Modules

Utilities

- Change Gain
- Channel Manager
- Concat
- Slice
- Make Loop
- Statistics
- Unary Operator
- Binary Operator
- Amplitude Shaper
- Resample

Spectral Domain

- FIR Designer
- Convolution
- Complex Convolution
- Fourier Translation
- Wavelet Translation
- Hilbert Filter
- Mosaic
- Sonagram Export
- Bleach
- Spectral Patcher

Multiband

- Band Splitting
- Chebychev Waveshaping
- Exciter
- Voocooder

Time Domain

- Step Back
- Kriechstrom
- Serial Killa
- Seek + Enjoy
- Lückenbüßer
- Pearson Plotter
- Sediment
- Dr Murke

Distortion

- Frequency Modulation
- Laguerre Warping
- Ichneumon
- Needlehole Cherry Blossom
- Rotation
- Seek+Destroy
- BlunderFinger

Misellaneous

- Sachensucher
- Recycle Or Die
- Convert files
- CD Verification
- SMPTE Synthesizer
- Beltrami Decomposition

Restauration

- Comb Filter
- Declick
- Schizophrenia

Special

- Batch Processor

Welcome to FScape!

FScape is a standalone, cross-platform audio rendering software.

To contact the author, send an email to `contact at sciss.de`. For project status, API and current version visit github.com/Sciss/FScape.

(For complete information on installing/compiling please see the README file included with FScape)

The following documentation is taken directly from the accompanying HELP Files. I have organized the information here because having a physical manual has aided me time and time again while using FScape.

This help is also accessible from the help menu within the application. For each of the processing modules, help is available via Help.

(Now for proper introductions)

Hello!

Welcome to the FScape user manual and guide. I can't explain what it is about FScape – maybe it's that it can run on a tiny Windows XP machine with an Atom cpu? Maybe it's because FScape and its companion apps can all fit on a small flash-drive and no matter where you are you can grab a computer and start processing and manipulating sounds.

Moreover it is the freshness that comes from a completely abstract mode of thinking about sound and working with sounds. Abstract if you, like me, come from the world of DAWs and Plugins. Big software and hardware companies with pricey packages (albeit exceptionally cutting edge and fun to use) and constant upgrades all that require the fastest computer with up-to-date everything. Ugh!

FScape opens you to a world or possibilities. It gets you thinking not about kick drums and club drops but Macro Organization, Timbre Evolution. It causes you to work partially in the dark – no instant gratification of hearing your parameter tweaks. It helps you to embrace the truest sense of nonlinear both in terms of processing whereas processing an input 7 or 8 times actually leaves you at the *beginning* of a new sound adventure. It also epitomizes nonlinear in that my work flow can consist of multiple open windows, programs and folders with files getting dragged every which way and place! Finally it causes you to become fastidious with your file organization because otherwise you will be left with folders of hundreds of samples with strange derivative names (TrvpB'1Shp2VcdBss17Ct5C1GnFM 4 – yes that is an actual sample name)

tweak/process/evaluate/repeat

Utilities

Change Gain

This module simply changes the gain (volume) of a sound file. It can be used to find the peak sample and the RMS energy of a file

Waveform Input: Sound file to be volume adjusted.

Info Button: Click to find the peak and RMS of the input file. The text box is frequently updated while scanning the file. Preliminary results are placed in brackets. A letter or number attached to the peak location indicates the channel in which the peak occurs. Once the info has been generated normalization is faster because FScape remembers the peak. Therefore if you change the file contents after the info has been output you have to re-enter the input name or re-press the info button to notify this module.

A special hidden utility feature is marker printout to the console, invoked by holding down the Meta key while clicking on the info button.

Waveform Output: Destination sound file and gain control. If the gain type is set to "**absolute change**" then the input is directly tuned according to the dB amount. In the "**relative unity**" mode the input is first normalized and then adjusted by the dB amount.

last modified: 28-Aug-05

Channel Manager

Utility for splitting multi-channel sound files into distinct mono files and vice versa

Add File: Loads sound files to be processed. Each new file will be added to the list gadget beneath. Use the Remove/ RemoveAll/ MoveUp/ MoveDown buttons re-order or remove files.

Operation Mode: In "**Merge channels**" mode, all channels from the input files are collected in the order of the list and put together info one target soundfile. Input files needn't be mono. E.g., if you have two stereo input files "schoko1.aif" and "schoko2.aif", you'll get a four channel output file with channels 1 and 2 corresponding to "schoko1.aif" and channels three and four representing "schoko2.aif". You can mix sampling rates and bit rates and even file lengths, however the target file will have the sampling rate of the first input file (no resampling algorithm is applied) and the length will be the minimum (?) of all input file's lengths.

In "**Split channels**" mode, a single (multichannel) input file is split into multiple mono files.

Output File: Destination sound file and gain control. Note that the default suffix is shown as a hyphen, so if you drop "schoko.aif" as input file, the output file will read "schoko-.aif". In **merging mode**, the output file will become "schoko-mono.aif", in **splitting mode** the output files will be renamed "schoko-1.aif", "schoko-2.aif" or "schoko-l.aif", "schoko-r.aif" depending on the auto-naming-scheme. *Beware : if you remove the hyphen from the output file's name you might overwrite your input files*.

Suffix List: A list of suffixes used for the output files. You can rearrange them with the **MoveUp/ MoveDown** buttons. Switch the **Auto-scheme** gadget to choose between a naming scheme. Whenever you select a suffix from the list, it will be appended to the output file name. You can now edit the output file name and confirm changes by hitting return. The chosen suffix is now known as "**manual**" and won't be affected by changes of the Auto-scheme. To go back to auto mode, select the suffix from the list and hit "**Auto**".

last modified: 29-Jul-04

Concat

Glue a series of sounds together

Waveform I/O: The sounds to be concatenated should contain integer numbers in their names. So if you drop "schoko001.aif" as the first sound and "schoko111.aif" as the last sound, the current input gadget will tell you that the program detected 111 sound files. I'm not sure if it works, but *I think you can also exchange first and last sound file and then they'll be concatenated in the reverse order*.

Scissor Settings: Each input sound file is known as a "**chunk**". You can cut off some of each sounds beginning by increasing the chunk offset, or truncate the chunk's length (cutting off bits at the end). A chunk overlap of zero means a new chunk starts exactly one sample frame after the last sample of the preceding chunk, whereas increasing the chunk overlap allows you to blend the sounds together, applying a crossfade. Note that the chunk overlap is not identical to the crossfade lengths, which is specified by "**Overlap Crossfade**". So, if you have an overlap of one second and no crossfade, then chunk 2 simply starts one second before chunk 1 is finished, but won't be faded in or the like. If you set Overlap-Crossfade to 100%, then the whole overlapping region is blended. If the chunks are phase-correlated choose the Equal-Energy crossfade type, in most other cases you'll want to choose Equal-Power crossfades which usually give the smoothest transition without an audible dip in the loudness.

Known Bugs: I vaguely remember there's a bug about parsing some sound file names, I think it appears if you convert audio CDs to sound files using iTunes which get labeled "Title 1.aif" etc. You might have to rename the files discarding all the white spaces.

last modified: 29-Jul-04

BASSO'S NOTES: Concat (like many of the utilities) may seem basic on the outside but can be used for more creative techniques.

Try grabbing a single cycle using Eisenkraut and save it. Then copy the file a number of times (around 20 to start.) You will have to rename the files to the same name with their numerical iterations (cycle1.wav cycle2.wav etc) this can be tedious so use a simple batch file naming program. Now open Concat and point it to the single cycle folder and drop in the first and last sound. From here your options open wide.

To start - process the files as is so you get a single sound whose pitch will be in relation to the number of samples in the original cycle (see Appendix E). Now you have a reference sound. Go back to Concat module and adjust some of the parameters! You could trim the files with Chunk Offset so that's worth trying but keep in mind a) you could also do that ahead of time in Eisenkraut (for this example) b) it will alter the perceived pitch. Try adjusting the Overlap and Cross-Fade and compare the results with your original. With some adjustments you can make all sorts of "single shot" synth sounds with internal motion.

Slice

Slice a sound into a series of equal-length chunks

Waveform I/O: The input file is a sound file to be sliced. The output file is a prototype for each slice. For example, if output is "MySoundCut.aif" and "**Separate files**" is checked, assuming the number of slices to be five, you'll get five files named "MySoundCut1.aif" to "MySoundCut5.aif". These files can in turn be glued together again using the Concat module. If "**Separate files**" is **not** checked, there will be only one output file "MySoundCut.aif" which contains all chunks right after another.

Slice Settings: the algorithm initially advances in the input file by the length given in "**Skip length**" (unless zero). It then writes an audio chunk with length designated by "**Slice length**" (first slice). It then advances by the length designated by "**Skip length**" (unless zero). It then writes the second slice and repeats. Note that "**Skip length**" may be negative which means that the slices will overlap. Also note that if "**Skip length**" is negative and its magnitude is greater than the "**Slice length**" you can walk backwards in the sound file. A value of "Skip length = -Slice Length" is forbidden.

In the **normal mode**, as many chunks (slices) are written as possible until reaching the ending (or beginning in "backward mode") of the input file. "**Automatic rescale**" enables an alternative mode which rescales "**Slice Length**" and "**Skip Length**" by a factor so as to result in a predefined number of slices, given by "**# of Slices**". In this alternative mode, you may optionally define a "Final Skip" which is the length between the ending of the last chunk (slice) and the ending of the sound file.

"**Normal mode**" example: let your input file be one minute long. Assume "**Slice Length**" to be four and "**Skip Length**" to be three seconds. Assume "**Initial skip**" to be two seconds. Output: Slice 1 is taken from input file 00:02 to 00:06 ; slice 2 is taken from input file 00:09 to 00:13 and so on; slice 8 is taken from input file 00:51 to 00:55, slice 9 is taken from input file 00:58 to 01:02 (with the last two seconds being silent). slice 9 is the last slice.

"**Automatic rescale**" example: let your input file be one minute long. Assume "**Slice Length**" to be four and "**Skip Length**" to be three seconds. Assume "**Initial skip**" to be two and "**Final skip**" to be three seconds. The net length of the input file is thus 60 seconds minus two minus three seconds = 55 seconds. Now assume "**# of Slices**" is set to five (you want five slices). "**Slice Length**" and "**Skip Length**" are rescaled by a factor of f = 55/(5*4+(5-1)*3) = 1.71875, so each slice has length 6.875 seconds, each skipping amounts to 5.15625 seconds. (the ratio between slice length and skip length remains 4:3).

last modified: 01-Jul-06

BASSO'S NOTES: Slice can be a good friend when you come to the end of a Macro Organizational treatment. You've taken a long field recording and reorganized it but now you want smaller chunks to work with on a Micro level. Enter Slice.

Try using times in BPM equivalents to "force" slices into potential loops. Use Appendix A included at end.

If you are entering manual Slice settings make sure # of slices is grayed out – sometimes you have to click "Rescale" twice to turn this off.

Make Loop

Apply a crossfade to a sound so that it can be seamlessly played back as a loop

This works by creating a crossfaded overlap between the beginning and ending of the sound. Either a piece from the beginning is cut away (= crossfade position "end of loop \ pre loop /") or the ending is cut away (= crossfade position "begin of loop \ post loop /"). In the former the maximum fade length is determined by the initial skip setting, in the latter the maximum fade length is determined by the final skip setting. The actual crossfade length is the minimum of this implicit maximum length and the "**crossfade length**" setting.

For example, with crossfade position "**end of loop \ pre loop /**", choosing an initial skip of 2 seconds and a crossfade length of 1 second results in a sound that is 2 seconds shorter than the input sound, starting directly at offset 2 seconds, running right to the end and having a final crossfade of one second at the very end that makes the loop work.

For strictly phase correlated signals, an "**equal energy**" fade type is provided, however usually signals are not correlated and you use "**equal power**" to approximately maintain original loudness during the crossfade.

*last modified:*03-oct-06

BASSO'S NOTES: This module is fairly basic. A better process to "Make a Loop" (Looperize?) – especially after a series of Slice operations – would be to open in Eisenkraut and Fade Out/In manually. While being tedious it would yield more controlled results.

Statistics

This module allows you to calculate a sound's overall amplitude (peak) or power (amplitude-squared RMS/Average) spectrum and some other datasets like *differentiated phase spectrum* and *histogram of samples* (elongation).

Record size specifies the size of the data set. E.g., when you calculate the amplitude spectrum, the module will take FFT's of this size throughout the sound file and then average them. (See Appendix C for information on FFT.)

Note that due to a bug, the axis settings (logarithmic frequency and/or amplitude scaling) have to be set *before* running the analysis. Also note that the "**View channels**" gadget has no effect at the moment, the module always analysis *channel 1* (even for stereo or multichannel files), this has to be fixed in the future.

Also note that due to a GUI bug, sometimes the display disappears, then you should resize the window a bit. The display is always normalized so the strongest frequency band corresponds to zero decibels. You can drag the mouse pointer over the display to see how strong a certain frequency is. Keep alt Alt (Option) key pressed to freely move in the freq/amp plane.

Analysis Windows – See Appendix D for information on Windowing.

last modified: 29-Jul-04

BASSO'S NOTES: This module is overall fantastic. Great for in depth analysis of files, viewing overall spectra and power distribution. Also useful for comparing samples that have undergone similar processing with different settings. H. Scale and V. Scale help to see either peaks or averaging. Engage both to focus on input's spectrum. Engage H. Scale to see peaks more easily. Engage V. Scale to see amplitude distribution across entire spectrum as opposed to engaging both which will focus graph on available spectrum only. Integer rounds the numbers to an integer (I believe) which can be useful for viewing a rough estimate of the peak harmonics. This works better visually with simple spectra (it shows a sort of "Staircase" graph with the "step edges" roughly at the peak harmonics.) Overall this works best with simpler harmonic spectra whereas very complex (approaching noisy) spectra tend to average out too much and show more a "sloping hill" graphically. To see this try analyzing a Sawtooth wave viewed in integers and compare that to white noise.

I tend to analyze always with the largest window size and compare the results with different windowing. However there may be times when smaller sizes are necessary. This will become illustrated as we move forward but suffice to say for now slower waves (bass heavy material) tend to respond better to smaller frame sizes.

