pp.tools manual

Contents

1 Introduction

pp.tools is a libary of Pure Date abstractions with a focus on electroacoustic composition, live electronics and sound design. It was developed by Philipp Schmalfuß under the supervision of Teresa Carrasco and Prof. Robin Minard at the Studio for Electroacoustic Musik (SEAM) in Weimar.

1.1 Installation

To run this software you'll need to install Pure Data version 0.48.1 or later. It won't work with alternative Pure data distributions, like Pd-extended or Pd-L2ork/Purr-Data.[1]
Pd can be downloaded from https://puredata.info/ or from Miller Puckettes homepage http://msp.ucsd.edu/.
Once you installed Pure data, you'll have to tell it where to find the pp.tools-libary. Go to Edit→Preferences→Path...→*New...* and set the path to the pp.tools directory. Alternatively, you can use the *declare* object to set a path relative to the directory of your patch like [declare -path ../pp.tools].

1.2 Pd-abstractions

There are 3 classes of objects in pure data: core objects or "vanilla" objects such as [osc~], these are the vital building blocks of every Pd-patch, "externals" that are written in c-code and compiled for Pd by users in the Pd-community and "abstractions".

Abstractions are essentially Pd-patches that can be called in a "parrent" patch like any other object. They can have creation arguments and methods for changing internal parameters via control-messages or graphical user interfaces using the "Graph on Parrent" function. One advantage of abstractions is that they can be opend and changed from within the parrent patch, but they lack in computational efficiency compared to externals. Many Pd-users are maintaining and sharing their personal set of abstractions online, some of which are quite inspirational learning resources.[2]

1.3 Anatomie of a pp.tool

Once you have downloaded and installed the pp.tools, you can simply create a Pd-object and type *pp.sfplayer~*, this will create a new module from the pp.tools libary, in this special case the soundfile player.
Right-click→Help will open a helpfile with some basic information about the object,

[1] Pd-extended is unmaintained since 2013, its successor Purr-Data currently misses some features that are needed to run pp.tools.

[2] Just to mention a few;

The *USSS Tool Kit* by Adrian Moore is the only abstraction libary I am aware of, that lies its focus primarily on electroacoustic composition; https://www.sheffield.ac.uk/usss/ussstools

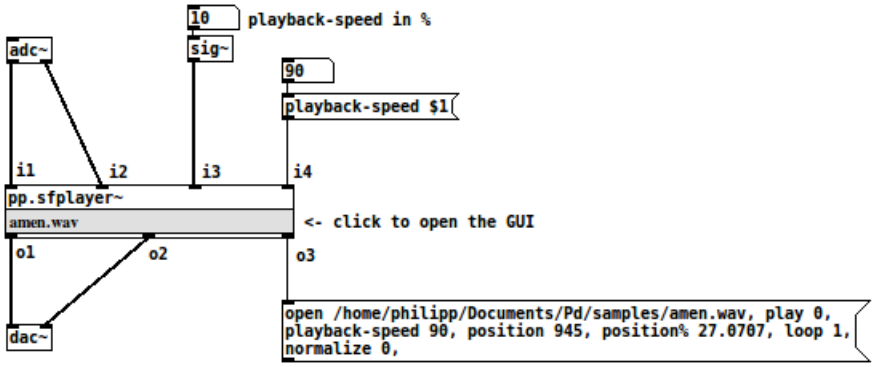
Automatonism, a "stand-alone" modular software-synthesizer developed by Johan Eriksson; https://www.automatonism.com/

.mmb, a collection of sophisticated general abstractions by Mike Mooser-Booth; https://github.com/dotmmb/mmb

