RIBS

manual

v 1.2.3

This document is a work in progress.
If you find any inaccuracies
or think that some topic is insufficiently covered,
feel free to contact us at hvoyaaudio@ya.ru

Thanks to all who patiently ask questions
and provide constructive feedback.

facebook.com/hvoyaaudio
vk.com/hvoyaaudio

CONTENTS

GENERAL INFO

Ribs is a virtual instrument / fx that uses granular synthesis (GS) as its main sound production method.

This manual describes Ribs v 1.2.3. To check which version you have press the “i” button on the GUI to the right of the logo or see changelog.txt that comes with the distribution.

Ribs is distributed as x86 and x64 VST2 and standalone builds for Windows and as universal builds of VST2 and AU plugins and a standalone app for OSX. It should work on all Windows systems starting from 7 and all OSX starting from 10.7. To use it just copy the corresponding files into your preferred plugin directory.

If you find this software useful and want to support its further development, you can do so at Patreon.com/EugeneYakshin or Paypal.me/EugeneYakshin. Your support is highly appreciated.

This software is written with love and care in the minds of its creators, however if as a result of using it your computer starts exterminating everything in its reach, you cannot blame the authors. See EULA.txt for more details.

If you encounter any bugs, please send an e-mail with a description to ribsey@ya.ru.

VST is a trademark and software of Steinberg Media Technologies GmbH.

QUICK START & FX MODE

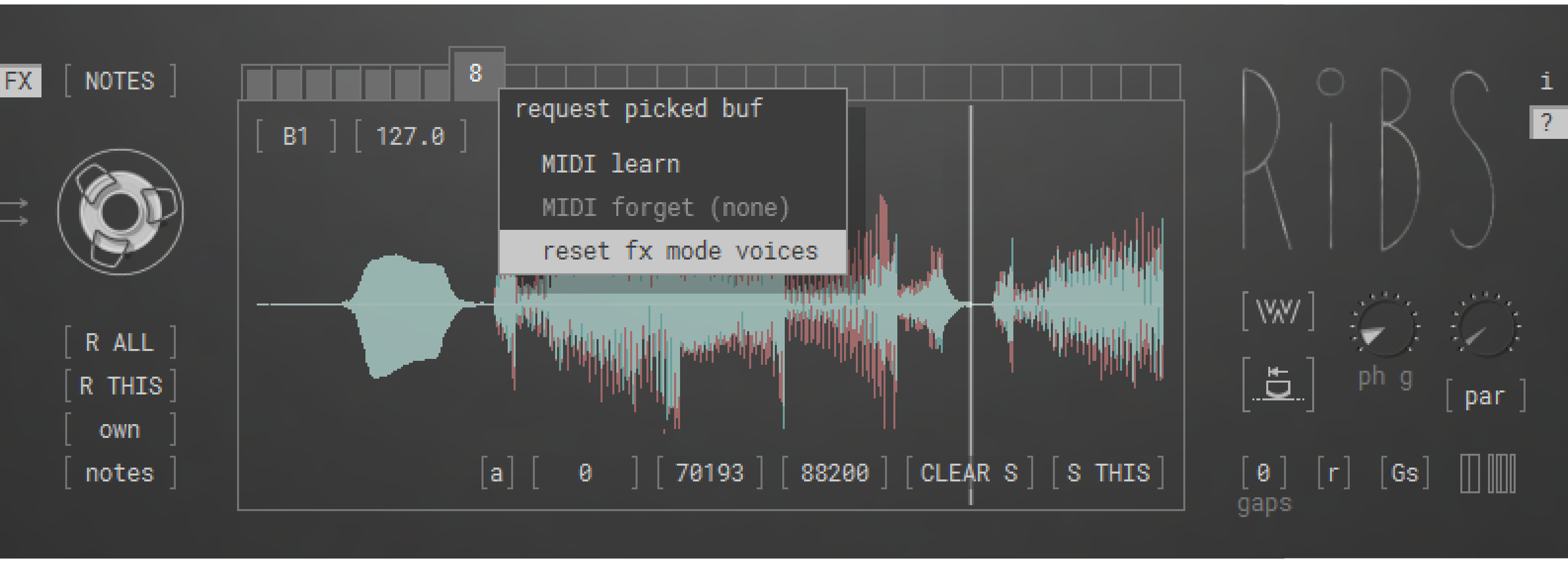
Ribs has 32 buffers that can store incoming audio and can be used as separate voices triggered by MIDI messages.

But the easiest way to use Ribs is to enable the **FX mode**. To do that press the FX button on the top-left of the GUI. In this mode Ribs behaves as an fx. By default the first tab on the top will be highlighted with light gray, indicating that this buffer will be used as soon as your host starts playing. Audio for this buffer will be grabbed from the output of the track that Ribs is on.

You can enable and disable any voice by clicking on a relative tab top. Technically, **request picked buf** param is responsible for toggling a voice’s state. By right-clicking the tab tops or the FX button you can find “reset fx mode voices” item that will release all active voices.

In FX mode the **voice note** and **voice volume** parameters are visible above the waveform display on the left. You can set them for each voice independently. By right-clicking the voice note you can find “set all to current” item that will set all voices’ notes to the current one.

In FX mode
MIDI note on and note off messages are ignored,
active voices parameter, **use buf for <note>** flags and **MIDI panic** are ignored,
poly/mono logic set by the corresponding parameter does apply.



⚠ For hosts that don’t support routing of both audio signals and MIDI messages to the same track FX mode is the only way to use Ribs.

By default when you hover the mouse over the controls **tooltips** with parameter name, short description and optional control modifiers will be shown. You can always toggle it by clicking the “?” button to the right of Ribs logo on the top-right of the UI.

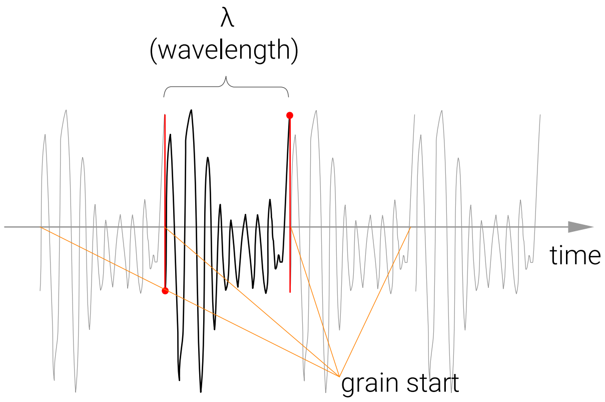
The “i” button right above it will show the panel with some technical data and support links.

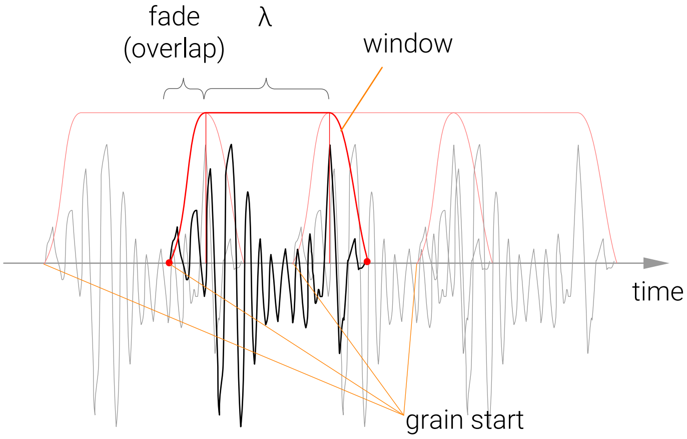
Also, see the presets folder that comes with the distribution. It contains several simple .fxp preset files as well as description.txt that can help you find your way around faster.

EXTREMELY SHORT GRANULAR SYNTHESIS OVERVIEW

The main idea of GS is cutting a sound into small pieces that are called *grains*, then overlaying and rearranging them to produce a new sound. So Ribs does not exactly generate the sound from scratch, but uses the source audio to build from it. If you cut a sound piece of a certain length (usually denoted by the Greek letter λ, lambda) and repeat it, you can hear a pitch that corresponds to the same *wavelength*. Thus, if you repeat a grain with the length of 1/261.6 seconds, you can hear C of the first octave, because it's wavelength is 261.6 Hz.