Sometimes I will even "screenshot" the analysis graph to keep as a reference if I am dealing on a micro level of spectral processing.

Unary Operator

Sample based operations like the Binary Operator but just with one input file

Useful for cutting parts out of a sound file and for converting Fourier output from *Cartesian* ("rect") to *polar format* and vice versa.

last modified: 29-Jul-04

Binary Operator

Two input files (real or complex) are connected via a simple binary operator on a sample by sample basis, like adding, multiplying, taking the minimum etc. It can also be used to create a trimmed version of a sound.

Input File 1: First sound file, check '[Imaginary]' to provide complex files such as generated by the Fourier Translation.

Offset/Length: Allows you to trim the processed input file. **Positive offset** cuts away the beginning of the sound, **negative offset** inserts silence before the start of the sound. Length means total length (including offset).

Drive: Used for balancing the two input files: applied before processing.

Rectify/Invert: **Rectify** takes the absolute value of each sample, **Invert** inverts all samples (positive samples become negative and vice versa). Inversion allows the 'Addition' operation to become effectively a 'Subtraction' etc.

Input File 2: Same settings for the second input file.

Output File: Output of the binary operation between input 1 and input 2, possibly complex.

Operator: The kind of sample-by-sample operation performed between the two inputs (sometimes represented by the letters 'a' and 'b').

'Re' and **'Im'** stand for real and imaginary part in case of complex files.

'Power' means input 1 to the power of input 2.

AND/OR/XOR are binary logical operators performed by scaling the inputs to 24 bit(?) integer samples. May be useful in the fourier domain.

'Gate' means to take sample from input 1 if it is greater than sample from input 2, otherwise zero. 'ArcTangens' is arctan(input1/input2)/pi. Useful in the fourier domain(?) or for images.

Output Mix: Optionally mix the operator output with the dry input 1 signal (possibly inverted for subtraction).

Note that not all operators are implemented at the moment, some don't work with complex files.

last modified: 02-Aug-02

BASSO'S NOTES: Another favorite. Binary is great for summing together files in interesting ways. Take a sample pack (any will do but a mix of functional and abstract sounds – say drum samples and found sounds – will yield interesting results) and start combining sounds. Experimentation is key!

Try layer the same sample with various offset on one. Try this with different operators.

Adding with an Inverted input will subtract it from the other

Here is a useful example. You have a sound that is fully rectified (unipolar) and want it to be Bipolar. Add it to a copy of itself inverted and delayed by a small amount (otherwise it will just cancel out.) The delay (offset) amount will alter its spectrum so experiment to find the best compromise.

Amplitude Shaper

This module lets you change the amplitude envelope of a sound. You can remove it, replace it or superimpose a second one.

Input File: Sound whose amplitude shall be modified.

Envelope File: Sound file whose amplitude shall replace or superimpose the original amplitude: becomes active when **Source** is set to **'Sound file'** or **'Envelope file'**.

Output File: Modified input file.

Envelope Output: Check to create a separate file that contains the original input envelope. This is useful when you want to restore the original envelope after intermediate processing. You can use this envelope file (of ordinary sound file format) as a source of **'Envelope file'** for a different Amp-Shaper process.

Operation Mode: In the Source choice gadget you can select which envelope you want to apply 1) the input file itself (needed to remove an envelope) 2) the one calculated in place from another sound file 3) the one written as a separate envelope output (see parameter 4) or 4) one taken from a graphical envelope editor (**this doesn't work yet**).

<u>You can feed the module a soundfile containing an audio-frequency sine and choose 'Envelope file' as a source to create ring modulation.</u>

'Inversion' takes the inverse of the chosen envelope (loud parts become silent and vice versa). The envelope can either replace the original one or superimpose it (i.e. the destination envelope is a combination of the original and the new envelope). Use the combination 'Source=Input file'/'Inversion'/'Superimpose' to flatten the envelope of a sound file.

Maximum Boost: Replacement of an envelope can cause almost silent parts to become very loud which may not be desired (because you get just loud (quantization or environmental) noise. *This limits the amount of amplitude boost.*

Smoothing: Determines the range over which the momentary amplitude is calculated. Affects both input file envelope calculation (in 'Replacement' and 'Source=Input file' mode) and modifier file envelope calculation ('Source=Sound file'). Does not affect a 'Source=Envelope file'. The amplitude is calculated as a RMS value (i.e. all samples are squared, summed, and then the square root is taken. It is therefore proportional to the short time signal energy).

Envelope: not yet implemented.

Known bugs: You should normalize the output. Because the amplitude is calculated as RMS you would otherwise get very silent or clipped files depending on the application mode.

To be done: Max attenuation parameter, graphical envelope working, different measuring modes (e.g. a peak mode or a windowed mode).

last modified: 03-Aug-02

BASSO'S NOTES: It is easy to overlook this little module. Being able to impose an envelope onto a sound can be useful especially in Macro manipulation. Note that you can also save sound files to be used as Envelopes – there are a few modules in FScape that output "information" files and when you can do so it is prudent to build your own collection of files for future experimentation.

This module also doubles as a Ring Modulator/Amplitude Modulator – select Source=Envelope File and start with a simple SIN wav (Pure Data is indispensable for this) at 250Hz. Any file can be used as the modulation source so experimentation is key!

Lastly the process of "Flattening" a sounds Envelope can sometimes yield a louder average than normalizing alone. It really depends on the dynamics of the input material but Amplitude Shaper can sometimes help to reduce dynamics. That is to reduce high volume and raise low volume sounds.

Resample

You can either resample a sound to be of shorter duration/higher pitch or longer duration/lower pitch or use this module for sampling rate conversion. (Based on a paper by J. Smith)

Input/Output files: The sampling rate of the input is the reference value for the "**new rate**" field.

New Rate: Destination sampling rate. For resampling check the "**Keep old rate**" checkbox and enter the amount of semitones in this field. Due to the logic of the conversion positive semitone shift corresponds to a lower pitched output and vice versa. To get a sound with a pitch one octave above the original one choose "-12 semi". For one octave down choose "+12 semi". For sampling rate conversion select the absolute "Hz" unit and enter the desired rate. Uncheck the "Keep old rate" gadget.

Rate Modulation: Activate the checkbox if you want the rate to change dynamically over time. In the Param Field right to the checkbox choose the maximum deviation relative to the "new rate". Click on the envelope tool icon to edit the modulation curve.

Distinct Right Channel: Activate if you want a different modulation on the right channel. If the input file has more than two channels the first envelope corresponds to channel 1, the second one to the highest channel, all other channels are linearly interpolated. *Note*: Due to round-off errors you might end up with a severe phase offset between two channels. Also at the moment the envelope is sampled at an interval of 4 milliseconds and not continuously.

Desired Length: For resampling instead of entering the pitch shift you can enter the destination file length. The "new rate" field is updated automatically and vice versa. At the moment it is not possible to specify a destination length when the rate is modulated.

Keep Old Rate in Header: It means that the output file's header contains the same sampling rate as the input file. This is how you do resampling. E.g. if your input is 44.1 kHz and you choose a new rate of "+12 semi" then with this gadget checked you get an output at 44.1 kHz which is one octave down. When you uncheck this gadget you get an output at 88.2 kHz sounding exactly like the input when played back properly at 88.2 kHz.

Quality: The algorithm performs a band-limited sinc interpolation which is the most accurate method for resampling. Unfortunately a sinc is of infinite length, therefore we have to truncate it. Shorter FIRs speed up the processing but result in worse quality and a broader lowpass transition band. The FIR is stored in a table, if "**Interpolate**" is checked the table entries are interpolated.

last modified: 18-Feb-02

BASSO'S NOTES: Quick and dirty pitch shifting. Check "Change Pitch/Speed" to pitch shift input. Remember due to its logic positive values shift DOWN while negative values shift UP. So +12 semitones shifts the entire input down by half. No time compression here just old-skool pitch-down slow-down.

Not to be overlooked here is the pitch shifting by envelope! Selecting the check box next to "New Rate" allows you to define a change in pitch over time. The envelope GUI is a little clunky at times however it is simple and uncomplicated. You can click in extra nodes. When creating the envelope try using a constant SAW 250Hz as the input to easily evaluate the results.

This module will easily find its way into your workflow. Essential when we explore the VooCooder later and want to combine octave shifted copies.

Spectral Domain

FIR Designer

A little patch panel where you can place and connect simple filters for low/high/bandpass and bandstop. The output is a finite impulse response file that can be used in the Convolution module.

File Name: The FIR Designer creates an ordinary sound file that contains the impulse response of the particular filter. This sound file can be used as the IR in the Convolution module. Since you normally won't listen to it, 32-bit floating point is usually the best choice for output resolution.

Circuit Panel: The desired filter is built up from elementary sinc-filters (brickwall highpass/lowpass/bandpass/notch). You can assembly any number of those filters in series or parallel to create the desired frequency and phase response.

Double click on the panel to add a new basic filter. Double click left or right to an existing filter block to create a serial setting. Double click above or below an existing filter block to create a parallel setting. Very often you only need one filter block. Alt+Click will erase one block. The black line symbolizes the input/output wire. To change the type and settings of a filter block, just click on that block (it turns blue) and adjust the settings on the right (parameters 3 to 6).

Type: Basic filter type. 'Allpass' means a flat spectrum (a unit impulse, possibly delayed and attenuated). 'Subtract' means that filter is subtracted from a unit impulse, therefore turning a lowpass into a highpass etc. All filters are windowed sinc-filters as designed by the Kaiser method. No rolloff can be specified at the moment.

Cutoff/Bandwidth: Margin from passband to stopband. For bandpass and notch you also have a gadget for the bandwidth. In that case the cutoff is the middle frequency - e.g. if cutoff is 400 Hz and bandwidth is 200 Hz, the low frequency of the bandpass becomes 300 Hz and the upper frequency becomes 500 Hz. Cutoff becomes a geometric middle frequency (Sqrt(F_upper*F_lower)) when the bandwidth is given in semitones - e.g. if cutoff is 400 Hz and bandwidth is 12 semitones, the low frequency of the bandpass becomes 283 Hz and the upper frequency becomes 566 Hz. *This is really awful and should be replaced by a direct parameter for lower/upper cutoff frequency.*

Gain/Delay: When building a circuit consisting of multiple elementary filter blocks these define the attenuation and delay of each block. *For example:* place one allpass with zero delay and unity gain in parallel to an attenuated lowpass delayed by 1000 ms to create a tapestyle delay effect IR.

Overtones: For bandpass and notch checking this gadget will cause the designer to generate multiple bandpass with the same settings but each spaced from the fundamental bandpass as given in 'Spacing' parameter until the "overtone" frequency reaches the 'Limit freq'.

Filter Length/Window: In theory the sinc-filters are infinitely long. Therefore to realize them in a computer they have to be truncated at a reasonable number of taps. The higher the quality setting the longer the FIR will be resulting in shorter transition bands. Often, however, it is not desirable to have that narrow transition widths, so you might choose a low "quality" as well. The longer the filters gets, the more time smearing is produced by the filtering.

After truncation, a window (an envelope rising from zero to maximum and ending again at zero) is applied to the filter. The windowing is a compromise between the amount of stopband ripple that results from the truncation and the width of the transition band. Try out different windows and watch the result in the Statistics module's spectrum view. Often Blackman is a good choice.

Normalize Gain: When checked the output file will have 0 dB peak, when not, the filter is just written "as is". That means the filter gain is untouched, so you can subtract for example a sound convolved with say a low pass filter from the wet signal and get exactly the highpass signal. Often desirable when floating point output is written.

last modified: 03-Aug-02

BASSO'S NOTES: FIR Designer may seem daunting at first but it provides easy access to custom shaped filters to use in the Convolution Module. In terms of workflow I have a folder called IR in which I save all the outputs from this module. Most of the time the IRs are custom tailored to a sound but all the same it speeds up your workflow to have them all easily located in the same place.

Usually I will analyze a sound in Statistics and then decide what I want to do (remove DC, attenuate above 12KHz, etc.) Then start building the filter. I usually specify parameters in Semitones (and usually in octaves and half octaves) or 0 rolloff will give your brickwall response. Try different window types and filter lengths and update the analysis window to see the filter response. The response graph will look similar to Statistics but note that window types directly affect the output response as opposed to just changing how the graph appears. I always check Keep Minimum Phase.

Try a serial chain of filters with different settings to either smooth or sharpen the response.

Convolution

Convolution is the one of the fundamental algorithms in digital signal processing. It is equal to filtering with a finite impulse response. Examples for filters include frequency equalization, reverb, delay, creating an output that contains the spectral information common to both the input and the impulse response.

Input File: Sound to be filtered with an impulse response.

Impulse Response: Second sound file to be used as the "impulse response" for the FIR filtering. Naturally input file and impulse response can be exchanged while the output is the same. Use the shorter file for impulse response to make processing less RAM expensive. An impulse response can be a "normal" filter IR like a low pass, high pass etc. (something produced by the FIR Designer module), response from a delay or reverberation system or any other sound snippet.

Convolution means the spectral characteristics of the IR (both amplitude spectrum and phase response) are superimposed onto the input file. When using normal sounds the output tends to emphasize the low frequencies unless the IR has a flat spectrum. It tends to emphasize the frequencies that exist both in the input and the IR and cancels those frequencies which are not present in either file.

Output File: Input convolved with IR.

Mode: '**Convolution**' for normal operation: spectra multiplied. '**Deconvolution**' spectra divided.

Unless all frequencies are present in the IR you get a lot of boost at some (usually the high) frequencies. Also there's a problem with the overlap-control of the algorithm, so deconvolution hardly works (well, you can convolve a sound with another one, process it and try to deconvolve it again which should work).

'Conv w/ inversion' means the IR spectrum is modified in the way that the strongest frequencies become very weak and the weak frequencies become strong (phases not changed), sometimes works as a kind of deconvolution.

Truncate Overlaps: FScape technically convolves by STFT spectra multiplication and overlap-adding of successive frames. The truncated overlaps used to be a bug in an early version. Sometimes I liked the "buggy" results so now you can reintroduce that bug by checking this gadget.

Normalize Impulse Energy: This should be placed in the **'Morphing'** section. It means each IR calculated in the morphing process is normalized so there's no change in the IR energy over time.

File Length: A normal non-circular convolution results in an output that is as long as the sum of the input length and the impulse length (minus one sample).