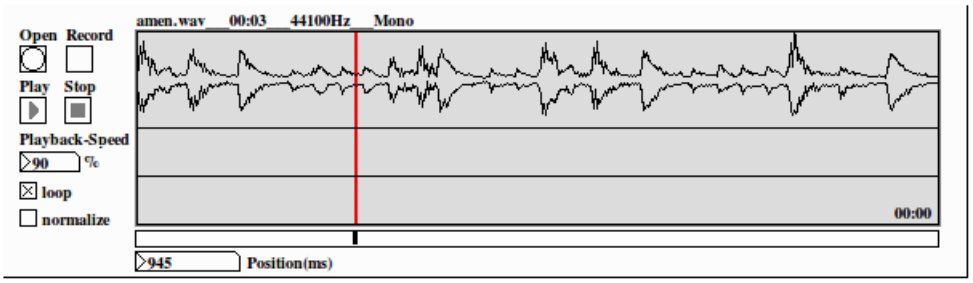
right-click→Open to look inside the abstraction. If you made changes inside the abstraction, Pd will ask you to save it once you close the parrent patch. If you have little experience with Pure Data, **do not save a pp.tools abstraction!**. Some of them will loose their functionality if they are saved.[3] In case you accidentally save a pp.tool, you might have to reinstall it.
You'll notice that grey area inside the object, a click on it will open a Graphical User Interface in a new window which gives you quick access to all the object's parameters(Fig.2.2). If there is only one parameter to change, this area will hold a slider and a numberbox.
Each module comes with multiple inlets and outlets; the leftmost inlets/outlets are usually reserved for audio-signals, the rightmost inlet is reserved for control-messages.

The rightmost outlet will set a message box with all the parameters that have been changed. This way it is possible to save changes with the parrent patch or to send parameters to other instances of the abstraction.(Fig.2.3)
There are extra contol signal-inlets in most modules. Control signals are added to the value that was set via message or in the GUI. (Fig.2.1)

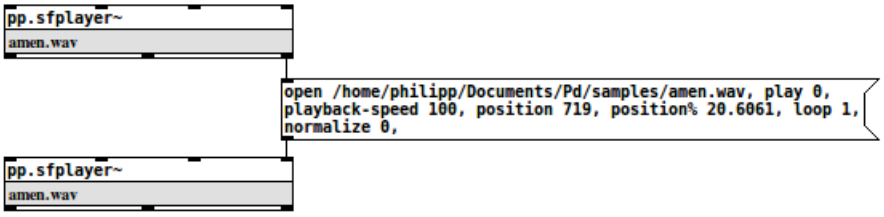
[3]This is due to the use of dynamic patching, a method to generate patches with message commands.



(Figure 2.1)
A simple Pd-patch with [pp.sfplayer~]. The signal at inlet i3 is added to the value set at inlet i4, so playback-speed sums up to 100%. Outlet o3 will set a message with all the parameters.
Inlets i1, i2 and outlets o1, o2 are stereo audio-signals.



(Figure 2.2)
The GUI of pp.sfplayer~



(Figure 2.3)
Sending parameter changes to another instance of the abstraction. There must be a massage box in between the objects, directly connecting two objects won't work.

1.4 Efficiency

Rendering graphical elements(bangs, toggles, sliders, arrays and so on) in Pure Data is a very cpu-costly process. That is one reason i choosed to hide these elements in a subwindow. If you have a lot of these windows open at the same time, they can slow down Pd significantly. Also, some processes for visual feedback will only start once you open a GUI-subwindow for the first time. To improve efficiency, I'd suggest to not open the GUI windows in the first place and use message commands instead.

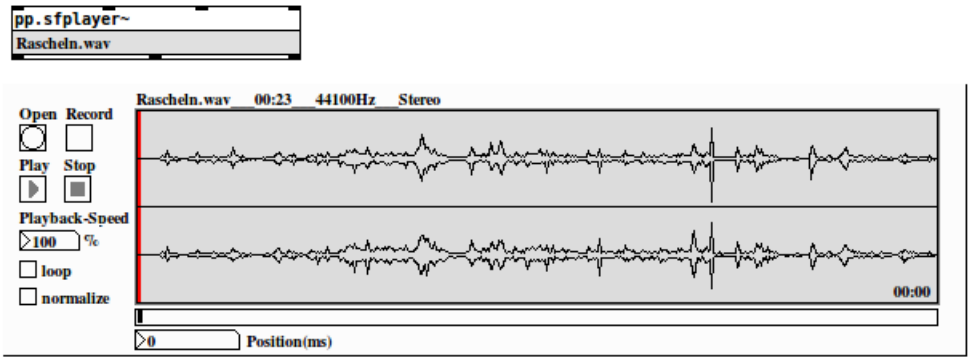
1.5 Latency

The pp.tools libary includes fft-based modules. They will delay the input signal n samples, where n is the size of the fft-block (1024 by default[4]). To compensate this latency, you can use [pp.fft-block~] or [pp.sdel~].