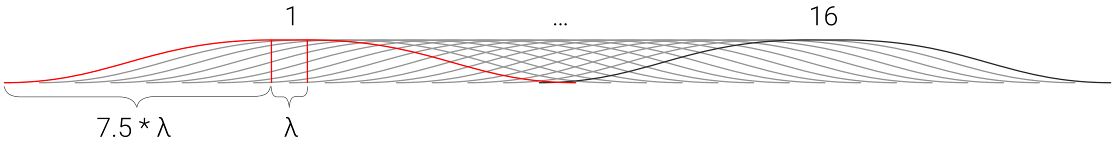
But if you simply repeat a grain, most probably it will sound harsh, since its end and start don't have the same value. Such jumps in audio signal create unpleasant clicks. This problem is solved by multiplying a grain by a *window function*. Window functions that are used in GS usually start at 0.0, smoothly go up to 1.0 and then fade back to zero. In general you need a longer grain to compensate for these fades, but if you keep the actual time between the grain starts unchanged the pitch will also stay the same. Consequently, the grains should overlap exactly by the length of the fades. (See **window type** and **grain overlap** params)





In Ribs wavelength is referred to as *base length* (sometimes also lambda), and *overlap* is called exactly the same.

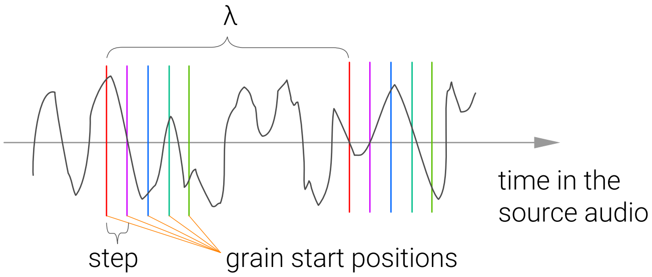
The number of simultaneously sounding grains is two times the overlap plus one, e.g. when overlap equals 7.5 there are 16 grains being played at the same time, though several of them are very quiet because they are in stages too close to window start or end.



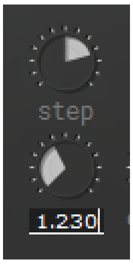
A grain is composed of the contents read from the source audio. This reading process starts from the *grain start position*. The reading position then moves through the source, building the grain sample by sample. Reading position is called *playhead* in Ribs.

If you repeat exactly the same grain without any changes, it sounds mechanical. But if you slowly shift the grain start position through the source audio, the sound becomes more alive, and the perception of pitch remains. If you shift it too fast, the pitch will fall apart. This shift parameter is referred to as **step**.

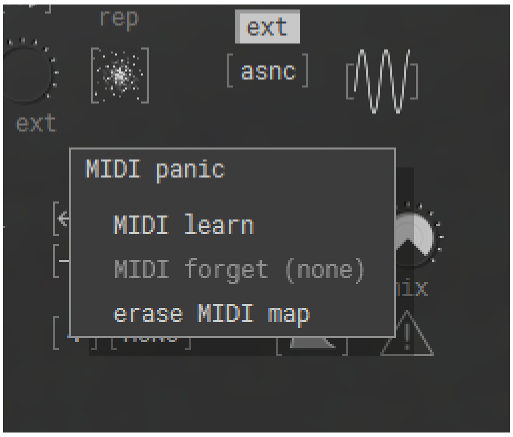
That's pretty much it.



GUI & MIDI CONTROL

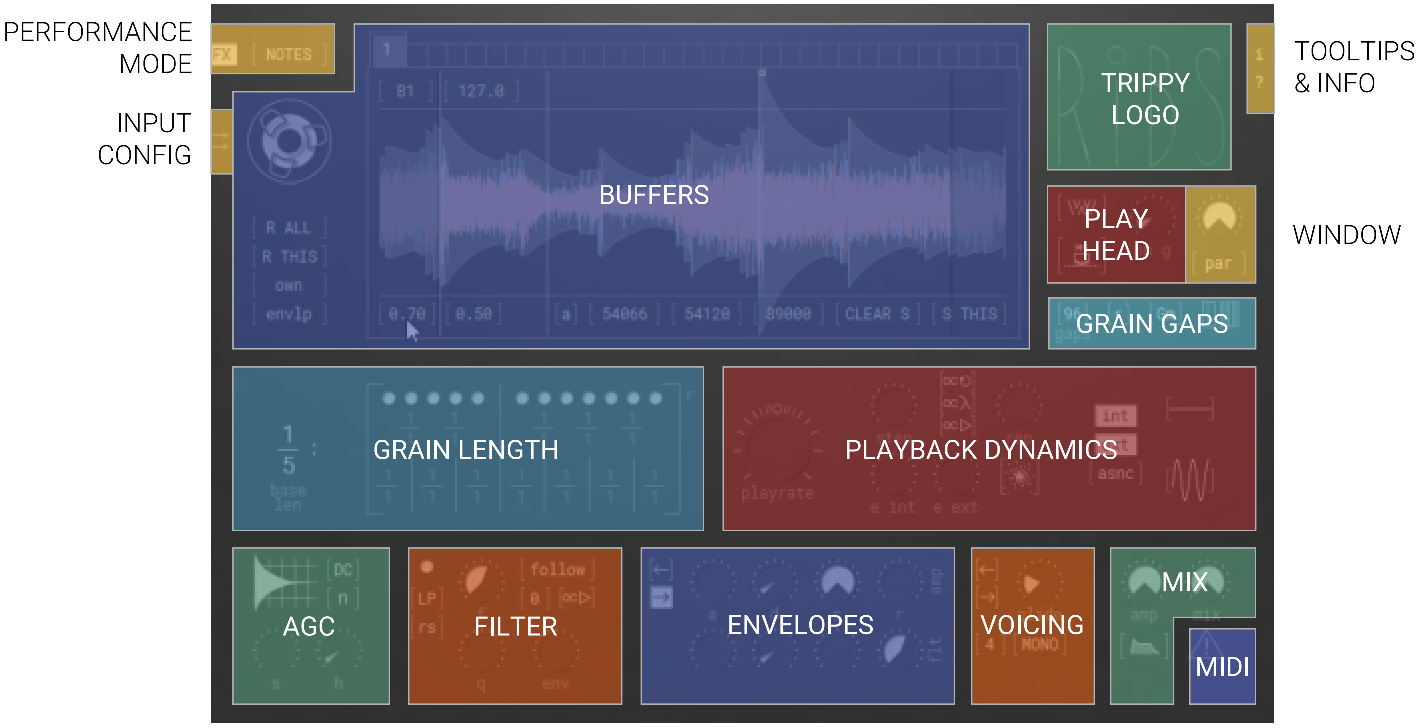


Most of the UI elements are controlled by mouse dragging or mouse wheel. When you hold down the Ctrl key (Cmd on Mac) the value changes slower. There are several exceptions that will be described in corresponding sections. When you change some controls, additional small rectangle with param value appears beneath a control. You can click this rectangle and type the value that you want.



Most of controls have a **menu** available by the right click. The 1st item is the exact parameter name, then go **MIDI learn** and **MIDI forget** items. Some controls have additional items in the menu. When this is the case, a tooltip reflects that. For example, the menu for **MIDI panic** param has the item that allows to **erase MIDI map**.