Often you want create an output file that has the same length as the input, then choose **'Input (no change)'**.

When FScape finds a Marker named **'Support'** in the impulse file, then the first "**support**" samples of the convolution are omitted and then the remaining last samples; if not only samples at the end of the convolution are omitted. Choose

'without support' to skip the support of the IR. The **'Support'** marker is written by the FIR Designer module which generates symmetric sinc-filters; the strongest sample is the one in the middle and that's exactly the support, meaning that attacks in the input file remain on their position counting from the beginning of the file.

Morphing: Means the IR file is split up into an integer number of equal length impulses.

By editing the envelope you can define which of these impulses is used for filtering at which moment in time (simplest case is a straight line from 0% to 100% over time meaning that the first IR is used at the beginning of the input file and the last one is used at the end).

The **'Policy'** determines how the IRs are interpolated. **'Rect X-Fade'** is normal linear interpolation. **'Polar X-Fade'** tries to interpolate amplitudes and phases separately which sometimes gives more interesting results (mode not really worked out). **'Correlate and shift'** doesn't work at the moment.

The **'Window size'** determines how long an interpolated IR stays valid. For fast movements you'll need a small window size to avoid clicks, for slow movements use bigger windows to increase processing speed. The **'Overlap'** helps to avoid clicks. Not perfectly worked out.

last modified: 03-Aug-02

BASSO'S NOTES: Can't go wrong with convolution. Unless you don't know what you are doing. It can be used for very basic utilities (simple filtering as explained in the FIR module notes) some creative work (reverb and "modeling" techniques based around convolution) or very abstract processing for spectral combinations.

For beginners keep one of these "modes" in mind while working with Convolution. Are you using it for basic filtering, Convolution Modeling or Abstract Experimenting.

For most basic/utility uses you will use "Input (no change)" – Filtering and certain Modeling. For uses that involve long trails where there was none before (think a dry SAW convolved with cathedral reverb IR) keep the length to "Input+IR".

Experimental uses of convolution will be discussed in greater detail in an appendix. But true experimental convolution will require a more sophisticated convolution module that allows for altering the IR – such a module is not included in FScape (and really is outside our scope) but some examples would be Space Designer by Apple or ReaVerb by Reaper.

That being said FScape does allow endless possibilities to create sounds to use experimentally in convolution plugins. I will expand about some ideas and experiments in an appendix.

Complex Convolution

This works exactly as the normal convolution module, however it's capable of using arbitrary length impulse responses, and using complex input (a combination of real and imaginary files such as output by the fourier translation module).

The normal convolution module needs to load the impulse response completely into the RAM, which can take up a lot if you bear in mind that sample resolution is 32bit internal, impulses can be multichannel, and for convolution the buffer needs to be twice the size of the IR. For impulses with durations greater than 30 seconds or one minute, the module may break down with an out-of-memory exception. In this case, you can use the complex-convolution module which is performing a partitioned convolution.

You specify the amount of RAM in megabytes that you wish to dedicate to the process. Depending on this value and the size and number of channels of the input files, the amount of steps into which the process is split, is shown in the text field right to the "Mem.alloc" gadget. When the step size increases way beyond 32 or so, the process may take up a really long time, so try to increase memory in this case.

When you leave the imaginary inputs unchecked, you perform a normal (real) convolution. Note that in version 0.68 the ceptral options are hidden because they never really worked.

last modified: 02-Jun-05

BASSO'S NOTES: As already stated this module allows for convolution with very large files or with imaginary files. Imaginary files (usually derived from Fourier Translation Module described next) allow for faster convolution (as in time spent processing). There is probably other advantages (mathematical) to using the imaginary files in convolution however at this current time these specifics are unknown to me.

Fourier Translation

Jean Joseph Baptiste Fourier became famous for his theorem known as the Fourier Analysis. In his model each signal (e.g. a sound) can be represented by a weighted sum of sines and cosines. The Fourier Analysis (more precisely the discrete Fourier transform) calculates those sine/cosine coefficients which we call the Fourier spectrum. The forward transform is paralleled by the possibility of a lossless inverse transform (the synthesis). This module allows a translation from one domain to the other. Once you transformed a sound to the Fourier domain you can apply all algorithms that you would normally apply to the time signal. Finally you can go back to the time domain.

Input File: Time domain or frequency domain signal. Generally the Fourier transform is defined for complex signals, i.e. those represented by complex numbers. Complex numbers are made of a real and an imaginary part. Because ordinary sound file formats do not support complex numbers, I decided to use separate files for real and imaginary parts.

Usually you start from a real time signal (uncheck "imag" in the input) and do a forward transform resulting in the complex spectrum (check "imag" in the output). Then you manipulate the spectrum (e.g. crop a portion) and translate it backward again (here you supply both real and imag so you should check "imag" for the input) to get a time signal (often real so uncheck "imag" for output).

Output: Time domain or frequency domain depending on the direction chosen. Note that this module always assumes complex signals independent of the checkboxes. Therefore a "complete" spectrum is calculated: It starts at 0 Hz (DC) and goes up to half the sampling rate (the Nyquist frequency, for 44.1 kHz sounds this is equal to 22050 Hz) followed by descending negative frequencies until we finally reach 0 Hz again.

Direction: Forward for time input/ fourier output; backward for fourier input/ time output.

Spectral format: Complex numbers can not only be represented by a sum of a real and an imaginary part (rect) but also as a 2D vector with a length and an angle. At the moment only the rect format is supported.

FFT length: The discrete Fourier transform is carried out via a speed optimized algorithm called the Fast Fourier Transform (FFT). In the most simple form it required the signal vector to be of the size of a power of 2 (i.e. 2 samples, 4 samples, 8 samples, 16, 32, 64 etc.). If the input sound's frame number is not a power of 2 you can choose to either truncate the sound to the next lower power of 2 or expand it by adding a certain amount of silence.

Frequency scale: This was just a test. Don't try to change the default value, you will almost certainly get stupid results.

last modified: 25-Feb-02

BASSO'S NOTES: Unfortunately I have not had much success or spent much time with JJ Fourier. Simple experiments such as converting to Fourier applying say Kriechstrom then converting back have not proved successful. I believe it is the processes I am using that are not yielding to the backwards transform.

Paying closer attention to the few modules that actively support Imaginary files may be the key to more successful results. For instance using them in the Binary Module and comparing the results of the same process with the regular file versions.

Also needs more experimenting with Convolution.

Wavelet Translation

Transform from time domain to a subband time-frequency scalogram . Wavelets are small "packets" of signals that can be thought of as high- and lowpass filters. A wavelet transform applies those filters recursively to a signal.

This module contains a pyramidal algorithm: The signal is high- and lowpass filtered and decimated (half of the samples are "thrown away"). Then the lowpass filtered signal is again put through the high/lowpass stage and so on until the size of the decimated subband signal is very small (the size of the filter kernel). What you get is a multiresolution subband coded version of the signal, a mixture of a time domain and a frequency domain representation.

High frequency information has bad frequency resolution and good time resolution, low frequency information has a fine frequency resolution and a bad time resolution which somehow corresponds to our auditory perception system. Like the Fourier transform the transform is invert-able. Unlike the Fourier transform the signals are always real and not complex and the filter has compact support (i.e. the coefficients are localized in time while the fourier transform uses sinosoidal filters that have an infinite duration).

Input and output file: Time domain or wavelet domain signal depending on the direction chosen. The output of the forward transform is that of a pyramid: We start with the few coefficients of the last lowpass filter stage followed by the last highpass filter. Next we find the highpass coefficients of the successive scales (going from subbands belonging to lower frequencies to those belonging to high frequencies). Each successive subband is twice as long as its predecessor.

Filter: Although an infinite number of filters (wavelets) exist you can only choose between a few that are widely used and were introduced by Ingrid Daubechies. The number after the name corresponds to the filter order (number of coefficients).

The difference between the higher order filters is very small while the Daub4 is most different from the others. Try to use different filters for forward and backward transform. Unfortunately the module requests that the input file for a backward transform be of a certain size (the exact size of a forward transform with the same filter). Be careful when you process the forward transformed files because changing the length of the file may cause FScape to reject the file for backward translation. This will be fixed in the next versions.

Gain per Scale: Often the coefficients belonging to low scales are much higher than those belonging to high scales. This can cause problems for integer output files because it will introduce high quantization noise in the high scales. Try to tune this parameter so that the volume of the forward transform output will remain approximately constant over the whole file (alternatively use floating point files).

Direction: Forward for time input/ fourier output; backward for fourier input/ time output.

Interleaved processing: This is the result of a bug in an early version. By checking this gadget FScape transforms multichannel files not channel by channel but as one big stream of interleaved samples. Beware of mono incompatible phase problems when manipulating the wavelet files!

Known bugs: No special treatment of the signal boundaries. Nonzero coefficients beyond the boundaries are thrown away at the moment. For large signals this does not seem to be a problem, however.

Contents   last modified: 25-Feb-02

BASSO'S NOTES: Like the Fourier module the benefits of this module are largely unknown to me. Recently I realized outputs from this module can be used in the Blunder Finger module effectively speeding up its process (Blunder Finger is a slow process) and probably increasing time accuracy. I am guessing at this but most likely there are complex mathematical reasons why working in the wavlet domain inside Blunder Finger is preferred.

Hilbert Filter

A Hilbert transformer takes an input signal and calculates two output signals that have a constant 90° phase difference across the whole spectrum. Applications are single sideband modulation (SSB) also known as frequency shifting and envelope generation.

Input File: The file to be processed.

Output File: The 90° outputs can be thought of as the cosine and the sine part of the signal or as the real and imaginary part alternatively. When doing the pure transform without modifications you'll get two output files. When performing frequency shifting or envelope generation the output is real (single file).

Operation: What should be done after the hilbert transform. You can leave the signal unchanged, shift the spectrum up or down or calculate the sound's envelope.

Antialiasing: When frequency shifting is applied we will normally want to have a protection against aliasing. Aliasing occurs for example when you shift a signal downwards by 100 Hz and the signal has significant energy below 100 Hz. Think of a partial at 30 Hz. When you downshift it by 100 Hz the mathematical outcome is -70 Hz and physically you'll hear it at +70 Hz. By checking this gadget all signal content that would generate aliasing is eliminated.

There are applications where you want to have aliasing. For example try to shift a signal up or down half the sampling rate (22050 Hz for a 44.1 kHz sound) - when you allow aliasing the result will be an inverted spectrum. (1000 Hz becomes 21050 Hz, 2000 Hz becomes 20050 Hz etc.).

Note that because of the antialiasing filter the bass frequencies are slightly attenuated. Try to feed this module with white noise and see what comes out. If you don't like it apply a bass compensation filter afterwards.

Shift Amount: The "**shift up**" and "**shift down**" operation. Note that frequency shifting is not pitch transposition. That means that harmonic relationships are destroyed resulting in bell-like spectra with non-linear partial spacing, the kind of sound you know from ring modulation.

Shift envelope: Not yet implemented!

Contents   last modified: 25-Feb-02

BASSO'S NOTES: Oh the Hilbert Filter. Never a dull moment with the Hilbert Filter. Frequency Shifting can yield awful results or it can yield wonderful new sounds, alien textures, and otherworldly "scapes".

Hilbert does not preserve harmonic structure – but you may be asking why that is. Simply consider this: 100Hz and 200Hz versus 1000Hz and 2000Hz – both sets are octaves (2:1 ratio) however the distance between them differ greatly (100Hz for the first pair but 1000Hz for the second.) Now Imagine a sound containing these 4 partials – yes an odd sound however these harmonics are integer relations – Fundamental 100hz 1st Overtone 200Hz 9th Overtone 1000Hz 19th Overtone 2000Hz. Now shift this up by 100Hz resulting in: 200Hz 300Hz 1100Hz 2100Hz – here there is no integer relationship between the harmonics and thus any Harmonic structure has been destroyed.

Now keeping this in mind if you wish to do frequency shifting and maintain the spectral structure you are better off trying the Laguerre Module (which has its own idiosyncrasies but more on that latter.)

There is an expression along the lines of if you can't fix it break it further. This is one philosophy when tackling Hilbert. Consider it a radical altering of input material into new material rife for furthering processing.

Furthermore you can use Hilbert as you might pitch shifting – just don't expect the results to be anywhere near the same. Great for conjuring up sounds from the "digital waste-bin." Try extreme shifting (up or down) to see what new tiny snippets of audio have emerged (evolved?)

Mosaic

A technique that tries to imitate photo-mosaic style. Given a visual template of the desired sound (essentially a sonagram) and some input sounds, try to spread chunks of the input sound across the time-frequency plane of the template.

This algorithm is inspired by the idea of Synthrumentation by Klarenz Barlow. It was developed for the piece "Achrostichon" dedicated to Folkmar Hein.

The template time-frequency plane is given by a black-and-white sonagram **Image input** file. Frequency is considered logarithmically spaced.

Small number of bands (e.g. 1 to 3 per octave) usually produce more interesting results than a literal sonagram. The sound material used to "paint" the template is given by the **Audio input** field. Note that this file must be a magnitude longer than the target sound file's duration. If necessary, use a sound editor to concatenate the sound material to itself a couple of times to produce the necessary length.

The target duration is given by **Nominal duration**. It is called "nominal" because the actual duration may differ a little bit due to the randomization processes. The time resolution of the input image is automatically derived from the nominal duration, e.g. when the image has a width of 100 pixels and the nominal duration is 10 seconds, each pixel corresponds to 100 ms.

The frequency span of the input image is given by **Min freq**. and **Max freq**. The bands per octave are automatically derived from these settings and the input image's height.

If you take the photo-mosaic metaphor, the algorithm now reads in chunks from the input sound file as "photos" and places them on the time-frequency canvas. The overlapping in the time-frequency plane is specified by **Time Spacing** and **Freq Spacing** (where 100% means dense packing without overlap, 50% means 1x overlap etc.). To produce less regular rhythmic and harmonic output, use some **Time Jitter** and **Freq Jitter** which is a ratio of the time chunk length and frequency band coverage in percent, by which the "photo" maybe moved from the nominal position.

The smoothness of the "grains" or "photos" depends on the fade-in and fade-out of the chunks, as given by **Attack** and **Release**.