2 The tools explained

2.1 Soundfile processors

2.1.1 Soundfile playback



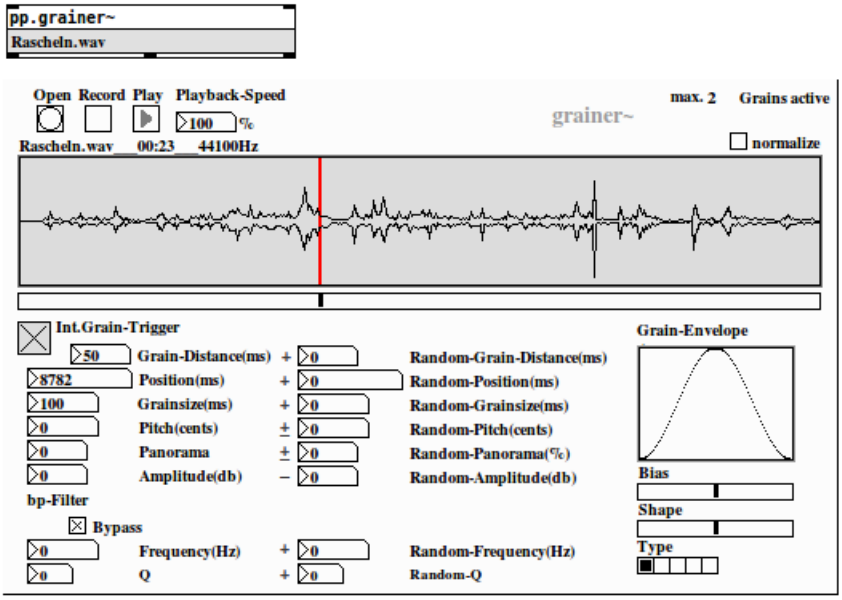
(Figure 3.1, pp.sfplayer~)

[pp.sfplayer~] allows playback of a soundfile at variable speed. Playback speed is in percent of the original playpack speed. If the playback speed is negative, the soundfile is played backwards. Just like in an analog sample device, the pitch is bound to the playback speed, so that if the speed is doubled, the pitch will jump up one octave. All other parameters are pretty much self explanatory: *Position* defines the starting position in the soundfile. *Play/Pause* will start playback or pause playback at the current position. *Stop* stops playback and jumps back to the starting position. The *Record* button will record audio signals from the first two inlets into the buffer of pp.sfplayer~.

[4] $1024/(\frac{SR}{1000}) = 23.22$ milliseconds at samplerate SR 44100.

The maximum lenght of a 44.1kHz soundfile that can be processed with this module is about 16,15 minutes. Indexing large buffers in Pd is problematic, the usual method is to multiply a sawtooth oscillator ranging from 0 to 1 by the total number of samples in the buffer. However, with buffers larger than about 32000 samples, this will cause distortion as the index increases.[5] Luckily, Miller Puckette provided a workaround with the onset inlet of [tabread4~] to avoid this problem and an example[6] that shows how to use this onset to allow clean variable-speed playback of long audio buffers. pp.sfplayer~ is based on this example, which makes it a fairly complex patch for a rather simple task.

2.1.2 Granulation



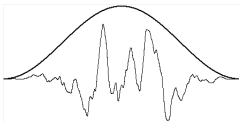
(Figure 3.2, pp.grainer~)

[pp.grainer~] is a granular sampler for mono soundfiles (if you load a stereo soundfile into pp.grainer~, only the left channel will be processed.) The technique of sound-granulation was explored by various composers throughout the history of electroacustic music. As described by Curtis Roads in his book *Microsound*, a grain is a very short snippet of sound (a waveform) shaped by an amplitude envelope(Fig.3.3). The duration of a grain typically ranges between 1 and 100 milliseconds.[7]

[5] For details, look at B15.tabread4~-onset.pd in Pd's audio examples Help→Pure Data/3.audio.examples/B15.tabread4~-onset.pd

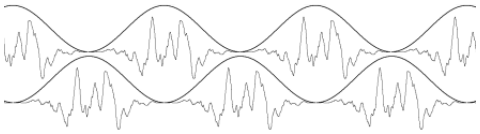
[6] in Pd Help→Pure Data/3.audio.examples/B16.long-varispeed.pd

[7] cf. Roads, 2001, pp. 86-87



(Figure 3.3, a single grain)

If many of these grains are combined over time, they are percieved as a pitched tone, a textured sound or as a swarm of little sound objects, depending on how much they vary in pitch, duration and amplitude.
In pp.grainer~ single grains are triggered by a bang sent into the rightmost inlet, or optionally by an internal metro. If the grain duration is longer then the time between two trigger events, the grains will start to overlap(Fig.3.4). The maximum number of overlapping grain streams is 128, hence to prevent clicks the duartion of the grains should not be longer than the trigger rate (in milliseconds) * 128.