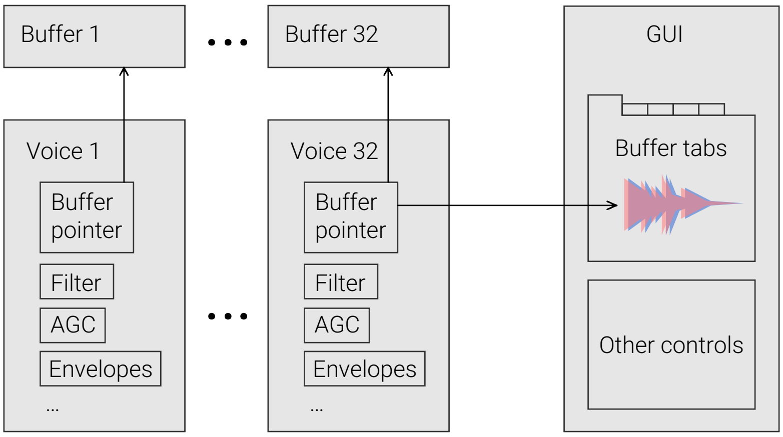
Here's an approximate layout of GUI sections based on the functions of the controls:



If GUI texts on your machine look different from what you see in this manual, install Roboto Mono font that comes with the distribution archive. Roboto font family is designed by Christian Robertson, it's beautiful and is free to use.

GENERAL STRUCTURE

Ribs is a polyphonic synth. Its structure is a bit more flexible than the usual one across the granular tools, so in some use scenarios it may seem a bit confusing. Here's its rough structure that will help you gain some intuition:



It has 32 *voices*. Each voice has its own filter, envelopes, AGS scheme and additional data such as playhead related parameters, selection range, note number and so on. When you press a MIDI note, the synth decides which voice to pick to play the note.

When a voice plays a note, it uses audio from a *buffer*. There are 32 independent buffers, and by default each buffer is assigned to a corresponding voice. The 1st buffer to the 1st voice, the 2nd to the 2nd voice and so on. But you can also make all the voices use only one buffer (see **own/common buf** param).

GUI has 32 buffer *tabs*. Each tab represents the contents of a buffer, that a corresponding voice uses. So by default n-th tab shows the contents of the n-th buffer, because n-th voice uses that buffer. While in common buffer mode, all voices use the same buffer (for example, the 3rd), so all the tabs display the same 3rd waveform.

ⓘ For shortness the terms "tab", "voice" and "buffer" will be used interchangeably when appropriate from now on.

All GUI controls affect voices globally, except for several, which will be specifically described. That is when a parameter is changed, the change is applied to all the voices.

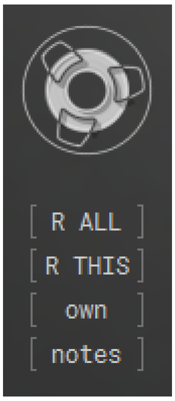
FILLING THE BUFFERS

By default when **FX mode** param is off buffer filling is triggered by incoming MIDI notes. In general if a buffer is empty it starts filling as soon as it starts being used by a voice. Further refilling is controlled by the **refill mode**, **refill all** and **refill this** params.

Each buffer records and plays back in stereo, and has maximum length of 480000 stereo samples (10 sec at 48 kHz sample rate). For shorter lengths you can adjust the **buf size** param.

Buffer contents are not saved in plugin states or presets. As of version 1.2.3 exporting to and importing from external files is not supported.

Buffers are filling only when the voices that use them are active and host transport is running. I. e. if a host has stopped or paused no filling is performed even if you still hear voices' release tails. If a refill request is sent in any of these cases "REFILL PENDING" will be written in the corresponding buffer tab.



buf size

refill all

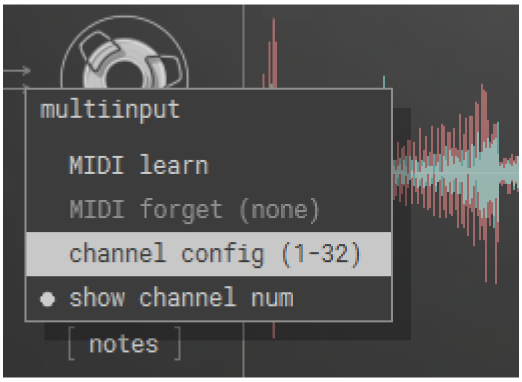
refill this

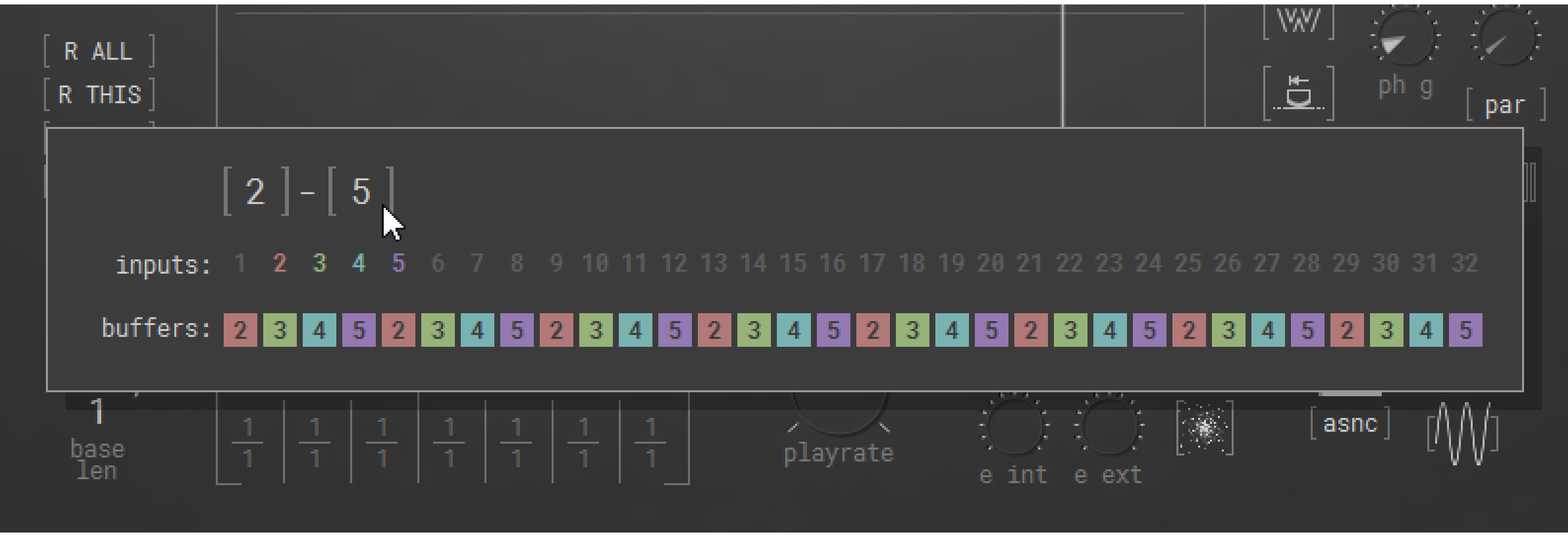
refill mode

Refill all refills all buffers. **Refill this** refills only the **picked buffer** (see **own/common buf** description for more details).

By default all buffers record audio from the first stereo input of Ribs. You can assign each buffer its own input by clicking the **multiinput** control on the top left (the two arrows near the reel).

Some hosts put limitations on the range of available input channels. If it's your case (the buffers are filling with silence, for example) right click the multiinput control and in the menu click the **channel config** item. The config window will pop up where you can set the input range. There's a color coding and numbering that clarify how the inputs are distributed. You can also enable **show channel num** to display an input number above the waveform in the buffer tabs.





[2]-[5]

inputs:	1	2	3	4	5	6	7	8	9	10	11	12	13	14	15	16	17	18	19	20	21	22	23	24	25	26	27	28	29	30	31	32
buffers:	2	3	4	5	2	3	4	5	2	3	4	5	2	3	4	5	2	3	4	5	2	3	4	5	2	3	4	5	2	3	4	5

REFILL MODES

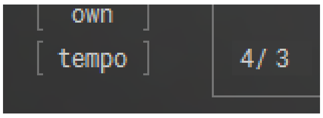
There are several refill modes:

notes refills a buffer when the associated voice starts playing a new note.