White pixels in the input image constitute the loud parts, black pixels the silent parts. The corresponding minimum volume is given by the **Noise Floor** parameter. The parameter **Max Boost** specifies the maximum gain that can be applied to each "photo" to make it match the template pixel's volume.

Finally, if your input sound material has already been segmented, you can use markers to choose the photo-boundaries instead of the fixed-size time windows resulting from nominal duration divided by image-width. Each new marker in the input sound file marks the beginning of the next photo chunk. To use this approach, check the **Read Markers** option.

last modified: 10-Jun-09

Sonagram Export

Converts a sound file to a sonogram image file.

The sonogram is created using a Constant-Q transform, resulting in a logarithmically spaced frequency axis.

Brightness depends on the logarithmic amplitude in decimals, therefore a Noise Floor must be specified.

The frequency resolution is given by the number of Bands per Octave, whereas the Max Time Resolution is given in milliseconds.

The actual temporal resolution in the bass frequencies is a trade-off with the steepness of the frequency filters, producing some temporal smearing. The FFT size is limited by Max FFT Size. The smaller this size, the less temporal smearing occurs in the bass frequencies, but the worse the frequency resolution in the bass register.

The image is written as a gray-scale TIFF file.

BASSO'S NOTES: The outputs from this module cannot currently be used in the Mosaic module. The Mosaic module does not respond to tiff files.

The image composition is defined by the frequency content of the source sound file. You can define the limitations by setting the **Lowest Freq** and **Highest Freq.**

Signal Ceiling and **Noise Floor** set the upper and lower thresholds for detection**.** It helps to remember that loudness relates to white pixels while silence relates to black pixels.

Bleach

Employs an adaptive whitening filter, using a simple LMS algorithm.

Be careful with the feedback gain. Values above -50 dB can blow up the filter.

---

last modified: 18-Feb-10

Spectral Patcher

A panel similar to Reactor and the like where you can place spectral processing modules and connect them together. Double click to open a popup window showing all available modules. Click+Drag a module to change its position. Alt+Click+Drag on a module to drag a connection line from source module to target module. Alt+Click on the connection line to delete the connection. Double click on the connection line to add a helper breakpoint. Double click on a module to open its settings dialog. Ctrl+Click on a module to bring up a popup menu. The popup menu allows you to delete, rename, copy, paste and duplicate modules. Alias modules cannot be edited but always share their settings with its originators.

Modules very brief: Input+Output file read and write spectral data files, you cannot load or save sound files with them (use Analysis/Synthesis instead). You can load analysis files from SoundHack or save analysis files for later processing (e.g. using the Convert-Files module). I'm not sure if FScape will accept CSound analysis files?

Splitter and Unitor will split one input into several outputs and vice versa. Envelope will calculate the spectral envelope of its input. Constrast will emphasize strong partials. Cepstral calculates the cepstrum and high pass filters it, ideally removing the resonances, but I never got it working right. Zoom is nice and will just shrink the spectrum but taking out a specified portion, the output file will be shorter and pitched up just like in resampling. Shrink is similar but "resamples" the spectrum. Flip-Freq will work only with tonal sounds and not noise, it can alter the harmonic relationships by reversing bands and stretching/ shrinking them. Tarnish is very nice and works well with speech, it attenuates harmonics and thus kind of keeps the noisy parts. Convolve: convolution in frequency domain, which is multiplication in time domain, but not necessarily circular convolution. Mindmachine, you can create nice spherical sounds. Smear will repeat STFT frames with a given attenuation. Extrapolate doesn't work. Percussion is nice: It captures the resonance of each spectral frame. Choose a very high frame size, like 65556 samples; then use the synthesized output sound file as a filter for convolution.

Bugs: The transition from java AWT to Swing was kind of cheesy so there are some display bugs. Resize the window if you get confused. When you abort the processing, sometimes restarting it is not immediately possible. Simply stop it and restart it again. There are bugs in many of the modules that keeps them from terminating at the end. Then you'll have to stop the processing manually. All output files should be accessible as normal.

last modified: 29-Jul-04

Multi-Band

Band Splitting

This module takes an input file and writes up to nine output files which are generated from tunable bandpass filters. Useful for emulating multiband behavior with fullband algorithms, e.g. dynamics.

Input File: full band input sound file.

Output File: prototype file name. FScape inserts a number before the type extension running from 1 to the desired number of bands. E.g. "OutputSplt.aif" would cause FScape to create the files "OutputSplt1.aif", "OutputSplt2.aif" etc.

Normalize Each File: When checked each split band file is normalized separately. If not checked, normalization is first applied to the loudest output file and all other gains are adjusted in order to maintain the original relative gains of the bands (i.e. summing the files together restores the original file).

Number Of Bands: The spectrum of the input file is split up into this number of adjacent bands that cover the full spectrum.

Maximum Boost: Replacement of an envelope can cause almost silent parts to become very loud which may not be desired (because you get just loud [quantization or environmental] noise.) This limits the amount of amplitude boost.

Quality: determines the length of the FIR bandpass filters. The name is badly chosen because in many applications you will want to have "Low" quality, i.e. short filters to minimize time smearing. Also "better" quality will increase processing time. For 3 or 4 band splitting as a preprocess for separate dynamic treatment "low" is a good choice. The higher the quality the narrower the transition band can get (see parameter 7).

CrossOvers: The (n-1) frequencies dividing the spectrum into (n) bands. E.g. choosing 3 bands with crossovers 1 kHz and 5 kHz will result in three files, the first being the lowpass up to 1 kHz, the second being the bandpass 1 kHz to 5 kHz, the third being the highpass above 5 kHz.

Rolloff: determines the width of the transition between adjacent bands. 0% means no transition/hard cut. The rolloff of the bandpass will be as small as possible depending on the quality setting. 100% means the frequency response of each band is in the shape of a raised cosine (cos^2), this is good for separate dynamic processing of the bands because the transitions are smooth enough to conceal the multiband nature of the processing.

last modified: 02-Aug-02

BASSO'S NOTES: Band splitting can be useful both to emulate multi-band processing and as a simple filter. When using 2 bands the *crossover* acts as the cutoff giving you a lowpassed signal and a highpassed signal with *rolloff* acting as the filter slope. Use three bands to emulate a bandpass filter (by discarding the high and low parts and keeping the middle file.)

Unless I am mistaken there is no efficient way to merge the band split files. If you have only 2 bands you could use Binary Operator to merge the two. If you had more than two files you could still use Bin-Op but would be limited to doing two at a time. However, limitations offer us a chance to improvise, and in this scenario combining band split parts using Bin-Op offers the opportunity to use various operations to combine.

Chebychev Waveshaping

An input file is split into bands which are fed into a waveshaping stage that produces a mix of user adjustable harmonics. (Based on a paper by Fernandez)

Waveshaping refers to a process of feeding a wave into a shaper function which can be visualized as a kind of table that maps input values to output values. If the mapping function is a line, the process is linear which means the input signal is not distorted (a sine input will produce a sine output), it's just attenuated or amplified. Any other shape will distort this signal which means it produces a complex mixture of overtones, if your input is a sine signal. If your input then is something different from a sine, you get additional intermodulation distortion, for example if you input a mix of two sines of different frequencies, using a nonlinear waveshaping function will produce both overtones of each of the sines plus those for the sum and the difference of the frequencies of the individual sines.

Feeding the input through a filterbank that tries to separate all sine components of the input signal, and then feeding each bandpass filter output into a separate waveshaper, will minimize the intermodulation distortion.

Now there is an interesting set of functions which are useful in waveshaping: Chebychev Polynomials. If the input is a pure sine of normalized amplitude (1.0), feeding this input through a Chebychev function will produce an exactly predictable mix of overtones. In this module, instead of dealing with the polynomials directly, you see a graphical representation of the mix of overtones.

The mix of overtones is specified separately for low frequencies, mid frequencies and high frequencies on the one hand, one for the beginning of the input sound file, the middle of the sound file and the end of the sound file, on the other hand. Therefore you find nine different panels for adjusting the overtone mix, each allowing to specify the fundamental (red color, leftmost bar) and the first seven overtones (from left to right). By pressing the mouse-button over the panel and moving it up or down, you can adjust the amount of each generated overtone. The scaling of the overtone strength was intuitively chosen so that a middle position means like "quite reasonably audible", 25% means "quite weak", 75% means "quite strong" etc. The buttons on the right allow you to quickly copy the current panel (indicated by a small blue border) to all other times (vertically, i.e. beginning / middle / end of the soundfile) for that frequency part, or to all other frequencies (horizontally, i.e. bass / mids / highs) for that time part.

The lowest frequency put in the waveshaper is specified by the "Low Freq" field. Frequencies below this frequency will be copied unmodified to the target soundfile. Note that the lower you set the low frequency, the slower the processing will become, because the bandpass filter for very low frequencies need to be very "long" (in terms of number of filter coefficients). The "Mid Freq" specifies the frequency at which the overtone mix of the three middle column panels are applied. The "High Freq" then specifies the top most bandbass filter frequency for the waveshaping process: everything above this frequency is copied unmodified to the target sound file.

Waveshaping is automatically adjusted so that no frequency aliasing can occur. For example, if your top-most frequency is 11 kHz and your soundfile's sampling rate is 44.1 kHz, frequency aliasing would occur above 22 kHz which is just above the first overtone of 11 kHz. So no matter what overtone mix you choose in the high frequency panels, the output of a bandpass filter at 11 kHz will not be excited with more than the first overtone!

While the top row of the panels corresponds to the beginning of the sound file, the middle row corresponds to the time specified in the "**Mid Time**" gadget. The bottom row corresponds to the end of the soundfile.

The overtone mix is automatically interpolated between bass/mid/high frequencies and between beginning/middle/end of the soundfile.

The closer the partials are in your input file, the steeper and closer the bandpass filters must be to effectively separate them, i.e. the more filters per octave you need and the longer the filter kernel need to be specified. There is nothing wrong, however, in using a very little number of filters per octave and short filters, only you get must more intermodulation.

Due to the nature of noise, using input soundfiles with a lot of noise or transient contents will produce strangely modulated outputs, simply because there are no sine components which can be clearly separated. In this case, try to use only few filters per octave to produce a less artifical sounding output and to just use the module as a brightener or exciter.

If this module is new to you, the best practice is to play around with a steady or swept sine input file! Use a sonagram to review the results for different overtone settings.

last modified: 02-Jun-05

BASSO'S NOTES: Chebychev! How do you even pronounce that? I must admit that due to my native tongue I have a certain deficiency in pronunciation of foreign terms: Kriechstrom being a prime example. Since I rarely talk about FScape out loud to anyone it is mainly in my head that I refer to the modules – and thankfully in the Vimeo Hans made for FScape he pronounces Kreichstrom (although in my head it's still Kreek-strum). But Chebychev wasn't German he was Russian! Something like *chebysheff* might be a more accurate pronunciation, although in my head it's often *Cheby*.

Why the long preamble? Because this module deserves some deep consideration and a special place in your artillery.

The Chebychev module epitomizes the very nature of FScape: complex "non-musical" ideas being applied with very musical outcomes. You do not need to know or understand the math in order to use Chebychev Waveshaping, like all FScape modules you only need to tweak, review, repeat.

Chebychev is great for generating melodic material where there was none (or perceivably none). Run a drum loop through it to get instant melody than can be used to great effect in your sound design. To best understand Chebychev let's take a look at a potential workflow.

Start with a drum loop, pick a few harmonics to use, and make them consistent in all 9 windows. Stick with 12 bands per octave, length medium, mid time should not matter if every window is identical – now process then review. What do you hear? How does it sound?

Ideally you should hear clear distinct tones being coaxed out of the material. If you hear very noisy elements (and you don't like them) then you may be hearing intermodulation distortion – or something along those lines. Try reducing the bands per filter – even down to 6 may alleviate the problem. You may also adjust the low frequency to act as a highpass filter and prevent any bass information from interfering. Adjusting the filter length (longer most likely) or adjusting the various harmonics may help.

The 9 different windows give you the ability to tailor the distortion over time. Once you get a handle on Chebychev as a static waveshaper (as described above) you can begin to add time and frequency variations by utilizing all 9 windows.

Chebychev is excellent at adding new harmonics to sounds but usually less complex tones are easier to work with. Try running a SAW wave through Chebychev and you may find it only brightens it, if anything. Try running something too complex (like a completed track), and you might be left with modulated mush. Get your bearings by using Chebychev every time you use FScape – you will learn about Chebychev's

Exciter

A frequency exciter that generates a signal that is one octave above the input; very slow but good for rich harmonic and not too noisy sounds.

Nowadays superceded by the Chebychev Waveshaping!

Input/Output files.

CrossOvers: The frequency band between these margins is split off and taken to the excitation path. Excitation of frequencies above half Nyquist (e.g. 11025 Hz for 44.1 kHz samples) is not possible. The smaller the low crossover is the longer the FIRs will get naturally and therefore processing time increases. Besides it hardly makes sense to excite frequencies below 300 or 400 Hz.

Rolloff: The part above the low crossover is highpass filtered before taken to the excitation stage. The rolloff specifies the transition bandwidth of this highpass filter. Usually one octave is a good choice (i.e. if low crossover is 400 Hz, then the highpass has full attenuation at 400 Hz and no attenuation at 800 Hz).

Bands per octave: The exciter works by splitting up the signal into narrow frequency bands and modulating these bands by themselves (producing second order harmonics and DC which is removed). The bigger the number of bands the longer the filter needs to be (parameter 6) and therefore the worse time smearing becomes (Note that 72 bands per octave are senseless when using a short filter length). The smaller the number of bands the better the attack quality is maintained while inter-harmonic distortion increases. The number of bands also proportionally influences processing speed.

Dry/Wet Mix: In "normal" excitement applications the newly generated harmonics tend to be much more quiet than the dry signal. In other cases you want to get the "solo" excitation which means you should set the dry mix to 0% and wet mix to 100%.

Filter length: The bands are split up using windowed sinc FIR filters. Long FIRs produce a lot of time smearing which means your attacks sound more like waterdrops than like clicks. When the harmonics are well separated you can use a small filter length (or in case you want interharmonic distortion), vice versa. Long filters need much more processing time unfortunately.

last modified: 03-Aug-02

Voocooder

Like a vocoder, but more voodoo and more cool, thus the double 'o's. Actually the vocoder frequency band combination mode is not the best thing you'll have heard in your life. But the other ones can produce very nice noisy sounds.

last modified: 29-Jul-04

Time Domain

Step Back

The input file is split into segments by an adjustable energy and spectrum investigation. The segments are then rearranged in reverse or random order.