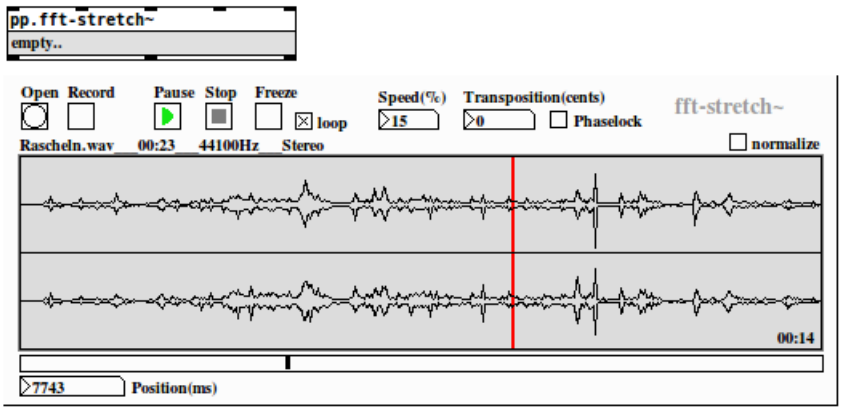


(Figure 3.4)
Two overlapping grain streams. In this example the grain duration is twice the trigger rate.

The source of the waveform in each grain is the soundfile that was loaded into the buffer, starting from the position that was set via the *Position* parameter. If you hit the *Play* button the position will change automatically. *Playback-speed* defines how fast the position will change. This way you can use granulation to change pitch and playback-speed of a soundfile independently. With a slow playback speed and two or more overlapping grain streams the soundfile gets stretched. *Pitch* changes the pitch of the grains relative to the original pitch of the soundfile. A change in pitch also implies that the grains contain a smaller or larger portion of the source material. *Panorama* changes the spatial position of the grains in the stereo field. Optionally, the grains are filtered by a resonating bandpass-filter.

Each parameter can be randomized. Random values are added to the parameter values of *Grain-Distance*(internal trigger rate), *Position*, *Grainsize*, Filter-frequency and Filter-Q.
Parameters of *Pitch* and *Panorama* are center values, i.e. random values are added/subtracted.

2.1.3 Time stretching

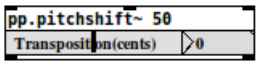


(Figure 3.5, pp.fft-stretch~)

[pp.fft-stretch~] uses Miller Puckettes Pd-implementation of the phase vocoder to recreate magnitude and phase precession of two overlapping read windows.[8] This way, it allows to change pitch and playback speed of a soundfile seperatly. Pitch and playback speed can also be modulated by a contol signal. The *Phaselock* option forces phase coherency, which is useful if you want to transpose sharp transient sounds but can sound artificial with slow playback speed.

2.2 Effects

2.2.1 Pitch shifting

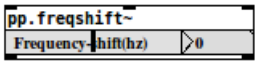


(Figure 3.6, pp.pitchshift~)

[pp.pitchshift~] uses two overlapping variable delay lines to transpose an incoming signal. This techinque is described by Miller Puckette in chapter 7 of his book.[9] As opposed to a frequency-shifted sound, the frequencies of pitch-shifted sound retain their harmonic relationships.
Transposition is in cents(one hundredth of a semintone). The variable delays are windowed by a raised cosine, somewhat like the two overlapping grain streams described above(Figure 3.4). The size of the windows is set in the argument of the module or via control message. Overlapping windows will cause phase cancellations and thus become audible if the sound is transposed. You'll have to figure out which window-size works best for your sound by experiment.

