponymous module by the American synthesizer manufacturer Make Noise. The moddemod-synthesizer module consists of two circuits, each capable of amplitude modulation that can be combined for ring modulation. It and the Make Noise product line have a distinctive sound I have tried to emulate.

BSOUND currently features two MOD- & DE-MOD opcodes: amplitude modulation / distortion (opcode: "moddemod") and ring modulation (opcode: "ringmod").

"moddemod" features amplitude modulation with an internally generated carrier wave. Amplitude modulation is a mathematical process of combining two signals, commonly referred to as “carrier” and “modulator”. Crucially, in amplitude modulation the frequencies of the carrier are preserved while the frequencies of the modulator are not, so the input audio is distorted but remains recognizable.

A special case of amplitude modulation is known in the world of guitar stomp boxes as “tremolo”. It can be achieved at very low modulator frequencies.

"ringmod" produces ring modulation with an internally generated sine wave. This process preserves neither the frequencies of the input nor of the carrier wave and results in a sound that may be familiar from 1970s frequency shifters or their use in period Sci-Fi movies.

As a kicker, the internally generated sine wave, that modulates the incoming audio, can in turn be modulated by another internally generated sine wave in a process known as frequency modulation. The spectra this method produces are similar to ring modulation, but it mainly serves as a way to further distort and disfigure input audio in this case.

Both frequency modulation and ring modulation produce the most profound changes in timbre when the pitch of the modulating signal is lower than that of the carrier (i.e. in the input audio). Furthermore, the “FM index” control affects the amount of frequency modulation being applied. High index settings will garble the audio beyond recognition, at least in part because the way the frequency modulation is performed internally breaks down at these settings.

SAMPLE RATE REDUCTION

The Nyquist–Shannon sampling theorem relates the maximum frequency that can be stored in a digital signal with the sampling frequency of that signal. This maximum frequency is called the Nyquist limit and lends the control on the “crush” algorithm it’s name. Reducing the sampling rate will warp all frequencies above the Nyquist limit and produce an effect known as aliasing.

“crush” exploits the weird sounds this technique can produce. Internally, it samples the input signal at a lower sampling rate and then tries to *recover* the input. When the sample rate has to be lowered in digital audio processing, it is common to filter out frequencies above the Nyquist limit before sample rate reduction is applied to prevent aliasing. In this manner, the pre-FX low-pass filter in “crush” might be used to mellow some of the harsh artifacts “crush” can produce.

The attempt at reconstructing the input signal after down-sampling has different levels of sophistication. Fundamentally, a signal stored at 20khz has half the number of samples as that same signal stored at 40khz. At the lowest value of “interpolate”, no attempt is made to recover this lost information. As the “interpolate” parameter increases, more and more sophisticated methods of guesstimating the values in between those available are applied in a mathematical process known as interpolation.

While sample rate reduction limits the *temporal* resolution of the input signal and thus changes the input frequencies, bit depth reduction implemented in the “mu” parameter limits the *amplitude* resolution. At the maximum setting of “mu”, every sample of audio is encoded in one of 64 values. At a value of zero, the audio passes with BSOUND’s default bit depth which is 64 bit: one in 18,446,744,073,700,000,000.

To make this loss in precision and the increase in noise more bearable I have used the μ-law algorithm that is commonly used in telecommunication and should lend the audio a pleasant distortion you might recognized from phone conversations back in the day.

PHYSICAL MODELING OPCODES

BSOUND currently has two physical modeling opcodes: a large reverb (opcode: "reverb") and sympathetic string mode (opcode: “pedal").

The Karplus-Strong algorithm introduced the idea that a vibrating string could be efficiently simulated with a single delay line. The interactions with the material holding the string in place at either end could be modeled by a filter placed in the feedback path and the damping of the string could be modified by changing the feedback coefficient.

Later research further generalized this algorithm. Slightly modified delay lines can also serve as models for vibrating columns of air, tubular bells etc. These algorithms and the methods associated with them are called physical modeling.

"reverb" is based largely on the work of Sean Caroll and the implementation of his work by Istvàn Varga in CSOUND (CSOUND opcode: *reverbsc*). It simulates eight strings, each with a filter in the feedback path. What sets this reverb apart and gives in, in my ears, a unique and beautiful sound, is that the delay lines are constantly changing in length. In addition to modifying the sound, this modulation provides the reverb with more stability at high feedback settings.