Documentation not yet written!

last modified: 18-Feb-02

Kriechstrom

The input sound is split into chunks (small grains or greater pieces) specified by **Min/Max chunk length**.

Min/Max simultaneous chunks specifies the density of the texture (inverse proportional to rendering speed). You can choose whether chunks may be repeated (Min/Max repeats) and if you allow repetition, if the chunk length should be altered on each repetition ("**Instaneous length update**").

The maximum distance in the original sound file between two chunks is given by "**Max entry offset**", where increasing the offset significantly slows down the algorithm.

"**Filter amount**" should be set to 0% (50% default is bad) because any other value will add a random low or high pass filter (depending on the colour setting) which tends to make all output sound bubbling.

last modified: 29-Jul-04

Serial Killa

Though serialism is something considered history of the 50s, it's still present in contemporary music in a subtle way. Many people still believe that the important dimensions of sound are pitch (above all) and duration. Look at MIDI if you want to know where this kind of thinking will lead to. Some other people believe that pitch sucks completely and that the structure of noises are much more interesting. However when you have a traditional score, all you see is pitch and duration. Even a sonagram is difficult to interpret in terms of noise and so on. I was trying to gather information from some pieces regarding noisiness and the like.

This module will analyze (in a rather intuitive and sometimes bad way though) some sound characteristics: **Noise**, **Tilt** (spectral focus) and **spectral energy** (A-weighted).

Besides its analysis character the output files (which are normal audio files only at a very low rate) can be used in conjunction with the Seek+Enjoy module.

last modified: 29-Jul-04

Seek + Enjoy

An input sound is considered to be analyzed by some means, the analysis output being saved in a control file. This module then rearranges the input sound according to the values in the control. E.g., if the control file contains the energy envelope of the input sound, chunks with high energy will be rearranged to appear at the beginning of the output file, those chunks with low energy (according to the control file) will be shifted to the end. Try to use outputs of the Serial-Killa module as control files.

The idea was to for shift all noisy parts to the beginning of the sound and all harmonic parts to the end. However this never really worked, but you get some idea at least.

The control file is always resampled to match the length of the input sound. The **segmentation** settings help the algorithm to break the input sound into chunks. The higher the **minimum chunk** length, the more continuous the output will be. The higher the **tolerance**, the longer chunk lengths grow "naturally".

In ascending order, those chunks corresponding to high values in the control file will appear at the beginning of the output file, in descending order this relationship is reversed. Try very small chunk lengths and zero tolerance for that famous New York Giuliani kind of sound.

last modified: 29-Jul-04

Lückenbüßer

Creates a mixture of two sounds by repeatedly crossfading them at each of their zero crossings.

The process starts with input A until a zero crossing occurs, it then scans input B and finds a zero crossing in B. it writes A up to the crossing and performs a crossfade for the specified length. Both zero crossings are chosen to have the same tangent signum.

The algorithm gives up after the span specified as "**patience**" and does a crossfade in this case after the patience span. The process stops when one of the inputs is exhausted.

It might also be that the algorithm doesn't look for zero crossings but for intersections between A and B. (have to look at the code to find out).

The idea was to create a mixture of two similar (phase correlated) sounds, but they run out of each other very quickly; often not too interesting results. The idea is nice but the algorithm should be modified by introducing "magnetism" between time offsets in A and B.

last modified:04-oct-06

Pearson Plotter

Triggers a sound on a cross-correlation basis

An input file is correlated with a pattern file. Whenever the cross-correlation exceeds a given threshold a third sound is placed at the trigger point.

last modified: 18-Feb-02

Sediment

Spreads crumbles of one file (pattern) across the timeline of another (control).

A multipass process is applied: for the duration of the pattern file, sequentially a grain is extracted from this file, having a random duration within the given limits. Next, the whole control file is traversed and the pattern grain correlated against it. The "best" matching position is memorized and the grain is "plotted" at this position in the output file, potentially scaled due to the time-scale setting.

Automatic gain control is employed to match the pattern and control grain, where the boosting of the pattern grain is limited by the max-boost setting. To avoid clicks, the envelope is applied to each pattern grain, using a fade-length determined by the win-size setting.

Determination of the "best" match in multichannel situations is controlled by the multichannel-correlation setting. The **clump parameter** can be used to speed up processing. The clump size determines how many grains are generated for each pass through the control file. At the small disadvantage of having several grains of exactly the same size, this clumping saves FFT calculations in the generation of the control file spectra.

last modified: 18-Feb-10

Dr Murke

Collects chunks of an input file according to a weighting function provided by another (control) file. When then control file is some kind of amplitude or loudness measurement of the input file, this can be used to keep only the loud or only the silent parts of a sound.

Input and control file do not need to have the same length. In that case, indices in the control file are translated to indices in the input file by applying a constant (stretch) factor. Typically the control file is an output from the Serial Killa module.

The module distinguishes between an up-track and a down-track; that is a stream of sound where the corresponding control signal stays above a given threshold or below a given threshold.

The "**Mode**" setting determines whether the up-track or the low-track is written. If the control-file is an amplitude measurement, keeping the down chunks means keeping the more silent parts of the input sound, and keeping the up chunks means keeping the rather loud parts.

The module initially starts at the "off" track; that is the track that is not written. For instance, if the up-chunks are kept (the loud parts), the parameters are initiated at a down-chunk position: For a loud part to be detected, the control signal must for a minimum period given by the "**Min. Up Duration**" stay above the "**Up Thresh**" threshold. When this happens, the input signal is faded in using the "**Attack**" fade-time. The module is then on the up-track and stays there until for at least the duration given by "**Min. Down Duration**" the control signal stays below "**Down Thresh**". In that case, a fade-out is applied using the time parameter "**Release**". The spacing between "**Up Thresh**" and "**Down Thresh**" form a kind of hysteresis. You can also think of the module as a kind of Noise-Gate.

The "**write-head**" of the output file is stopped at the moment a fade-out (release) begins, and restarted at the moment a fade-in (attack) occurs. Note that the attack phase occurs before the position at which the control-signal crosses from off-track to on-track. For example, if the last fade-out would finish at 3 seconds in the output file and an on-track attack is detected at input position 6 seconds, assuming an attack time of 100 ms, this means that the write-head resumes at 2.9 seconds plus spacing (see below) and begins to copy the input signal from 5.9 seconds into the input file. This is so that when a loud part is detected, the actual attack is not lost or blurred.

For each attack the write-head between successive fade-out starting positions and fade-in ending positions is advanced by a given spacing function. This can be either a constant ("**Fixed Spacing**") or the special setting "**Original Spacing**". In the latter mode, all chunks from the input file are written exactly so that write-head and read-head match, so you end up with a file that has the same duration as the input file, and all kept sounds occur exactly at their original positions, with off-track chunks being "wiped out".

last modified: 03-Nov-10

Distortion

The Distortion offering in FScape is extensive and offers opportunities for unique processing.

Notable distinctions are the ability to use a file input as modulator in Frequency Modulation module - while it can be hard to tame it offers unique opportunities.

Laguerre Warp can be process intensive and is technically imperfect with its comb filtering artefacts but with that in mind it can be used to create some wonderful metallic "alien" textures and timbres.

Ichneumon is perfect for injecting some randomness in pitch/time and is useful for adding variations into a heavily processed sound to be cut into samples.

Rotation can add some spectral "crunch" or in its alternative mode it can offer some dirty time stretching - the dirt being occasional strong audible artefacts.

Seek+Destroy gives some basic but useful SIN or TRI waveshaping that can add some warmth and max amplitude or extreme distortion. Finally it too allows a soundfile to be used as the shaping function.

Frequency Modulation

Frequency modulation creates spectral sidebands. A source signal, the carrier, is rate modulated (resampled) at a frequency determined by the modulator signal.

Carrier Input: The source to be modulated.

Modulator input: Signal that modulates the rate of the carrier. This gadget is only active if "Soundfile" is chosen as the modulation source.

Modulation source: Either an internally generated signal (sine, tri, saw, rect) or a user supplied sound file. Tri, saw, rect are not yet implemented.

Quality: FM is performed via resampling. See also the Resample module.

Modulation depth: The maximum resampling ratio. Refers to the input sampling rate.

Oscillator frequency: Frequency of internally generated sine, saw, tri and rect.

last modified: 18-Feb-02

BASSO'S NOTES: Tuning the modulator and carrier is the key to success. Inform your decisions from the spectrum of your input - Statistics module is indispensable for this. When you find a good pairing of mod + car try different Mod depths.

The option to use a soun-dfile as modulator offers opportunities for unique results. However, care must be used in choosing the modulator file. Try altered versions of the source file - pitch or frequency shifted/Cheby'd. Once you have a good pairing it may be worthwhile to alter the modulator in the time domain (Krieckstrom or Stepback) This may possible yield a rhythmic FM effect as well as timbral changes.

A similar idea with rhythmic modulation could be to use a percussive loop as the modulator although again pitch/timbre or the modulator will impact the results the most.

You can also use simple waveforms (SAW/TRI/REC) to achieve some classic FM.

One finally point: I am unsure about the units for "Mod Depth". I am inclined to use percentages but semi-tones seem to respond better.

Laguerre Warping

The Laguerre Transform is used to alter the harmonics of a sound. Individual frequency components are warped to other frequencies based on a mapping curve which can be adjusted by the user. (Based on a paper by F. Evangelista).

Here you cannot adjust the curve but it kind of applies a constant factor to most frequencies, thus preserving harmonic spectra usually. The algorithm is related to the discrete Fourier or chirp transform which means it's slow. Also because of the chirp property, transients are destroyed (time domain aliasing).

For transient sounds small frame sizes are good, for rich harmonic sounds greater frame sizes are good.

I found out, that the best results are obtained if you split the sound into two bands (using Band Splitting), where crossover is ca. 250 Hz. For the bass part use 1024 or 2048 samples frame length, for the high passed band use 512 samples; this is a good compromise between frequency distortion and transient preservation, though it slows down the process quite a bit.

"**Warp amount**" is linked to the input/output freq. gadgets beneath. So if you want to see to which frequency 1 kHz gets mapped at a warping of -10%, first specify "-10%", then enter 1000 Hz in the input freq. field. Hit return and the output is shown in "**Output freq**." (You have to select an input file first because this only works if the sample rate is known). Alternatively, if you want 1 kHz mapped to 1500 Hz, type 1000 Hz in input freq., then 1500 Hz in output freq., hit return and the required warp amount is calculated.

"**Overlap**" is tricky, you have to play around with different values. The problem is because of the stretching of each sound frame, in the recombination of the warped frames comb filter effects occur. They change with different overlap settings but virtually never disappear. This has to be fixed in a future version.

last modified: 29-Jul-04

BASSO'S NOTES: While it can be processing intensive the results are always interesting if not useful.

As noted it always imparts some amount of comb filtering but as I alluded to earlier this can be intentionally exploited to achieve wild timbres: metallic, alien, unnatural reverb, synthetic or even Sci-Fi.

When this is not desired try different overlap settings or even Frame size but this will increase processing time. As with all FScape modules it is best to run the same process with various Frame Size and overlap settings (noting what they are) to determine their effect. Changing both of these parameters can drastically alter the timbre.

Ichneumon

Segment a sound into tiny bits separated by zero crossings - go back from each zero crossing until you find a local maximum or minimum. Go forward until you find a local maximum or minimum. The apply a resampling to this part of the wave.

Min/ Max RSMP: The minimum and maximum resampling factors. There is a decision algorithm inside that tries to find out whether a wave crossing a zero is "fast" / "transient" or whether it moves slowly. Depending on this analysis, a resampling value between min and max is chosen. Can't remember now which was which.

Quality: length of the sinc interpolation for the resampling algorithm. (see Resample-Module).

last modified: 29-Jul-04

BASSO'S NOTES: Here we find a great source of random variation and real extreme shaping.

Initially you will want to try different MIN MAX values keeping in mind the + values pitch down while - Values pitch up (I believe.) Try different combinations of upward or downward shifting with either MIN MAX.

Some important things to note: due to the nature of the process (as described above) it responds best to clear distinctions between sounds but it also seems to alter resample based on the envelope of the sound.

Example: a drum loop will essentially trigger a different fixed resample value for every distinguishable sound occurrence. A chord sequence of held notes or even fast changes but legato will trigger far fewer resample variations.

One alternative is to run a sound through the Kriechstrom with fixed spacing that offer more distinction between sound occurrences.

With extreme MIN MAX values or techniques as described here you will most likely want to cut up the Ichneumon results for use as single shot or loops. When approaching Ichneumon as described here I am usually moving on to final chopping/selecting of sounds from the results.

However with more care paid to the MIN MAX values and choice of program input you can get subtle glitch results.

Needlehole Cherry Blossom

The idea is to provide a generic module for window based time domain processes. There's only one process at the moment which is a median filter. In image processing you use median filtering to remove noise, it has a kind of low pass filtering characteristic. Thus, by specifying a very short window length like one or two milliseconds and subtracting the dry signal, you kind a very nice insect highpass filter.

last modified: 29-Jul-04

Rotation

Segment a sound into tiny bits separated by zero crossings

1) Go back from each zero crossing until you find a local maximum or minimum. Also go forward until you find a local maximum or minimum. Then reverse this part of the wave both horizontally and vertically and make some proper adjustments so it perfectly fits into the original wave.

2) Simply repeat this small wavelet a couple of times giving a strange but sometimes interesting time stretching possibility (picking up an idea I got from Trevor Wishart which I guess is in CDP).

last modified: 29-Jul-04

BASSO'S NOTES: Rotation can give some beautiful time expansion albeit sometimes rife with artefacts and distortion.

Try processing some vocal samples with 2 repeats for interesting dirty pitch down. Further repeats yield wild and amazing sounds but quickly losing resemblance to the original.

Sometimes you may get a great time expansion/pitch shift but with audible artefacts. One possible solutions (although it will alter the sound) is to Hilbert to sound so the Artefacts are close to the "top" and then LP the file then shift it back to its original value.