[8] cf. Puckette 2007, pp. 292-294
[9] cf. Puckette, 2007, pp. 202-208

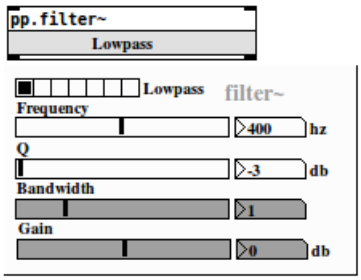
2.2.2 Frequency shifting



(Figure 3.7, pp.freqshift~)

[pp.freqshift~] shifts the frequency spectrum of an incoming signal by a set amount. This way the frequencies will not retain their harmonic relationship. The module is based on Puckettes single sideband modulation example[10], but uses Olli Niemitalo's quadrature transform coefficients as implemented by Katja Vetter instead of the hilbert transformer.[11]

2.2.3 Filtering

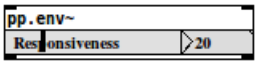


(Figure 3.8, pp.filter~)

[pp.filter~] and [pp.vcfilter~] are convinient filter modules. They can act as lowpass, highpass, bandpass, resonating bandpass, notch or allpass filters. The filter type is set in the first argument and can be changed via message command or in the GUI. Filter coefficients are calculated following Robert Bristow-Johnsons Audio-EQ-cookbook rezipes.[12]
pp.vcfilter~ is "voltage controlled", i.e. it offers control signal-inlets for filter-frequency and q/bandwith. Although written in portuguese, the filter section of Alexandre Torres Porres live-electronics tutorial was a great learning resource for me.[13] The implementation of the signal controlled biquad filter that is used in pp.vcfilter~ is roughly based on one of this tutorials examples.[14]

[10] cf. Puckette, 2007, p. 259
[11] http://www.katjaas.nl/hilbert/hilbert.html
[12] http://www.musicdsp.org/files/Audio-EQ-Cookbook.txt
[13] downloadable from github: https://github.com/porres/LiveElectronicsTutorial
[14] [biquad~].pd in LiveElectronicsTutorial/Parte 6 Filtros e Reverb/30.Filtros-II/3.Plano-Z

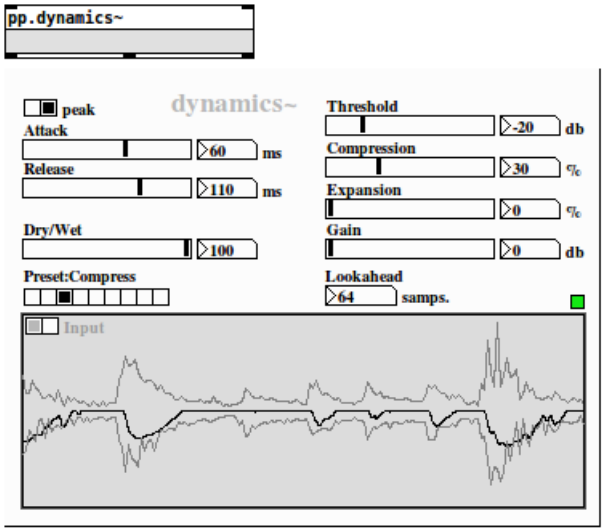
2.2.4 Envelope following



(Figure 3.9, pp.env~)

[pp.env~] is a simple envelope follower. It uses the RMS (root mean square) method for amplitude detection. *Responsiveness* refers to an averaging lowpass filter that smoothes the envelope curve.[15]

2.2.5 Dynamics processing

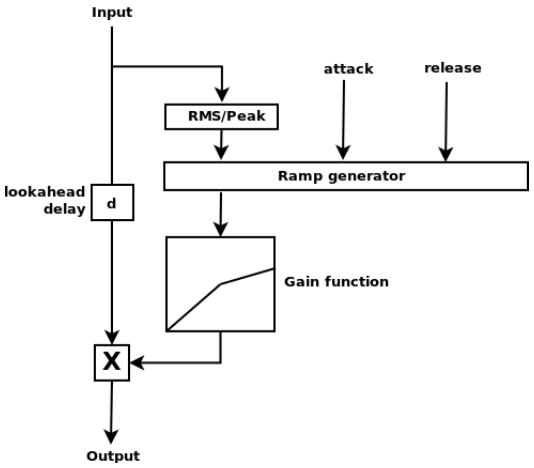


(Figure 3.10, pp.dynamics~)

[pp.dynamics~] is a combined compressor/expander. It affects the dynamic range, i.e. the level difference between loud and soft passages of an audio signal. By default this module is a limiter but it can serve multiple purposes like compressing, gating, distorting or softening transients. *Compression* reduces the amplitude of a signal that exeeds the *Threshold* level by a set amount in percent. *Expansion* reduces amplitudes that fall below the threshhold.
Attack defines the time it takes to complete the gain reduction, *Release* sets the time it takes to gradually reduce the amount of compression/expansion.
Attack and release times refer to a ramp-generator (fexpr~) inside the abstraction, the actual attack and release times vary relative to the amplitude of the signal and the threshold level.