!glide does the same except for the case when a voice is gliding in monophonic mode (see **poly/mono** param) or when the portamento MIDI CC is on.

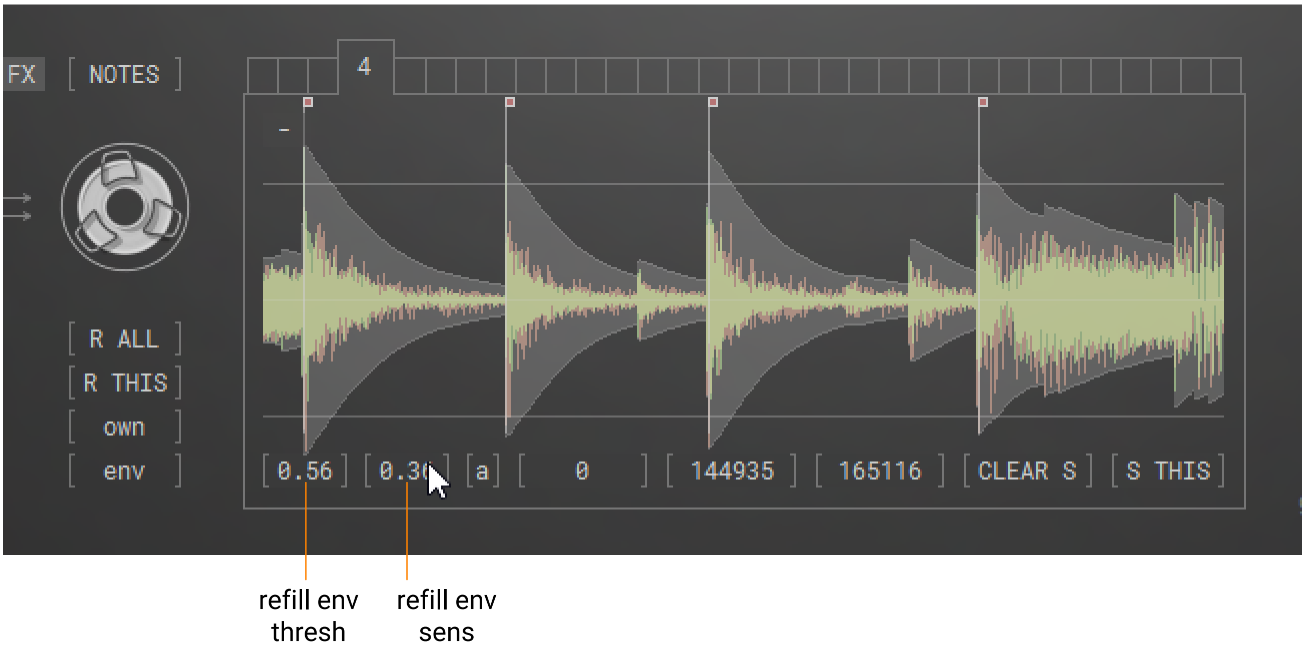
hold keeps the contents until you explicitly request a refill.

tempo refills buffers every so often. When you switch to this mode, **refill nom** and **refill denum** controls appear to the right of refill mode control. You can define the refill frequency measured in host beats by these controls.

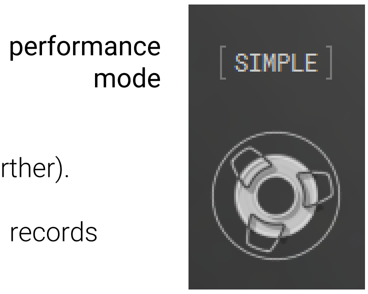


run refills a buffer continuously. As soon as the filling position reaches the end of the buffer it starts refilling from the beginning.

env refills a buffer on high input signal envelope level. Parameters **refill env thresh** and **refill env sens** become visible on the bottom left of the waveform area. You can adjust the threshold and sensitivity with which the refilling envelope reacts to the input changes. By hovering the mouse over these controls you can see a transparent curved shape that shows the approximate behavior of the envelope based on existing buffer contents. Possible refill events are marked with vertical lines with marks on their tops.



PERFORMANCE MODES



There are 3 performance modes:

NOTES grain lengths correspond to the played MIDI notes.

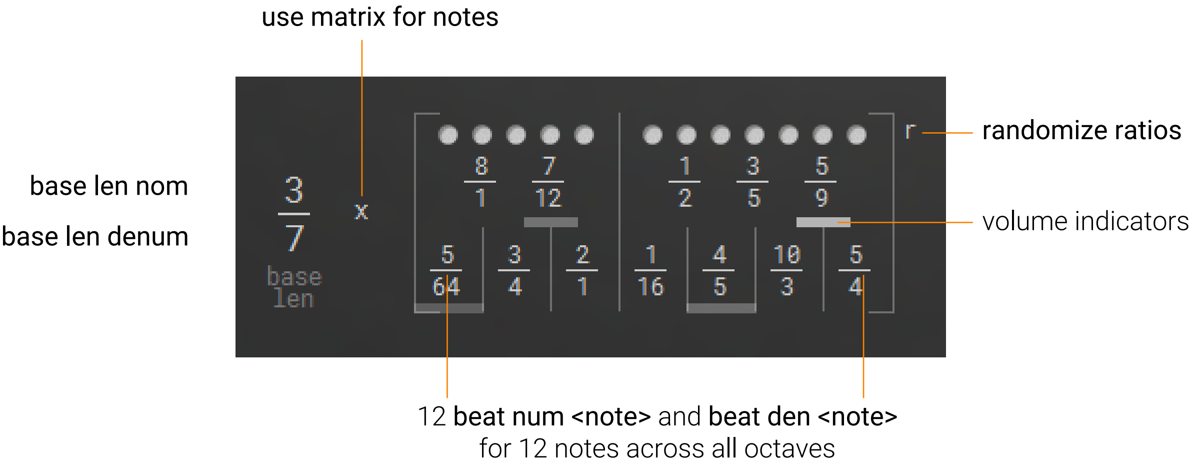
BEAT grain lengths correspond to the host tempo and **beat ratios** (described further).

SIMPLE simple buffer playback. Useful if you want to scratch the buffers as vinyl records or use Ribs as a sampler.

GRAIN LENGTH, WINDOWS & GAPS

GRAIN LENGTHS

Ribs allows for some flexibility when it comes to grain lengths. Instead of one number, it uses separate numerator and denominator params. In NOTES mode it allows for interesting harmonic experiments and simple fast octave shifts, and in BEAT mode you get to build lots of possible polymetric structures.



12 **beat num <note>** and **beat den <note>** for 12 notes across all octaves

Base len nom and **base len denum** form a ratio by which grain lengths are multiplied in all granular performance modes.

Beat num <note> and **beat den <note>** also change lengths, but by default only in BEAT mode. If **use matrix for notes** param is on, they affect grain lengths in NOTES mode too. When beat ratios affect grain lengths, they do so across all octaves. For example, when **beat num E** is 1 and **beat den E** is 2, all Es ' grain lengths will be two times shorter.

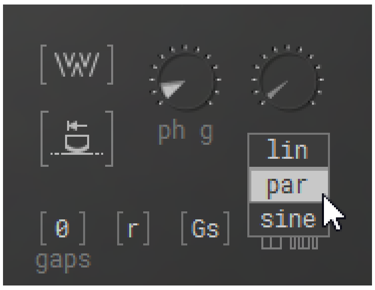
If you hold down Shift and use mouse wheel, you can quickly change a value of a numerator or a denominator to be two times bigger or smaller. Holding down Ctrl (or Cmd on Mac) will make values change slower.

Beat ratios are organized graphically in a structure that resembles one octave keyboard, but without complete key borders and colors. Transparent rectangle on the bottom of each key shows the approximate sum of volumes of all voices that play this note, if there are any.

To set up the key ratios easier, use **randomize ratios** param, or, to the contrary, use **align others** menu item to align all numerators or all denominators to the clicked one.