In Rotate mode it is useful to subtract the dry to get the results of the process which could be further altered and added back to the original (Binary Op) or possibly used as a modulator in FM or in place of a control file for other modules.

Seek+Destroy

Seek+Destroy can be operated as a waveshaper (distortion) or as a "seeking" process with arbitrary locator control file.

last modified: 18-Feb-02

BASSO'S NOTES: This module offers processing that may be familiar to most - waveshaping. The SIN and TRI functions offer great results for adding gain and warmth or all out distortion. Try SIN or TRI Bipolar Centered for the least destructive processing. Wrapped UNI and SQR Functions offer more extreme hard clipping distortion.

Not to be missed is the options to use a sound file as the Function. Most likely the results will be very noisy but like everywhere else try altered versions of the input.

I suggest always using the DC Blocking because the module can really bring out or introduce DC OFFSET.

Blunder Finger

This module applies a basic genetic algorithms on the sample level.

The general algorithm follows the pattern:

1. [Start] Generate a random population of chromosomes (possible solutions)

2. [Fitness] Calculate the fitness f(x) of each of the chromosomes x of the population

3. [New population] Use the following steps to generate a new population:

a. [Selection] Choose two parent chromosomes, by weighting their chance of getting selected with the fitness (the better the fitness, the greater the chance to be selected)

b. [Crossover] According to a probability of crossover, mix the genes. It is possible that the offspring is a 1:1 copy of one parent.

c. [Mutation] According to a probability of mutation, mutate the genes at each defined Locus

d. [Accepting] Add the offspring to the new population

4. [Replace] Use the new population as the base population of the next iteration.

5. [Test] If a condition for stopping the algorithm has been defined, check the condition and stop if necessary, return the solution.

6. [Loop] Otherwise, go back to step 2

In FScape, the algorithm is interpreted as follows:

A chromosome consists of a time window of a given number of samples (*Chromosome length*).

The original population is created by reading a corresponding time window from an input sound (*Population input*). For a given size of population (parameter *Population*), the time window in the input sound is shifted sample-frame by sample-frame. E.g. if chromosome length is 32 and population is 8, the first individual is read from sample frames (0...31), the second individual from (1...32), etc., the eighth individual from (7...38).

A number of iterations of breeding is carried out (*Iterations*), consisting of:

Calculating the fitness of each indivdual of the current input population, by comparing it with a corresponding time window in a fitting sound (*Fit input*), and calculating the root-mean-error. This is performed in the time domain sample-by-sample, or, if *Domain* is set to "Wavelet" by transforming both population and fitness sound chunk into the wavelet domain.

For multichannel sounds, according to the parameter *Multichannel fitness*, either the worst-case the best-case or the mean of all the channel fitnesses are taken.

Generate the next output population by choosing *Population* parents from the input generation, with a probability depending on their fitness, and breeding each offspring. To breed offspring, the chromosomes of the two parents are cut at *Crossing points* positions and mixed. Afterwards, with a probability given by *Mutation probability*, the resulting offspring is mutated. Mutation is performed, by the noise amount given by *Mutation amount* to the chromosome.

The output population of the iteration is used as input population of the next iteration

The resulting population of these iterations is written out to the target sound file. Each chromosome is first highpass filtered by differentiating, the concatenated output stream lowpass filtered by integration, to prevent clicking at chromosome borders.

If *Elitism* is checked, incest is introduced in the iterations, by copying directly the "best" two parents from one population to the next.

last modified: 10-Jun-09

Miscellaneous

Sachensucher

This takes any input file and tries to spot what kind of sample data it could be (tries to find "wordlengths" and sample encoding (integer or float)). Not sure about the "windowlength" but I think that's the minimum length for analyzing the most probable "sample format". Not sure about the "headroom", maybe a volume factor.

sometimes nice with uncompressed images and uncompressed movies.

last modified: 04-oct-06

Recycle Or Die

Emagic Logic Audio has a very useful function which is called "Optimize sound file". It means, it checks which parts of a sound file are used in a song and then compacts the sound file by rewriting it using only the used parts and an optional safety bound (this process is called "Compact sound file" in ProTools). When you work with very long field recordings and the like, so sometimes like to recycle the recordings, i.e. do exactly the opposite of "optimizing" -- just remove all the parts that have been used and keep the unused sound bits. That's what the Recycle module will do for you. Only you have to provide both the original sound file and the compacted sound file. The output will be the parts that are in the original but not in the compacted sound file.

Min. Matching: The minimum length of sound chunk that must be identical in the original and compacted sound to be considered used (and be thrown away).

Compact Spacing: At least in version 4 (don't know if this is still in Logic 6), Logic was putting a few milliseconds gap between the chunks in the optimized file which would confuse my algorithm, thus you can specify a small skip-over chunk that will be ignored between the compacted bits.

Min. Recycling: Each chunk not found in the compacted version must be at least as long as specified here to be kept in the output file. Thus, higher values mean "don't recycle small bits".

Padding: Same as the safety bounds in "Optimize", only for the recycling.

Write Markers: If checked, markers will be placed between recycled chunks, so you could split the bits later on in Peak or so.

last modified: 29-Jul-04

Convert files

Map data from one medium to another, like convert a sound file to an image etc.

Input / Output: The Convert-Files module was I think the first one in FScape so it was made I think in 2000 or 2001. I haven't changed it since then and rarely have used it, I'm not sure if it is still working after the migration to MacOS X. The direct conversion from sound file to image file didn't work in my recent test. However what you can do is create a spectral file with the Spectral Patcher (or use Sound Hack), then convert the spectral file to a TIFF image (and back).

Conversion Settings: So assuming you selected a spectral file as input and a TIFF file as output, you see the following settings: "Seperate colors for each channel" means you can have a mono, stereo or three channel input file, where mono sounds will be converted to grayscale images, stereo and 3-channel will use the R/G/B channels of the image. "Handle overhead", I'm not exactly sure what it was, but I think it asks you how to separate amplitude and phase spectra. In "horizontal tiling" mode, the left half of the image will be the amplitude spectrum, the right half will be the phase spectrum. In "separate files" mode I guess you get two files. Also note that because images files are stored row by row, time elapses from top to bottom of the image, so frequency is spread from left (0 Hz) to right (smprate/2).

The "**Channels**" gadget allows you to discard some of the input channels. Usually use "Leave untouched". The "**Noisefloor**" is important, because most image processing programs (at least at the time the module was written) could only handle 8-Bit images. So to preserve as much sound quality as possible, the amplitude spectrum is converted to logarithmic scale, whereby the energy at noise floor corresponds to an RGB value of zero. Try playing around with different values to see the effect of the noise floor. Only when going back from image to sound you should take care that you choose the same noise floor again.

last modified: 19-Feb-10

CD Verification

I use this module to verify the quality of burned CD Audio media. The common burning program on Mac OS X, Toast, is not capable of verifying them.

While all good players will automatically use error correction when reading CDs, you can import CDs with error correction disabled to see how good the medium actually is. On Mac, you can do this for example with iTunes by disabling the error correction in the importing preferences. This module will compare a re-imported audio CD with the original soundfile which was used for burning. When verifying multi-track CDs, you have to concatenate all re-imported tracks first, using the Concat module (zero offset, zero overlap, 100% chunk length, no crossfade). CD players/recorders turn out to produce a variable gap before the actual audio content when re-importing the tracks. This gap can be as low as 24 samples (half millisecond; measured on an external firewire burner) or as high as almost 700 samples (equivalent to 15 milliseconds; measured on an iMac G4). FScape will automatically correlate the two files and skip the initial gap in the re-imported file. If for some reason, you get an error message saying that FScape failed to sync the files, you could try to increase the maximum initial gap size.

This module will not produce any audio output. Instead it posts a message in the console when verification was successfully completed (i.e. original and copy are 100% identical) or shows an error message if discrepancies where encountered. If you check the "Quick Abort" gadget, the process terminates immediately, if at least one sample in both files differed, otherwise verification is carried out to the end of the files.

Due to rounding errors when concatenating the files or such, you may encounter discrepancies on the least-significant-bit (LSB) of the two sound files, something you will usually ignore in verification, so the "Ignore LSB toggle" gadget is checked by default. Note that when a reading error occurs when re-importing the audio CD, discrepancies are always way higher than the LSB, usually between -80 dBFS and -40 dBFS, so it's safe to keep this option checked.

"What if my CD wasn't verified?" - Well, that's up to you. You may wish to use other media, or you just ignore the result because you rely on the error correction of CD players, or you just throw the CD away before shipping them to people, if you run a label or this is a CD you send for an important application ;-)

last modified: 02-Jun-05

SMPTE Synthezier

- NO DOCUMENTATION –

Beltrami Decomposition

Converts a (two-dimensional) image file into a sound file by using single value decomposition as a technique of dimensional reduction.

I have used this experimental module in the sound installation Amplifikation to create rhythmic sounds from scanned sheets of hand-writing.

Restoration

Comb Filter

A quite simple IIR filter that creates resonances at a basic frequency and all multiples of this frequency. Useful for hum removal.

Waveform I/O: The usual input output stuff.

Frequency and Q: A comb filter will notch a frequency and all its linearly spaced overtones. I don't know if this a bug of my algorithm but the notch spectrum is shifted upwards by I think 50% of the fundamental frequency. Thus for hum removal if you choose 50 Hz, you'll get notches at 75 Hz, 125 Hz, 175 Hz etc. So for effective hum removal you'll have to shift the input sound up 25 Hz using the Hilbert, Comb filter it, then shift it down again. But every tiny audio app has a good hum filter nowadays, so I'm not sure if you'll want to use this module at all. The Quality is the amount of feedback in the filter, higher Q's give deeper notches.

Dry/Wet mix: You can subtract the original sound from the comb filtered sound by increasing the dry mix and inverting the original signal.

last modified: 29-Jul-04

Declick

Digital click artifacts are detected by a stochastic comparision and/or removed by filling in the silence with a short reverberation.

Input/Output: Input sound containing clicks and declicked output file.

Detection Sources: 'Let FScape detect' activates this module's own mechanism for finding clicks in the input. Controlled by the parameters 3 (see below).

'Read markers' includes clicks located by Markers named "**Click**" in the sound file (e.g. placed in Peak). Therefore, if FScape fails to find clicks itself, you can place manual markers and then use the Declicker only to repair these ones.

Detection settings: Active when **'Let FScape detect'** is checked. Detection is based on the assumption of a Gaussian distribution of sample changes (difference between adjacent sample values). Mean and standard deviation are calculated from frames defined by **'Check size'**. Then each sample change is threshold by the chosen **'Probability bound'** (e.g. if this is set to '1:2000' it means FScape detects a click if there's a sample change in a frame that has a chance of less than 1:2000 to appear in a Gaussian distribution with the calculated mean/stddev). Because such irregularities occur often in very silent regions which are hardly audible as "clicks" you have to choose a resonable noise floor (**'Min. amp'**).

Repair Modes: **'Let FScape repair'** activates this module's own mechanism for repairing clicks in in the input. Controlled by parameters 5 and 6 (see below).

'Write markers' inserts Markers named 'Click' in the output file, so when you uncheck **'Let FScape repair'** and check **'Write markers'** you can manually repair the detected clicks (for example with the click repair function of Peak).

X-Fade Size: When you choose FScape to repair the clicks, it will fadeout the input just before the click and fade it in after the click. In order to conceal this fadings the last **'X-Fade size'** samples before the click are convolved with an IR specified in parameter 6 (see below) and added to the clip location.

ImpulseRresponse: Normally use the file that comes with FScape ('sounds/declickIR.aif'). That is the IR of a short reverb. Therefore when a click occurs the portion before the click is reverberated and covers the attenuated click location. Try small delays as well. The IR should have a quite flat spectrum.

Click View: A visual feedback while processing. Indicates clicks as red vertical lines whereby the horizontal position corresponds with the timeline position in the input file.

last modified: 03-Aug-02

Schizophrenia

This module creates a stereo file from a mono source by convolving it with another sound and using the result as the "side" signal in a MS matrix.

last modified: 18-Feb-02

Special

Batch Processor

This module lets you create a list of modules which will be processed one after another. It is helpful because the modules cannot be linked directly. E.g. you can assemble a forward fourier transform followed by amplitude flattening and inverse transform. Hit the "Start" button and go to sleep for a couple of hours..

Batch list: Click "Add" to add a new entry. The list is processed top down. Elements can have different command types. To change a command type keep the mouse pressed over the "Command" column. The lines are numbered for convenience. The "Object" column contains the command arguments. The "On Error" column specifies the behaviour if the command fails, i.e. if a module aborts or a file deletion cannot be completed.

Command "Module": Choosing this type will bring up a dialog with all currently opened module windows. Selecting a module and hitting "ok" will copy all the module's settings. Changing the module window's setting after this "paste" action will not have any effect. If you need to modify the module's parameters again, you'll have to create a new module dialog by hitting the "Edit" button and re-adding the module to the batch list.

Command "Delete file": You can use this to delete temporary files which aren't used any more. The full path has to be given in the "Object" column. Usefull to free harddisk space during extensive batches.

Commands "Begin Loop" and "End Loop": These command must be added as corresponding pairs. Command lines between a begin/end block are repeated. The number of repeats is given in the "Object" column of the begin command. The format is something like "A = 1 TO 9" which means the block is repeated 9 times. The looping variable, i.e. "A" in this case, can be used to specify variable filenames in the block, see below. The "End Loop" command must have the same variable name as the corresponding "Begin Loop" command.

Command "Label": Labels are target positions in the batch that can be accessed in two ways: in the error handling configuration of a module you can specify that if an error occurs the batch should jump to a certain label. The name of the label is given in the "Error Label" column and the "On Error" column must be set to the "Skip To" item. Besides you can use the batch command "Skip to" to jump to a certain label.

Command "Skip To": Branches to the given label name.

Editing the batch list: New batch commands are inserted by hitting "Add", the default command is a label and can be switched by keeping the mouse pressed over the "Command" column. Cut/Copy/Paste can be used to delete or duplicate items or to move them to a different position in the batch list. Dragging items is not possible at the moment, therefore to move an item, cut it, select the predecessor row of the target position and hit "paste". The "Edit" button brings up the parameters window of a module command. Changing the parameters has no effect on the batch list unless you re-add that module.