[15] cf. Puckette, 2007, pp. 252-254

There are two methods to define the amplitude of a signal; mean amplitude or *RMS* and *Peak*, the peak method uses Olli Niemitalo's quadrature transformer that I mentioned above.[16] Both methods are inflicting some distortion if the attack and release times are very short.
Lookahead actually delays the signal by some samples(64 by default) and averages the amplitude envelope to reduce distortion.



(Figure 3.11, Workings of a dynamics processor)

[16]Katja Vetter is using the same method in her Quadrature Compander:
http://www.katjaas.nl/compander/compander.html

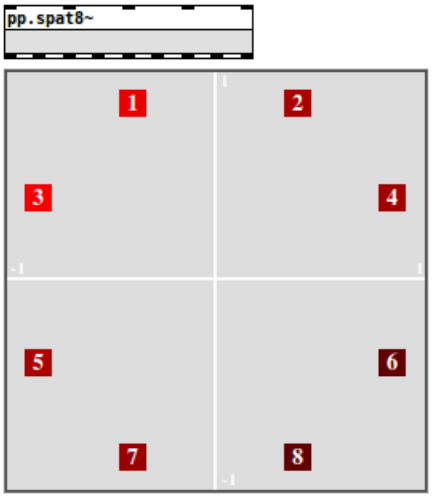
2.3 Spatialisation

2.3.1 Panning

(Figure 3.12, pp.pan~)

[pp.pan~] places a mono signal in the stereo image following the sine-cosine panning law. Position in the stereo field is set with *Panorama* or via a control signal, ranging from -1(left) to 1(right).

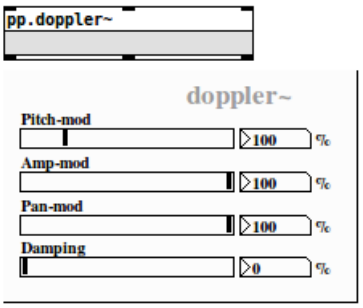
2.3.2 Multichannel spatialisation



(Figure 3.13, pp.spat8~)

[pp.spat8~] is a spatialisation tool for arbitrary 8 channel speaker setups. The amplitude weighting of each loudspeaker is based on the distance from a virtual sound source in a cartesian coordinates system. The speakers can be rearanged in the GUI or via message commands. In the GUI, each speaker is represented by a canvas, so to move them around you will have to switch to editmode. The position of the sound source is defined by x and y signals, ranging from -1 to 1. A third signal defines the spatial spread of the virtual sound source. With a message command *preset* you can switch between two common 8-channel speaker setups.

2.3.3 Doppler

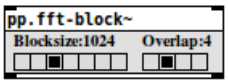


(Figure 3.14, pp.doppler~)

[pp.doppler~] emulates the changes in pitch, amplitude and high frequency attenuation that occour when a sound source is moving relative to the standpoint of an observer. A signal to the second inlet defines the trajectory of the moving source, e.g. a 20 seconds signal ramp from distance -100(far left) to 100(far right). The percieved moving distance results from the interaction of the control signal amplitude and the adjusted parameters *Pitch-mod*, *Amp-mod*, *Pan-mod* and *Damping*. A second outlet sends out a control signal for spatialisation with [pp.pan~] or [pp.spat8~].

2.4 Spectral Processing

2.4.1 FFT-Blocks



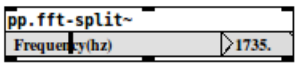
(Figure 3.15, pp.fft-block~)

Modules that run processes in the spectral domain analyse signals in overlapping dsp blocks that are usually larger than Pd's default block size (64 samples). These block sizes must be powers of 2, e.g. 512, 1024, 2048 samples. Different block sizes affect the fft-processed sound in various ways; in general, larger fft-blocks result in a higher frequency-bin resolution but they can also "smear" the sound, wich is most noticable in sharp transients. You can reduce this effect with more overlapping.