ⓘ Note that grain length ratios affect the filter cut-off when **flt follows note** param is on. **Base len nom** and **base len denum** will do that always, while **beat num**s and **beat den**s for notes will do so only when **use matrix for NOTES** param is on.

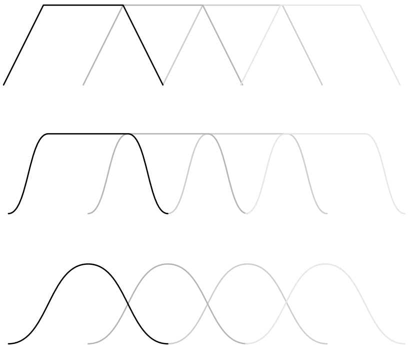
WINDOWS



Grain overlap is measured in base lengths. Naturally, the more overlap there is, the more high frequency details get smoothed out. High frequency content is still there, but the details are not as localized in time as they originally were. However, with relatively big base lengths this effect disappears. With longer overlaps some audible band-reject filtering and flanger may also appear.

Longer overlaps imply higher CPU load. You can balance it out by lowering the amount of **active voices**.

There are 3 window functions:



overlap = 0.5

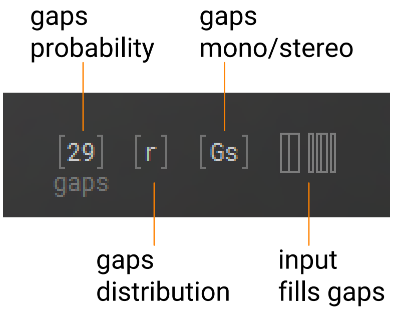
linear	the most harsh-sounding one
parabolic	fades are composited of pieces of a parabola. More soft-sounding.
sinusoid	different from the previous two in that it doesn't have a flat part. With zero and short overlaps it introduces noticeable amplitude modulation.

GAPS

It may sound interesting when you skip grains. **Gaps probability** controls how often it happens, **gaps mono/stereo** allows to skip left and right contents of a grain independently, and **gaps distribution** toggles between random and evenly distributed gaps

Input fills gaps does what it says. When a gap is filled with input it will be processed by the filter and AGC just as a usual grain would.

One of possible use cases would be having average overlaps and moderate amount of stereo gaps - it makes the panorama of a sound wider and the structure more interesting.



PICKED BUFFER

Picked buffer is one of the key params in Ribs. It is used to modify several params on a per-buffer basis. When you modify them, the synth looks at the value of picked buffer to choose which buffer gets the modifications (also, see Automation section). These dependable params are:

own/common buf
use for <note> flags
request buf

selection start
selection end
clear selection

refill this
refill env sens
refill env thresh

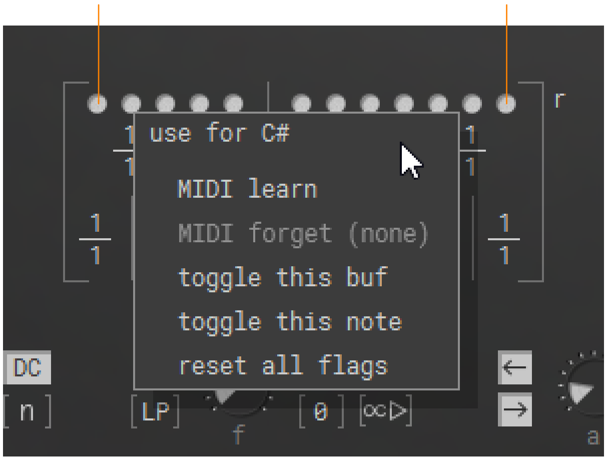
playhead position
aiming playhead

voice note
voice volume

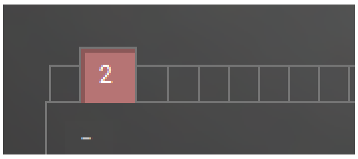
Using these params together with **picked buffer** allows to set their unique values for every buffer without bloating the parameter list. Consequently, their automation curves in your DAW will reflect the states for the picked buffer.

To change the value of **picked buffer,** you have to left-click on a desired tab. The number on the picked tab displays the value of the parameter. If you just hover the mouse over the tabs without clicking, you can see the waveforms of other buffers without actually picking, as well as the states of **selection start**, **selection end**, **playhead position** and the **use for <note>** flags for the buffer that you are looking up.
Detailed description of operations with waveforms (selecting a region, aiming playhead, scratching, zooming, etc.) is described in the "Waveform Actions" and "Playhead Motion" sections.

USING CERTAIN BUFFERS

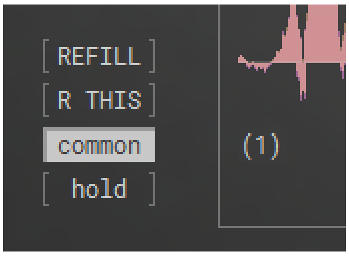

12 **use for <note>** flags for 12 notes across all octaves

Use for <note> flags are among those parameters which values are unique to every buffer. These 12 controls reflect the states of use flags for the **picked buffer**. They define whether the voice that is associated with the picked buffer will be selected by the synth to play the corresponding note. It is useful when you set Ribs up to work as a sampler. This way you can specify a set of buffers for the synth to choose from when picking a voice to play a note. There are additional group operations available in the menus of these params to make setting up easier.
Note that if for any reason there are no other voices to play the note (e.g. in monophonic mode), the flags will be ignored.
In FX mode the flags are ignored.

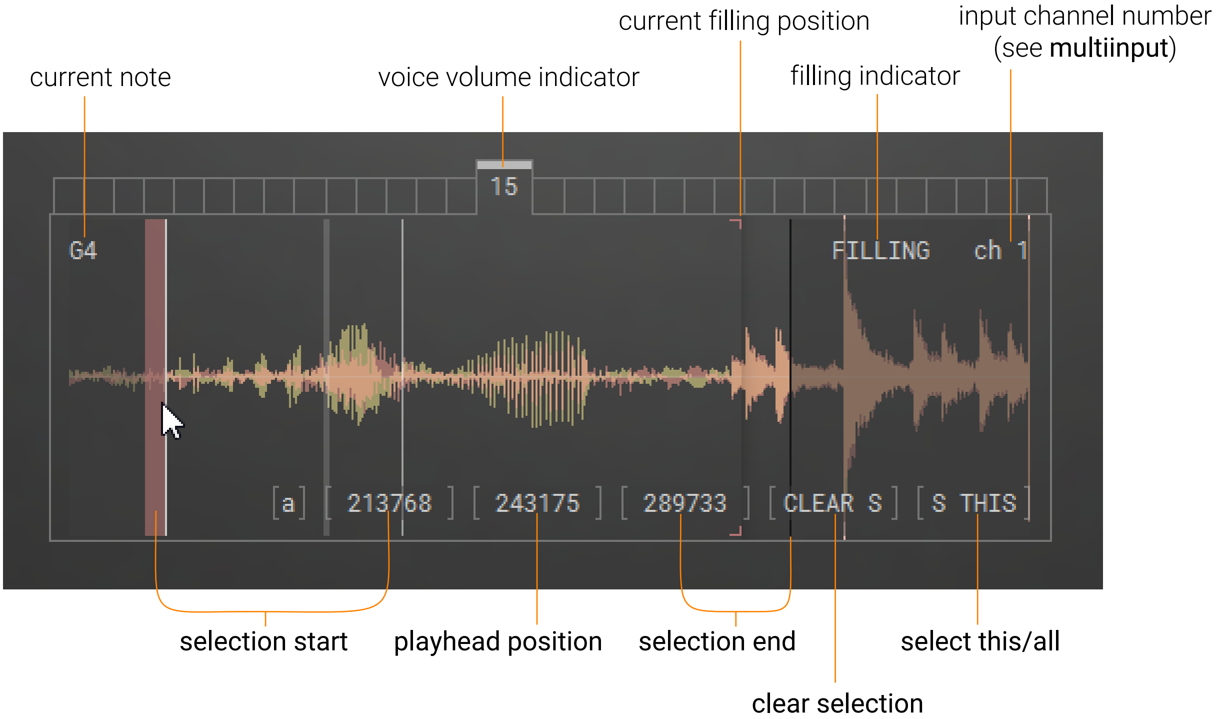


The flags are also ignored if you request a certain buffer to be used to play the next note. You can do that with **request picked buf** param. To request a buffer just click on the top of the picked buffer tab. It then gets highlighted. When you play the next note, the flag is automatically switched off. If you pick another tab, the request will be transferred to it. If you want to cancel the request before the note is played, click it once more.
You can use this param right before switching to or during the monophonic mode or portamento. The currently used voice will then be released as soon as the new note is sent to Ribs.
When **FX mode** is on, **request picked buf** is used to trigger the picked voice state. Menu for this parameter is available by right click on any tab except the picked one.