Module Parameter Settings: This table contains all known file arguments of a module and is a convenient mechanism for adjusting the paths of input/output files of a module without having to re-open the dialog using the "Edit" command. Usually you'll build a reusable batch list and then subsequently only exchange the initial input files and final output files. Between looping statements, placeholders can be used. For example inside a loop of variable "A", any path name can contain the string "$A" which will be replaced on the fly by the integer value representing the loop iteration. This is usefull for processing a bunch of files with the same module. FScape will determine the maximum number of characters used by the iteration variable and left-pad with "0" characters. That means, if your loop variable runs from 1 to 5, the "$A" will be replaced subsequently by the strings "1", "2", ... "5". If your loop variable runs from 1 to 15, the "$A" will be replaced subsequently by the strings "01", "02", "03" ... "15". Keep this in mind when naming files.

Control Settings: "Visible modules" specifies that the corresponding module parameters windows be opened during the processing of a module. This can be a visual help for learning how batches work. However note that in this case the batch will be interrupted by visual requests (such as a gain overdrive warning dialog at the end of a module process)!
"Console Output" specifies that each batch entry that is processed be output on the console text display. In MacOS 9 the console would automatically be opened upon text output. This doesn't happen in MacOS X any more, the only way at the moment of using this feature is to launch FScape from a terminal window! This will be fixed sometime in the future.

last modified: 13-apr-04

Appendix A - BPM to Milliseconds

Tempo	1/4	1/8	1/8T	1/16		Tempo	1/4	1/8	1/8T	1/16
60	1000	500	333	250		120	500	250	167	125
61	984	492	328	246		121	496	248	165	124
62	968	484	323	242		122	492	246	164	123
63	952	476	317	238		123	488	244	163	122
64	938	469	313	234		124	484	242	161	121
65	923	462	308	231		125	480	240	160	120
66	909	455	303	227		126	476	238	159	119
67	896	448	299	224		127	472	236	157	118
68	882	441	294	221		128	469	234	156	117
69	870	435	290	217		129	465	233	155	116
Tempo	**1/4**	**1/8**	**1/8T**	**1/16**		**Tempo**	**1/4**	**1/8**	**1/8T**	**1/16**
70	857	429	286	214		130	462	231	154	115
71	845	423	282	211		131	458	229	153	115
72	833	417	278	208		132	455	227	152	114
73	822	411	274	205		133	451	226	150	113
74	811	405	270	203		134	448	224	149	112
75	800	400	267	200		135	444	222	148	111
76	789	395	263	197		136	441	221	147	110
77	779	390	260	195		137	438	219	146	109
78	769	385	256	192		138	435	217	145	109
79	759	380	253	190		139	432	216	144	108
Tempo	**1/4**	**1/8**	**1/8T**	**1/16**		**Tempo**	**1/4**	**1/8**	**1/8T**	**1/16**
80	750	375	250	188		140	429	214	143	107
81	741	370	247	185		141	426	213	142	106
82	732	366	244	183		142	423	211	141	106
83	723	361	241	181		143	420	210	140	105
84	714	357	238	179		144	417	208	139	104
85	706	353	235	176		145	414	207	138	103
86	698	349	233	174		146	411	205	137	103
87	690	345	230	172		147	408	204	136	102
88	682	341	227	170		148	405	203	135	101
89	674	337	225	169		149	403	201	134	101
Tempo	**1/4**	**1/8**	**1/8T**	**1/16**		**Tempo**	**1/4**	**1/8**	**1/8T**	**1/16**
90	667	333	222	167		150	400	200	133	100
91	659	330	220	165		151	397	199	132	99
92	652	326	217	163		152	395	197	132	99
93	645	323	215	161		153	392	196	131	98
94	638	319	213	160		154	390	195	130	97
95	632	316	211	158		155	387	194	129	97
96	625	313	208	156		156	385	192	128	96
97	619	309	206	155		157	382	191	127	96
98	612	306	204	153		158	380	190	127	95
99	606	303	202	152		159	377	189	126	94
Tempo	**1/4**	**1/8**	**1/8T**	**1/16**		**Tempo**	**1/4**	**1/8**	**1/8T**	**1/16**

100	600	300	200	150		160	375	188	125	94
101	594	297	198	149		161	373	186	124	93
102	588	294	196	147		162	370	185	123	92
103	583	291	194	146		163	368	184	123	92
104	577	288	192	144		164	366	183	122	91
105	571	286	190	143		165	364	182	121	91
106	566	283	189	142		166	361	181	120	90
107	561	280	187	140		167	359	180	120	90
108	556	278	185	139		168	357	179	119	89
109	550	275	183	138		169	355	178	118	88
Tempo	**1/4**	**1/8**	**1/8T**	**1/16**		**Tempo**	**1/4**	**1/8**	**1/8T**	**1/16**
110	545	273	182	136		170	353	176	118	88
111	541	270	180	135		171	351	175	117	88
112	536	268	179	134		172	349	174	116	87
113	531	265	177	133		173	347	173	116	87
114	526	263	175	132		174	345	172	115	86
115	522	261	174	130		175	343	171	114	86
116	517	259	172	129		176	341	170	114	85
117	513	256	171	128		177	339	169	113	85
118	508	254	169	127		178	337	169	112	84
119	504	252	168	126		179	335	168	112	84

Appendix B - Note to Hz Chart (440 tuning)

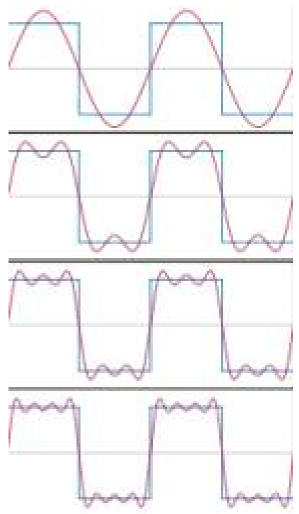
Note	Frequency (Hz)	Wavelength (cm)
C_0	16.35	2109.89
$C^{\#}_0/D^{b}_0$	17.32	1991.47
D_0	18.35	1879.69
$D^{\#}_0/E^{b}_0$	19.45	1774.20
E_0	20.60	1674.62
F_0	21.83	1580.63
$F^{\#}_0/G^{b}_0$	23.12	1491.91
G_0	24.50	1408.18
$G^{\#}_0/A^{b}_0$	25.96	1329.14
A_0	27.50	1254.55
$A^{\#}_0/B^{b}_0$	29.14	1184.13
B_0	30.87	1117.67
C_1	32.70	1054.94
$C^{\#}_1/D^{b}_1$	34.65	995.73
D_1	36.71	939.85
$D^{\#}_1/E^{b}_1$	38.89	887.10
E_1	41.20	837.31
F_1	43.65	790.31
$F^{\#}_1/G^{b}_1$	46.25	745.96
G_1	49.00	704.09
$G^{\#}_1/A^{b}_1$	51.91	664.57
A_1	55.00	627.27
$A^{\#}_1/B^{b}_1$	58.27	592.07
B_1	61.74	558.84
C_2	65.41	527.47
$C^{\#}_2/D^{b}_2$	69.30	497.87
D_2	73.42	469.92
$D^{\#}_2/E^{b}_2$	77.78	443.55
E_2	82.41	418.65
F_2	87.31	395.16
$F^{\#}_2/G^{b}_2$	92.50	372.98
G_2	98.00	352.04
$G^{\#}_2/A^{b}_2$	103.83	332.29
A_2	110.00	313.64
$A^{\#}_2/B^{b}_2$	116.54	296.03
B_2	123.47	279.42
C_3	130.81	263.74
$C^{\#}_3/D^{b}_3$	138.59	248.93
D_3	146.83	234.96
$D^{\#}_3/E^{b}_3$	155.56	221.77
E_3	164.81	209.33
F_3	174.61	197.58
$F^{\#}_3/G^{b}_3$	185.00	186.49
G_3	196.00	176.02
$G^{\#}_3/A^{b}_3$	207.65	166.14
A_3	220.00	156.82
$A^{\#}_3/B^{b}_3$	233.08	148.02
B_3	246.94	139.71
C_4	261.63	131.87
$C^{\#}_4/D^{b}_4$	277.18	124.47
D_4	293.66	117.48
$D^{\#}_4/E^{b}_4$	311.13	110.89
E_4	329.63	104.66
F_4	349.23	98.79
$F^{\#}_4/G^{b}_4$	369.99	93.24
G_4	392.00	88.01
$G^{\#}_4/A^{b}_4$	415.30	83.07
A_4	440.00	78.41
$A^{\#}_4/B^{b}_4$	466.16	74.01
B_4	493.88	69.85
C_5	523.25	65.93
$C^{\#}_5/D^{b}_5$	554.37	62.23
D_5	587.33	58.74
$D^{\#}_5/E^{b}_5$	622.25	55.44
E_5	659.25	52.33
F_5	698.46	49.39
$F^{\#}_5/G^{b}_5$	739.99	46.62
G_5	783.99	44.01
$G^{\#}_5/A^{b}_5$	830.61	41.54
A_5	880.00	39.20
$A^{\#}_5/B^{b}_5$	932.33	37.00
B_5	987.77	34.93
C_6	1046.50	32.97
$C^{\#}_6/D^{b}_6$	1108.73	31.12
D_6	1174.66	29.37
$D^{\#}_6/E^{b}_6$	1244.51	27.72
E_6	1318.51	26.17
F_6	1396.91	24.70
$F^{\#}_6/G^{b}_6$	1479.98	23.31
G_6	1567.98	22.00

$G^{\#}_6/A^b_6$	1661.22	20.77
A_6	1760.00	19.60
$A^{\#}_6/B^b_6$	1864.66	18.50
B_6	1975.53	17.46
C_7	2093.00	16.48
$C^{\#}_7/D^b_7$	2217.46	15.56
D_7	2349.32	14.69
$D^{\#}_7/E^b_7$	2489.02	13.86
E_7	2637.02	13.08
F_7	2793.83	12.35
$F^{\#}_7/G^b_7$	2959.96	11.66
G_7	3135.96	11.00
$G^{\#}_7/A^b_7$	3322.44	10.38
A_7	3520.00	9.80
$A^{\#}_7/B^b_7$	3729.31	9.25
B_7	3951.07	8.73
C_8	4186.01	8.24
$C^{\#}_8/D^b_8$	4434.92	7.78
D_8	4698.63	7.34
$D^{\#}_8/E^b_8$	4978.03	6.93
E_8	5274.04	6.54
F_8	5587.65	6.17
$F^{\#}_8/G^b_8$	5919.91	5.83
G_8	6271.93	5.50
$G^{\#}_8/A^b_8$	6644.88	5.19
A_8	7040.00	4.90
$A^{\#}_8/B^b_8$	7458.62	4.63
B_8	7902.13	4.37

Appendix C – Fourier Introduction

The Fourier Series

Decomposition of a Signal



The first four Fourier series approximations for a square wave.

A **Fourier series** decomposes any **periodic function or periodic signal** into the sum of simple oscillating functions. Any function which changes in time can be divided in single periodic signals. A non periodic signal can be considered as a succession of periodic portions of this signal. The Fourier transform can then also apply to non periodic signals, step by step, to allow this decomposition on each portion of the signal.

Application:

The **Fourier Transform** is an **algorithm** that can be used for the decomposition a sequence of values – an digital audio signal, for instance – into components of different frequencies. Hence, it can be applied to analyze the spectral components of a sound. The **Fast Fourier Transform** is a variant of the Fourier Transform, which allows the fast calculus of the components.

Analysis Window

Window Parameters and Analysis Resolution

Because the analysis is done step by step, it is based on a **window**. This window is applied to a number of **signal samples**, which determines its **width, or size**.

Windows are applied successively in time to the signal until the whole file is analyzed. The number of samples of each window is also divided in bars, or "bins". The number of bins is proportional to the number of samples in the window: it is the **FFT size**, which reminds us of the main principle of the Fourier Series. Consequently, the window size and the FFT size determine the resolution of the representation in **frequency** and **time**.

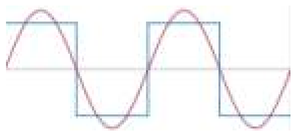
Overlapping Windows

To perform an efficient analysis, windows **overlap**. This "space" between each window is called the **window step**. To avoid clicks when overlapping the windows, the windows have a **windowing curve**, which defines the **window type**. These parameters interact with the **sample rate** of the signal.

Window Size

The **window size** represents a number of samples, and a duration. It is the main parameter of the analysis. The window size depends on the fundamental frequency, intensity and changes of the signal. The **FFT size** is a consequence of the principles of the Fourier series: it expresses in how many frequency bands the analysis window will be cut to set the frequency resolution of the window. The **window size** influences the temporal or frequency resolution of the analysis.

Reminder: Period and Frequency



Two periods in a periodic signal.

The FFT is based on a supposed periodicity of sounds.

The **period** (T) is the duration of a cycle, in s.

The **frequency** (F) is the number period per second. It basically determines the pitch of a sound.

T = 1/F

F = 1/T

For instance, an A_4 has a 440 Hz.

$T(A_4) = 1/440 \simeq 0{,}0023$ s.

$F(A_4) = 1/0{,}0023 \simeq 440$ Hz.

The lower a sound, the lower its frequency, the longer its period is.

Window Size Parameters

A window size is expressed in **samples**. This parameter is a variable. But we also have a fixed parameters, which is the **sampling rate** (44100 or 48000 samples per second, for instance). From this, we can calculate the other parameters of the window, such as its duration of frequency resolution.

Duration of the Window

The relationship between the number of samples of the window and its duration of the window) We know that the sampling rate (SR) of the sound corresponds to **1 second**, and that the period is inverse to the frequency.

T = Window Size/SR.

With a 1024 samples analysis window and a 441000 sampling rate, we have:

The duration of the window must be five time longer than the period of the signal, that is :

T(Window) = 5* T(Signal).

For instance, the window size for a 440 Hz signal should be:

5*(1/440) : 0,025

Each strip of the analysis will represent an image of the spectrum which will be worth 25 ms.

Lowest Detectable Frequency

The choice of the window size must be done considering the frequency of the signal. If these factors evolve, this must be taken into account. The lowest detectable frequency (F_0) is determined by the size – duration – of the window.