[pp.fft-block~] serves two purposes; the right outlet sets a message with blocksize and overlap factor that can be used to set block sizes and overlap in other fft-based modules, a signal to the leftmost signal-inlet gets delayed one block.[17]

[17] cf. section 2.6 Latency

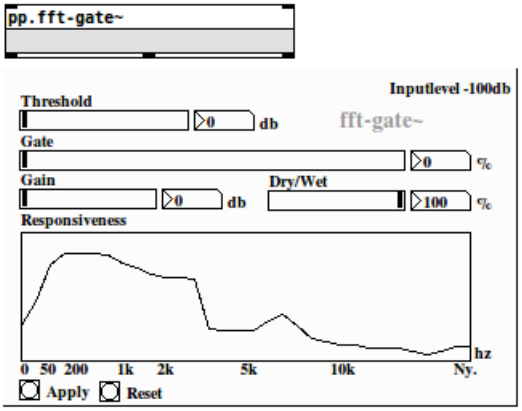
2.4.2 Frequency Splitting



(Figure 3.16, pp.fft-split~)

[pp.fft-split~] splits the frequency band of a sound in two parts (leftmost outlet and middle outlet) at a set frequency.

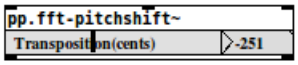
2.4.3 Spectral Gate



(Figure 3.17, pp.fft-gate~)

[pp.fft-gate~] is a spectral denoiser module. The *Gate* parameter reduces the amplitudes of frequencies that fall below the threshold level by a set amount in percent. In the GUI, you can draw a responsivness curve that acts roughly like a noise profile, e.g. you could reduce high frequency noise, but leave the lower frequencies untouched. The denoised part of the sound comes out of the leftmost signal-outlet, the second signal outlet spits out the "missing" parts, i.e the lower amplitude frequencies that were reduced. This way, you can process lower amplitude frequencies(noise) and higher amplitude frequencies(formants) seperatly.

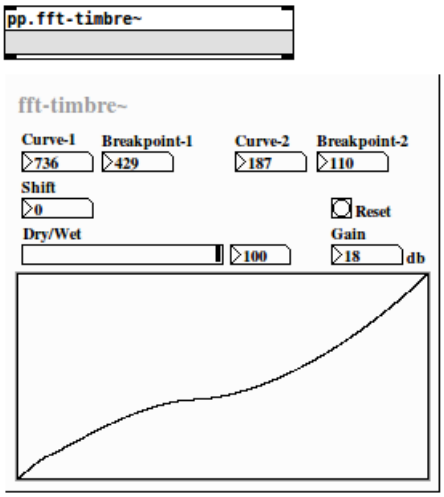
2.4.4 FFT-pitchshifting



(Figure 3.18, pp.fft-pitchshift~)

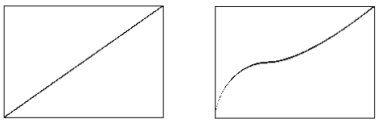
In this module two overlapping variable delay lines and the phase vocoder that I mentionned above[18] are used to transpose a sound source. Compared with the [pp.pitchshift~] there are considerable less artifacts if the sound is transposed up or down one octave.

2.4.5 Timbre



(Figure 3.19, pp.fft-timbre~)

This peculiar module uses a transfer function to mangle the frequencies of a sound. A straight line from bottom left to top right implies input frequency equals output frequency. As the gradient changes, some frequency-bins drop out while others shift position.

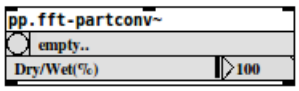


(Figure 3.20)
With a straight line input equals output, e.g. 0 1 2 3 4 5 6 7 8 9.
A curved function like that on the right changes the output to 0 3 5 5 5 6 6 7 8 9.

[18] cf. 3.1.3 Time stretching

This type of bin-reordering works best for sounds with rich harmonic content and sharp transients, like plucked string instruments. However, similar to a frequency shifter it will mess up the harmonic relationship of the frequencies. To generate a transfer function, you can use the *Curve*, *Breakpoint* and *Shift* parameters or use the GUI to simply draw it by hand.