Own/common buf param is used to set all voices to use the picked buffer as their audio source. When on, all buffer tabs display the same waveform, and the id of the common buffer is shown to the right of the control. However, all per-voice params will keep their unique states (selections, playheads positions, use flags, waveform zooms, etc). All other buffers will keep their contents, and refill commands will be ignored by them. **Refill this** will be ignored only when the common buffer id and the picked tab id are different. For example, if you picked 1st buffer as common and then switched to the 5th tab, R THIS will be ignored, When you switch back to the 1st tab, R THIS will do its job. Picking another tab after enabling common buffer mode will not switch voices to this new buffer.



WAVEFORM ACTIONS



Buffer tabs area has several controls in it and contains a lot of information. Most of it is explained in this section, but parts that are directly related to playhead are described in the “Playhead Motion” section. With tooltips enabled hovering the mouse over the waveform displays short description of controls. **Picked buffer** and **request picked buf** were already described earlier. You can look up another buffer contents, selection range, playhead and use buf flags states without picking a buffer by simply hovering the mouse over the tabs.

Just as on beat ratio keyboard, transparent white rectangles on the tops of the tabs show approximate volume, but here it is on a per-voice basis.

Each tab has its own unchangeable pair of slightly transparent colors to display left and right channels of the corresponding buffer contents. When **own/common buf** param is on, all waveforms display the contents of the common buffer, but each tab keeps its own colors and zoom levels. When the contents are too loud, the overdriven audio peaks will be marked with transparent white.

EXAMINING WAVEFORMS

Waveform display allows for detailed examination of buffer contents. By default a waveform displays the full length of a buffer. You can use mouse wheel to **zoom** in and out on the time axis. When zoomed in, you can use Shift + mouse wheel to **scroll** the view to the sides. If the contents are too quiet, use Ctrl (Cmd on Mac) + wheel to zoom in and out on the amplitude axis. Zooms do not depend on anything, however changing the **buffer size** resets horizontal zoom. As of version 1.2.3 zooms are not saved in presets and in the plugin states.

SELECTING REGIONS

You can temporarily restrict the working area inside a buffer with **selection start** and **selection end** params. This is done by right-dragging with the mouse over the waveform or by left-dragging and holding the Shift. The edge that is currently being modified will be highlighted. You can edit the edges separately when you hover the mouse over the needed one and drag it in the same way. If you want to transpose the existing selection, right-click or Shift + left-click the position on the waveform where you want to set the transposed selection start.

Minimal selection size is 4 samples. Selection is unique to every buffer (see **picked buffer**) and saved in presets and plugin states. **Clear selection** function is obvious. To manage selections for all buffers at the same time, switch the **select this/all** param on.

For hosts both start and end values are normalized (i.e. from 0.0 to 1.0, where 1.0 corresponds to the current value of **buffer size** param), but to make it easier on the GUI the values in the square brackets on the bottom of the waveform region represent edges’ positions measured in actual samples in a buffer. Since all mouse clicks directly on the waveform are used to perform actions, to access the menus of selection params just right-click their values in square brackets.

Selection can be controlled by MIDI CC. To make setting these params by MIDI more precise, the values of incoming CCs are mapped on the visible range. However, since in version 1.1.0 visible waveform ranges and zooms are not saved in presets, the reliable way to save selection automation is by recording automation curves for these params.

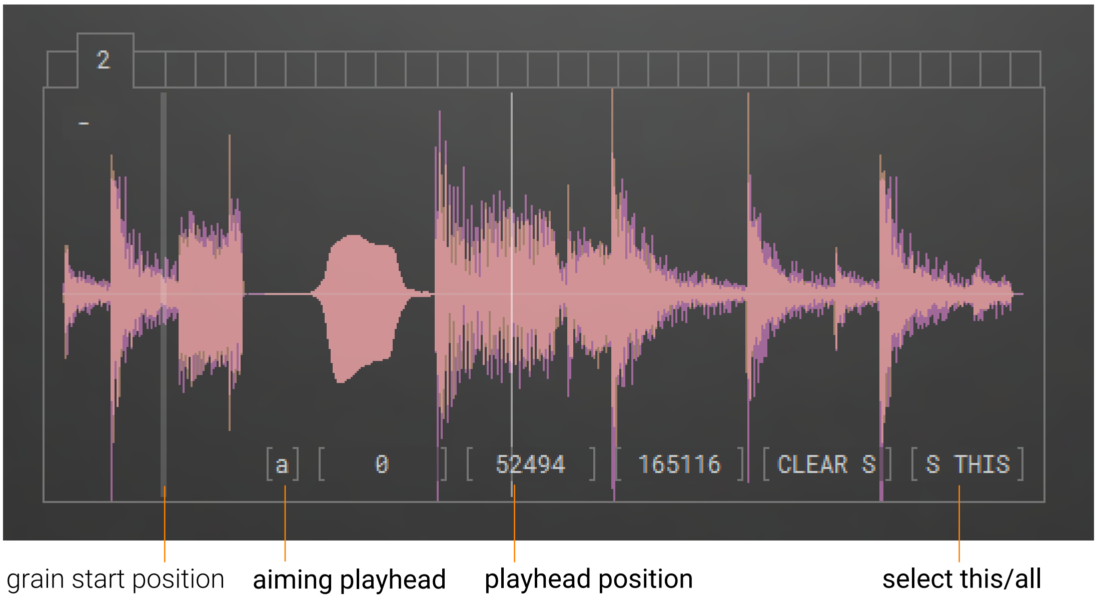
PLAYHEAD MOTION

Playhead related controls are plenty, but the logic is fairly simple here. There are params responsible for propelling the playhead, params that introduce different types of errors into the motion, params responsible for special cases behavior and params related to direct user control of playhead motion (*aiming* a playhed).

But first, there are two vertical lines on the waveform that indicate playhead position data for displayed buffer.

Thick transparent white line shows last calculated grain start position. This line is present only in granular performance modes.

Thin light gray line shows current playhead position. Just in case you're wondering: since with nonzero overlap there may be several grains playing simultaneously and each grain reads from a buffer independently. This line shows the playhead of the latest launched grain.

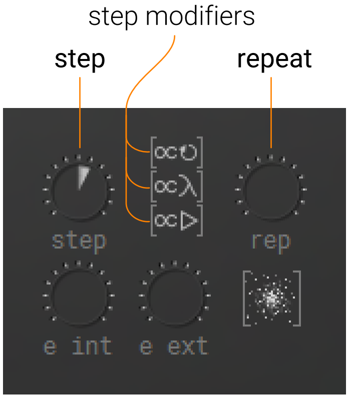


AUTOMATIC PROPELLING

Initially all voices' playheads start from the first sample of a buffer and move automatically forward in recorded source audio time, from left to right by default.

In SIMPLE mode the speed and direction of motions are controlled by the **playrate** param.

In granular modes there are two aspects of motion: propelling of a grain start position and propelling of the playheads that are used by the grains. The former is controlled with the **step** param, and the latter with the **playrate** param. Both controls allow to double or halve their values by using mouse wheel and Shift key.