"pedal" is based on an article I read a while ago and a subsequent CSOUND instrument. The input audio is “passed over” a set of 12 strings tuned to the frequencies of the lowest octave of a grand piano. This was described as way of efficiently simulating the reverberation the damper pedal in a piano produces. This method is called “sympathetic strings”, since in contrast to the original Karplus-Strong algorithm, the strings are not directly excited, but are simply *coupled* to the input. This corresponds to the strings in the piano not being struck, but simply vibrating because they are physically coupled to other strings via the bridge.

As a simulation of a grand piano, this algorithm falls short, but at the default settings the delay lines emphasize individual harmonics in the input signal in a way I find quite stirring. This makes an excellent appendix to “shimmer”.

FUTURE VERSIONS

When I first started working on BSOUND, the first opcode and algorithm I came up with was a very simple delay that you can access with the “delay”, “tape”, “pingpong” and “multitap” opcodes. I have learned quite a bit during the last couple weeks, and this has led me to re-write the delay algorithm many times: It had to be more efficient, it had to support multitap type delays, it had to have per channel feedback coefficients etc. There is one major limitation of the opcode design that I cannot overcome and this has led me to write yet another delay algorithm.

This opcode (available by typing “bbd”) does not yet have all of the intended functionality but it offers the ability to change the delay time without compromising audio quality and is available as a beta version for now.

When you change the delay time in the original “delay” algorithm this introduces noise the volume of which is roughly proportional to its speed of change. For this reason, I have limited the maximum speed with which you can change the delay time. This lagging of the delay time can itself be exploited for interesting effects and this is not the only BSOUND parameter to suffer from such a speed constraint. Changes to the modulation parameter in the physical modeling opcodes only take effect after a delay roughly proportional to its last setting, so changing from low to high settings takes more time than the reverse. The spread parameter in the granular synthesis algorithm has a similar constraint, and any opcode involving feedback is effected not only by its current settings but also by previous settings.

“bbd” does not suffer from this speed limitation. Delay time changes can be very large and sudden, and the only loss in audio quality occurs at very long delay times in a way that resembles the disintegration of audio in Bucket Brigade Delay chips, commonly used in the 70s and 80s for high fidelity delays. Since these chips responded exceedingly well to modulation, they gave rise to effects such as the “chorus” and “flanger”, both of which employ modulated delay lines. This is one avenue I intend to pursue further.

On the other hand, the advantage of modeling analogue circuits in the digital domain is that we have virtually unlimited control over such seemingly random processes like the degradation of audio in integrated circuits. It may be interesting to simulate the lossy nature of integrated circuit capacitors (that play a large role in the resultant audio

quality) or things such as clock bleed (which is also very common in BBDs).

Modeling interesting analogue circuits has certainly piqued my interest and I am looking forward to presenting some of my results to you in future versions of BSOUND.

Furthermore, here are some other ideas I have come up with that might come to an updated version soon:

Panning opcodes: BSOUND currently does not support panning in any direct way. The implementation of this is probably quite simple but I have not yet figured out how panning opcodes would fit into the BSOUND workflow.

BRANCH command: One major limitation of BSOUND at the moment is that all of the processing is applied to all of the audio equally. This limitation will be removed with a BRANCH command that gets its name from a CPU instruction found on almost any CPU since at least the 1960s.

MACRO control: It can be cumbersome to edit all of the available parameters one by one, so I am planning on introducing MACRO controls that allow you to edit all parameters that share certain features (or all parameters of your choosing) at once.

PERFORMANCE LOAD: If you ever plan on using BSOUND in front of an audience, it would probably be quite convenient to be able to concatenate different “scenes”. This might be as simple as adding a “load” command, or I might give you the ability to crossfade between different BSOUND states (this is of course where the FORK command will come in, as well).

If you have gotten this far in the manual, I hope that means you are just as excited as I am about this little application I have written. I hope it will bring you at least as much joy as it has brought me, and I hope you will come up with some interesting uses for its features and quirks. Also, if BSOUND suddenly crashes or something else unexpected happens, please let me know, so that I can patch it for future versions.

Happy new ears!

KNOWN BUGS

What would software be without bugs?! “Touch Bar can’t be negative” (this is an actual error I got a couple days ago), “404 Page requested not found”, “Segmentation fault” etc. are the best friends of the programmer.

This section serves as a TODO-list for me and a reminder for you that this is work in progress. I would kindly request that you inform me when something goes wrong. Useful information might include: what went wrong (i. e. what did you expect to happen vs. what actually happened) and steps necessary to reproduce the bug.