2.4.6 IR-convolution

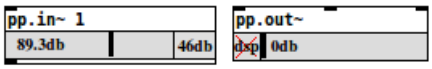


(Figure 3.21, pp.fft-partconv~)

[pp.fft-partconv~] allows low latency impulse response convolution. It uses a non-uniformly partitioning scheme for impulse responses of arbitrary lengths proposed by William G. Gardner.[19] The maximum length of an impulse response is about 11 seconds, wich should be more than enough for most reverb impulses. Latency is defined by the size of the first partition in samples set in the argument of the module. Technically very low latencies are possible(the minimum fft-block size is 4 samples in Pd), but small sizes of the first partition will also result in significantly higher cpu-load.

2.5 Miscellaneous

2.5.1 Input and Output



(Figure 3.22, pp.in~ and pp.out~)

[pp.in~] and [pp.out~] are basic objects for microphone input and stereo speaker output.

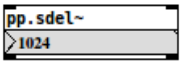
2.5.2 Signal multiplier

(Figure 3.23, pp.mult~)

[pp.mult~] is a signal multiplier for the lazy patchers. The number of inlets/outlets is defined by the creation argument.

[19] cf. Gardner, 1995

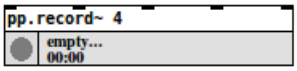
2.5.3 Sample Delay



(Figure 3.24, pp.sdel~)

[pp.sdel~] delays a signal by n samples.

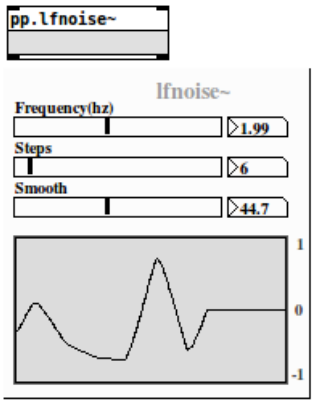
2.5.4 Recording



(Figure 3.25, pp.record~)

[pp.record~] writes audio signals to a .wav soundfile. The number of channels is defined by the creation argument. Bitsizes 16, 24 or 32 can be set with a message command. A click on the object opens a save file-browser to choose the directory and the name of the soundfile to be saved.

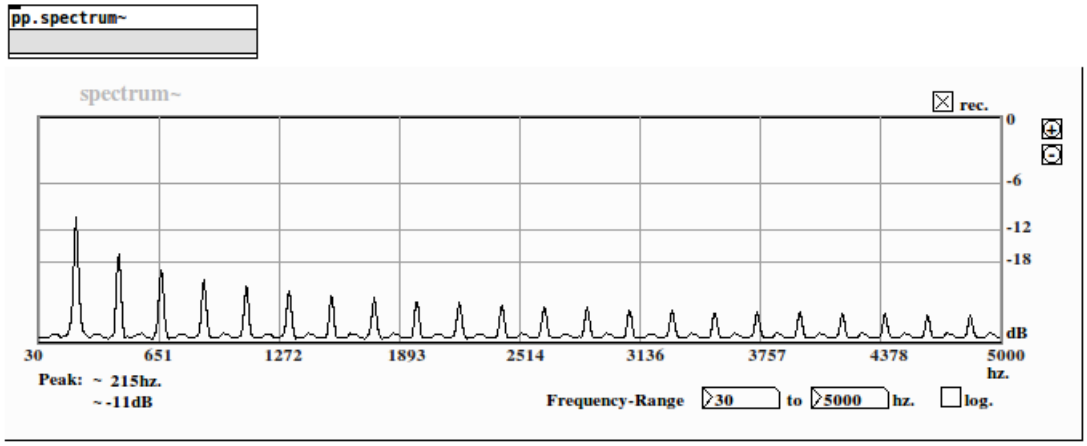
2.5.5 lfnoise



(Figure 3.26, pp.lfnoise~)

[pp.lfnoise~] is a random signal generator. It can be used to modulate other signals with an arbitrary low frequency signal or to generate various types of noise.

2.5.6 Spectral Analysis



(Figure 3.27, pp.spectrum~)

[pp.spectrum~] plots the spectrum of a sound source. It is possible to set the amplitude and frequency range of the the spectrum-plot and to switch between linear and logharithmic scaling.

3 References

Miller Puckette, *The Theory and Technique of Electronic Music*, World Scientific Co. Pte. Ltd., 2007.

Curtis Roads, *Microsound*, The MIT Press, 2001.

W. G. Gardner, *Efficient convolution without input-output delay*, Journal of the Audio Engineering Society, Vol. 43, pp. 127–136, 1995.