F_0 = 5*(SR/Window Size)

For instance, with a 1024 samples analysis window, we have :

$F_0 = 5*(44100/1024) \simeq 215$ Hz.

From there, the window size should be :

WS = 5*SR/F(Signal)

For a 440 Hz signal, we get WS = 501

For a 100 Hz signal, we get WS = 2205

The lower the pitch, the bigger the window size should be.

Window Size and Temporal Resolution of the Analysis

Calculating the Time Resolution (TR)

The duration of the analysis window, or **time resolution, is inversely proportionate to the frequency resolution.**

A sampling rate corresponds to a 1 second duration. The analysis window duration is :

TR = Window Size/SR

The longer the window, the less "images" we get of the signal evolution in time.

Example

With the same 44100 sampling rate and **1024 points FFT , we get**

T = 1024/44100= 0.023

The spectrum is equally split into images representing a **23 ms. duration**.

If we choose a **4096 FFT**, we get

FR = 4096/44100 = 0,093

The spectrum is equally split into images representing a **93 ms. duration**. The frequency resolution is less precise.

Window Size and Frequency Resolution of the Analysis

This frequency resolution is determined by the number of "bins" in the analysis window. The number of bins actually is the FFT size, another parameters that will be discussed more specifically in the next section. For the time being, you just need to focus on the notion of "bins".

Bins

The number of samples of a window is divided into a number of "horizontal" strips, or **bins.** The number of bins determins the frequency resolution of the analysis, that is, how accurate the analysis can be in terms of frequency detection.

- **N (Bins) = Window Size/2**
- This **number of bins** must be a **power of 2** starting from 512: 512, 1024, 2048, 4096...

Note that the window size is generally also defined as a power of two, but this is not compulsary. This means that the number of bins is generally equal to the window size.

For instance, a 1024 samples window has 512 bins.

Window Size and Frequency Resolution

The **frequency resolution (FR) is the frequency band of a bin.** Remember: for a given sample rate, we have a corresponding frequency range **Fmax**, or NyQUist frequency.

We have two ways to know the frequency resolution :

- **Fmax** is split into a number of bins.
- The sample rate is split into the number of samples in a window.

When the number of bins in a window is equal to the window size, the frequency band of a bin is the same as the frequency band of the window.

FR = Fmax/N(Bins) = SR/Window Size

The more bins, the more slices of frequency range we get, and the more precise these slices are.

Calculating the Frequency Resolution (FR)

The **frequency resolution (FR) is the frequency band of a bin.**

For a given sample rate, the corresponding frequency range of the representation is split into a number of bins.`

FR = Fmax/N(Bins)

This also means that the sample rate is split into the number of samples in a window:

FR = SR/Window Size

The more bins, the more slices of frequency range we get, and the more precise these slices are.

Example

Let's take a 44100 sampling rate. SR=44100 Hz, F(max) = 22050 Hz.

With a **1024 window size (512 bins), we get** .

FR = 44100/1024 = 43.066

FR = 22050/512 = 43.066

The spectrum is equally split into 512 bins of **43.066 Hz** width.

If we choose a **4096 window size with 2048 bins**, we get

FR = 44100/4096 = 10, 76

FR = 22050/2048 = 10,76

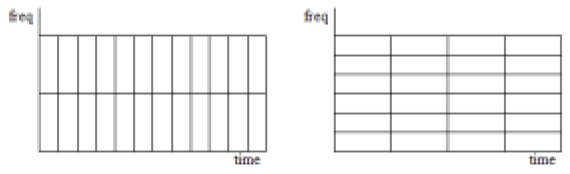
The spectrum is equally split into 2048 bins of **10.76 Hz** width. The frequency resolution is more precise.

Maximum Frequency Resolution

The number of bins in the window shouldn't be superior to 16 384 in order to display the sonogram, which corresponds to a 1.35 Hz frequency resolution – which is very high.

Choosing the Right Window Size - Time and Frequency Resolution of the FFT

The analysis window has a **fixed resolution**, which determines whether there is either a good frequency resolution – frequency components close together can be separated – or good time resolution – the time at which frequencies change – A wide window gives better frequency resolution but poor time resolution. A narrower window gives good time resolution but poor frequency resolution. These are called narrowband and wideband transforms, respectively. The size of the FFT can improve the frequency definition of the analysis.



Comparison of two window resolutions. The left schema shows a better time resolution, with an important succession of windows in time. The right one shows a better frequency resolution, with an important number of bins in the same window.

Sounds Characteristics

All sounds don't have the same characteristics, and these characteristics can change in time, or not. Selecting an FFT size involves making a compromise in termes of time and frequency accuracy. The more accurate the analysis is in one domain, the less accurate it will be in the other. The user most often make a compromise...

Temporal Variations

Variations in a stable sound occur every 2000 to 4000 samples, that is, 44 to 88 ms.

Variations in a rhythmic sound occure every 50 to 1000 samples, that is, 11,3 to 22,6 ms.

- If we adapt the window size to the frequency of a 100 Hz sound – G_2 –, and take a 2048 samples and 50 ms analysis window, we can easily analyse a stable sound, but not a fluctuating sound.
- If we adapt the window size to the frequency of a 440 Hz sound – A_3 –, we have a 512 samples and 11 ms analysis window, which is more appropriate for a fluctuating sound.
- Nevertheless, with a 2048 window size, our frequency resolution is equal to 44100/2048, that is 21,5 Hz, which is quite precise. With a 512 window size, we get a frequency resolution of 86 Hz, which is poor.

Frequency Variations

If we want to analyse a sound with a low and/or fluctuating pitch, we should take an important window size.

In the case of a low pitch, a C_1 for instance, we have a 32 Hz frequency with a 31 ms period. We would need a 8192 samples window size.

Frequency Resolution Linearity and Human Ear

The FFT size is **linear**, but the response of the human ear to frequencies is **logarithmic**.

For instance, with a 50 Hz frequency resolution, bins go from 0 to 50 Hz, 50 to 100Hz, 100 to 150 Hz, etc. If we take the frequencies of the octaves from G_1 to G_6, we get : 100, 200, 400, 800, 1600 Hz...

In a low frequency range, 50 Hz is quite a wide interval. From a G_1, a **fifth**. But from a G_6, 50 Hz represent a **semitone**.

The same FFT has very fine high frequency pitch resolution, but very poor low-frequency resolution.

Appendix C has been copied directly from:

AudioSculpt 3.0 User Manual by IRCAM

http://support.ircam.fr/docs/AudioSculpt/3.0/co/AudioSculptguideWeb.html

Appendix D – Windowing

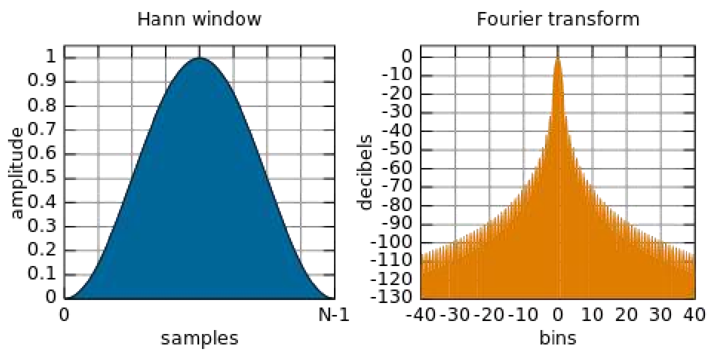
Many modules in FScape rely on windowing to define their operational zone. Some modules allow the type of window to be defined. Consult the following descriptions and images to infer results. But like all FScape functions experimentation with subtle changes yields knowledge.

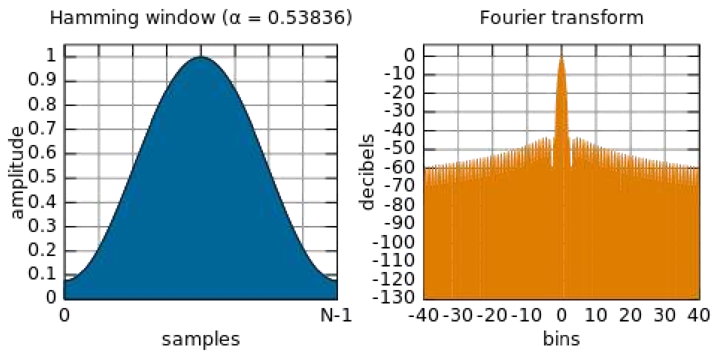
Windowing works best on a theoretical level where data falls perfectly in the center of each window. With real world applications input material rarely behaves perfectly and there can exist a sort of “blurring” between windows

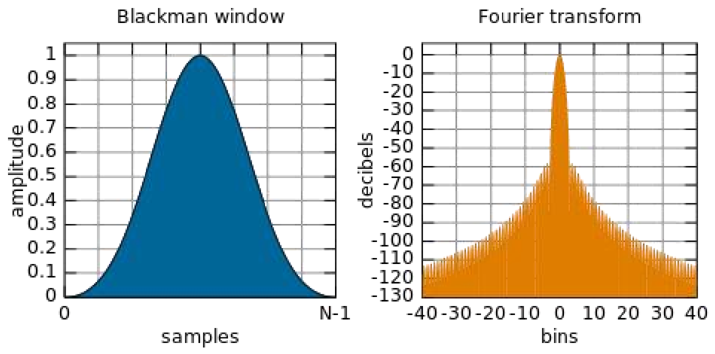
The situation can be improved by letting the segments overlap. For windows that are relatively wide in the time domain (such as Hanning), 50% is a commonly used value for the overlap. For narrower window functions (in particular at-top windows), a higher overlap (up to 84 %) may be appropriate.

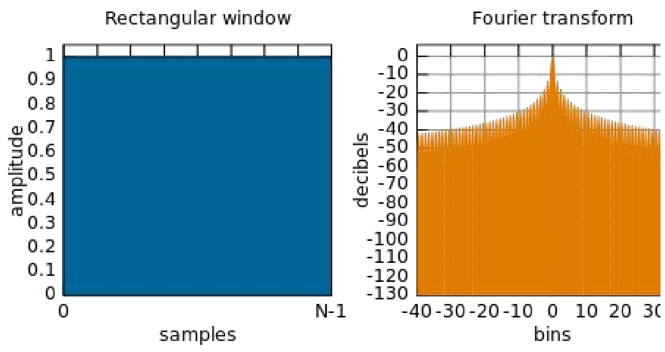
In some FScape modules overlap is adjustable in terms of multiples 2x, 4x, 8x etc.

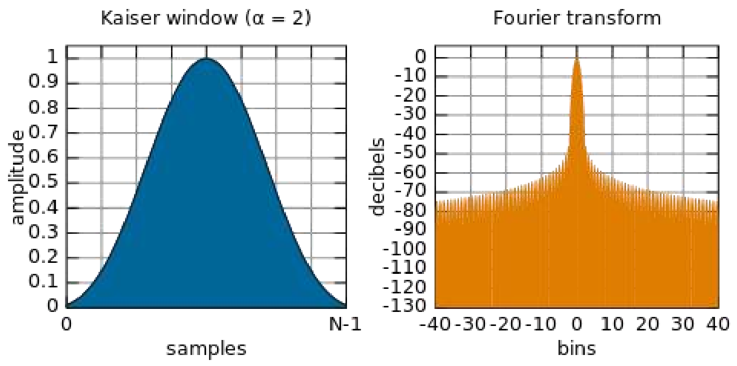
Hann and Hamming are good for stable low frequency material. Blackman is good for unstable noisy material. (IRCAM)

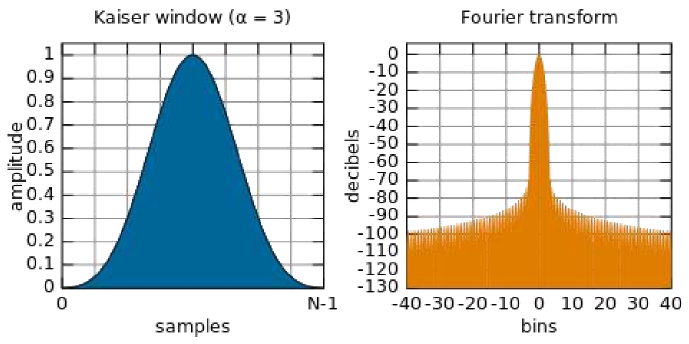


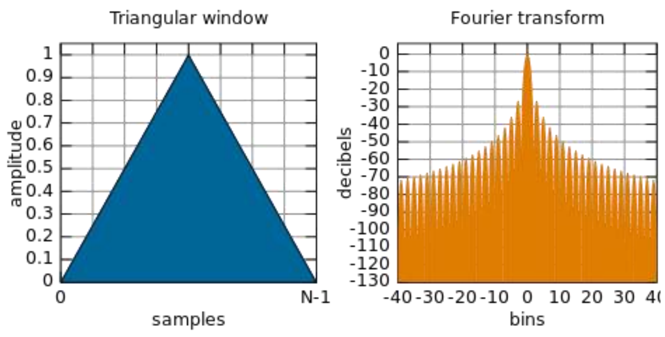












Appendix E – Useful Formulas

Time and time again I find myself using the following formulas for various reasons. These plus a calculator can help you make informed decisions when experimenting with FScape.

Converting between Time and Frequency(Hz) – This is especially useful when calculating pitch for single cycle waveforms.

1/t = Hz where t = seconds

1/0.0085034013 = 117.60Hz

n/R = t where n = time in samples and t is time in seconds and R = sample rate

375/44100 = 0.0085034013

1/f = t where t = time in seconds f = Hz

1/117.60 = 0.0085034013

t*R = n time in seconds multiplied by Sample Rate gives time in samples

0.0085034013*44100 = 375 samples

References – Appendix D

Heinzel, G., R�udiger, A., & R. Schilling. (2002). Spectrum and spectral density estimation by the Discrete Fourier transform (DFT), including a comprehensive list of window functions... *Max-Planck-Institut f�ur Gravitationsphysik*, 1 - 84.

IRCAM. (n.d.). *AudioSculpt 3.0 User Manual.* Retrieved from IRCAM Support: http://support.ircam.fr/docs/AudioSculpt/3.0/co/Introduction%20to%20Signal%20Analysis.html

Wikipedia. (2015, March 30). *Window function*. Retrieved from Wikipedia: http://en.wikipedia.org/wiki/Window_function