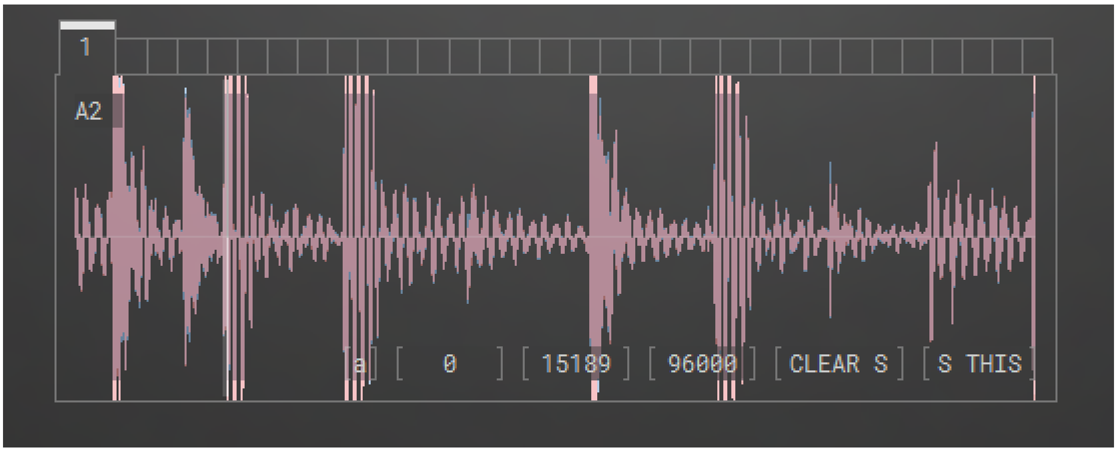


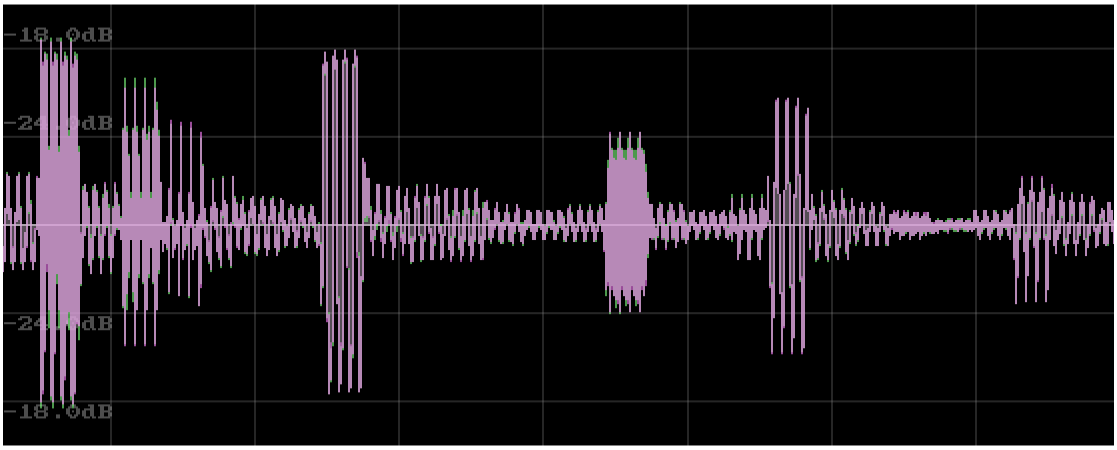
As explained in the Extremely Short Granular Synthesis Overview, **step** is the distance between the previous and the new grain start positions. It is measured in samples. To preserve the perception of a pitch this distance shouldn't be too long. Or, if it does, before moving the grain start position further, the current grain should be repeated several times. This is done by using **repeat** param.

If you want the sounds that you play to have temporal structures that resemble that of the related buffers to some extent, the following three step modifiers may be very useful (in many other cases they're useful too):

step *= repeat	multiplies step by the number of grain repeats,
step *= base len	multiplies step by the length of a grain,
step *= playrate	multiplies step by the playrate.

The effect of using these parameters will be intuitively clear if you compare the contents of a buffer and a snapshot of Ribs' output. Here Ribs plays a single note, step and playrate are set to 1.0, repeat equals 4, and step is proportional to the number of repeats and base len:

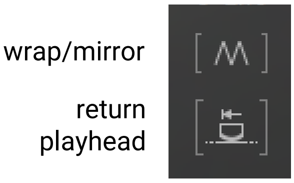




this snapshot is taken with Oscilloscope Meter by Cockos

By the way, the symbol "*=" is a programmers' shortcut for multiplying the value on the left by the value o the right, and "∝" is a mathematical symbol for proportionality.

SPECIAL CASES



Playheads and grain start positions are essentially just read positions in a buffer. Each buffer has a valid range inside which these positions should be kept. Initially this range is defined by the beginning and the end of the buffer. While the buffer is not completely filled the fill level puts tighter restrictions on the valid range. Moreover, you can specify the range by using **selection start** and **selection end** parameters.

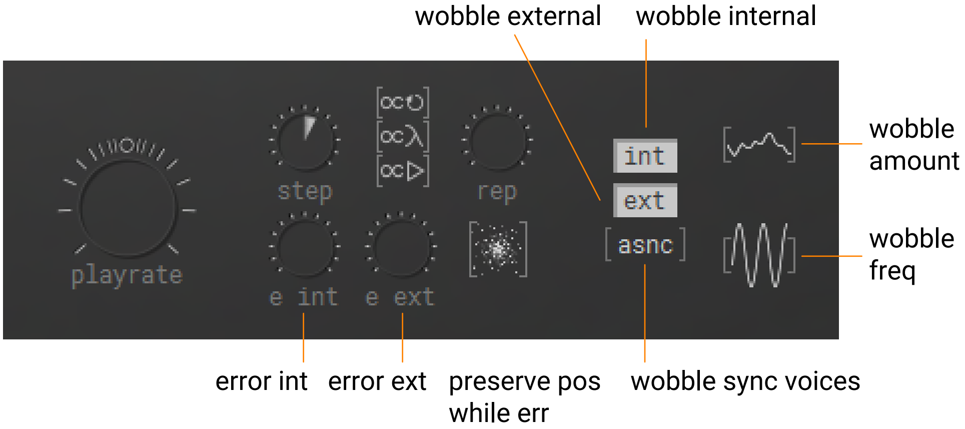
When a read position reaches either of the edges of the valid range, there are two options: the position can be wrapped around the range and keep the direction of movement, or be reflected against the edge, changing the direction. **Wrap/mirror** param defines this behavior.

Another special case is what happens when a new note is played by a voice. If **return playhead** param is on, read positions will be reset to the beginning of a valid range. Otherwise they stay intact.

INTRODUCING WOBBLE AND NOISE TO PLAYHEADS' TRAJECTORIES

Things appear to be dead when everything is exactly the same. People love all things analogue because in general real world devices always have some imperfections and noise as essential parts of their nature. Analogue filters and amplifiers have all sorts of nonlinearities, some analogue synths don't play perfectly correct frequencies, and so on.

Ribs has a number of parameters that can make sound more interesting by gently changing it here and there or on the contrary by turning it into an ocean of noise.



Wobble does it in a continuous manner. Wobbling is an independent process that happens on a per-voice basis. It can be applied to the **playrate** by enabling the **wobble internal** param, and to the base length of each new grain by enabling the **wobble external** param. **Wobble amount** and **wobble freq** functions are obvious. You can also synchronize the wobble across all voices with **wobble sync voices** param.

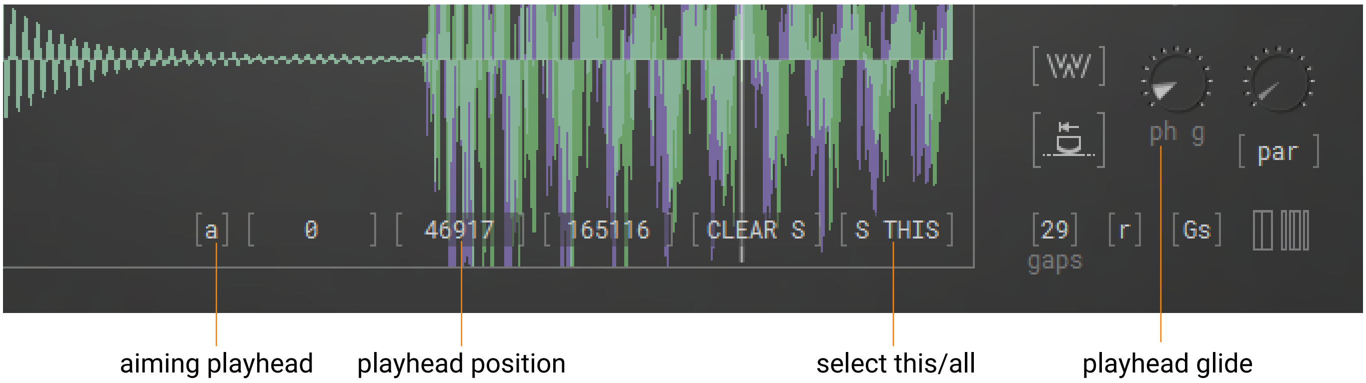
As a tip, high amplitude low frequency wobble resembles an old vinyl record, and low amplitude high frequency wobble sounds a bit like a magnetic tape.

It is possible to add some error to read positions. **Error int** will do that every sample to all the *playheads*, and **error ext** will do that to *grain start position* every new block of <**repeat** param value> grains. E.g. if **repeat** equals 5, the error will be introduced every 5th new grain. **Error ext** differs in NOTES and BEAT modes. In NOTES it's proportional to a grain length. In BEAT mode it's not, so it could be used to liven an otherwise mechanically repeating slice.

By default the errors are added to the correct position, so that the temporal dynamic structure of the audio will on average stay the same. But you can add the errors to the previous erroneous position by switching the **preserve pos while err** param off, and then, as you add bigger errors, there even will be no guarantee that the grain start position makes it to the edge of a buffer, forget the structure at all. Naturally, if an error is smaller than the speed of smooth motion, the motion will at least keep its direction.

Wobble external and **error ext** have no effect in SIMPLE performance mode.

AIMING PLAYHEAD AND SCRATCHING



You can *aim* playhead at a certain position by left-clicking on a place in a waveform. The playhead will reach that target in time controlled by the **playhead glide** param. Set it to zero if you want the position to be set instantaneously.

In SIMPLE mode you're controlling a *playhead*, whereas in granular modes you're controlling a *grain start position*. It is done to preserve the granular character of sound. Naturally, in SIMPLE mode gliding is accompanied by distinctive change of playrate. But in granular modes the playrate is internal to the grains, so it remains intact and glides sound more like skipping throught the audio CD if you have ever heard this sound.

Grain start position takes into account the **repeat** value during its glide to the target. For convenience both *playhead* and *grain start position* are referred to simply as *playhead* in this chapter.

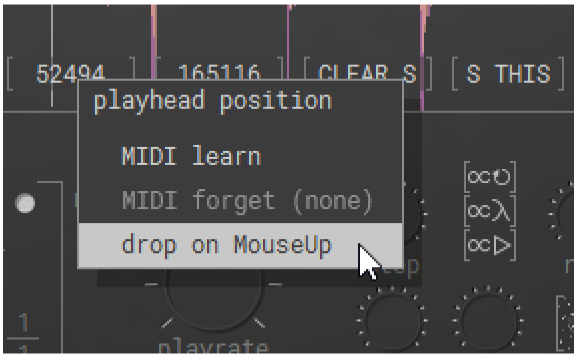
After the glide is over, the playhead will continue its automatic motion. If you want to keep the playhead at the target, don't release the mouse. This way you can scratch through the buffers like if it's a vinyl record. **Playhead glide** function will then be similar to the inertia parameter in the real world.

You can aim all the playheads and scratch all the buffers at the same time, using the **select this/all** param.

ⓘ During the glides all wobble and error parameters are applied correspondingly.
If the **flt freq *= playrate** param is on, the filter cutoff will be changed correspondingly during the glides. Wobble does not change the cutoff.

AUTOMATION

When you click on the waveform, the **aiming playhead** param is switched on automatically. When you release the mouse, it is switched off. Originally this param's sole purpose is proper interpretation of MIDI CCs and recorded automation curves, but you can also use it to stop the playhead motion just for fun. But only if it is real fun. Don't use it if it's no fun to you. Otherwise what's the point? Man, this manual is such a pain in the Nevermind, we're already past the middle, aren't we?



Playhead position parameter is also used for MIDI CCs and automation curves. In general its values do not reflect the actual playheads' positions. The only exceptions when it may be close to a real position are the moments like when you change it by clicking on the waveform or sending a related CC, or when the next value of its automation curve is not equal to the previous value.

For better accuracy MIDI CCs for this parameter are mapped to the range defined by **selection start** and **selection end** values of the corresponding buffer.

Like with selection edges, valid **playhead position** range is normalized, i.e. spread over the [0.0, 1.0] range where 1.0 corresponds to current **buffer size**. However, it's minimal value is not 0.0, but a special small negative value. The reason is the following. While recording automation, many hosts smooth out small changes in a parameter values. When you aim a playhead to a target in a long buffer and then aim it again really close to the previous target, the difference in the values may be smaller than 0,000002. Most likely the host will smooth this difference out. In this case you can tell Ribs to drop the value of **playhead position** to this negative minimum as soon as you release the mouse. This way the smoothing will be avoided, and the negative value will be ignored by Ribs. You can find the **drop on MouseUp** option in the menu. This setting is saved with plugin states. Like with the selection edges, and for the same reasons, the menu is available by right-clicking the value in the square brackets on the bottom of the waveform display.

ⓘ Be sure to check the recorded automation curves. If the drops from the valid values to the minimum are not steps, but smooth fades, edit them to be steps so that they can be interpreted as accurate as possible later on. However, it is unlikely to have 100% accurate automation.

FILTER

Filter has a number of traditional parameters with obvious functions such as on/off switch, pass type, cutoff (**flt freq**), resonance (**flt q**) and cutoff modulation by envelope (**flt envelope**).

There are several filter models:

rs resonant on high q values, nice with noisy sources and **flt follows note** or for emphasizing on sub bass range.

sm simple. Useful for most of usecases.

fr not french, but formant. Sounds good with spectrally reach sources. **Flt q** serves as a timbre type.

Beside that there's a couple of interesting params that modulate the frequency of the filter.

flt freq *= playrate multiplies frequency by the absolute value of **playrate**. It's usefull while scratching the playhead, for example.

flt follows note sets filter to the frequency of the played note, multiplied by the ratio **base len nom** / **base len denum** and optionally by the corresponding beat ratio, when **use matrix for NOTES** param is on. Since long base lengths can set the frequency too low and in low-pass mode it can render output almost inaudible, **flt freq** serves as a minimal frequency value in this mode. When the frequency is too low, it will be set equal to the bigger integer of the calculated value. E. g. if the calculated cutoff should be 400 Hz, buf **flt freq** is set to 500 Hz, the resulting 400 * 2 = 800 Hz will be set. However, it does not restrict the effect range of **flt freq *= playrate**.

flt freq shift multiplies frequency by its value +1. After improvements in v 1.1.0 this param seems to be redundant and most likely will be removed in the future updates.

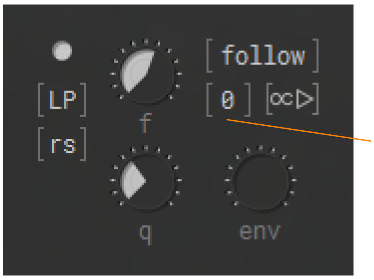
Just in case, here are the abbreviation of flt pass types:
LP - low pass, BP -band pass, HP - high pass, P - peak, BR - band reject.

flt freq

flt on/off

flt pass type

flt model

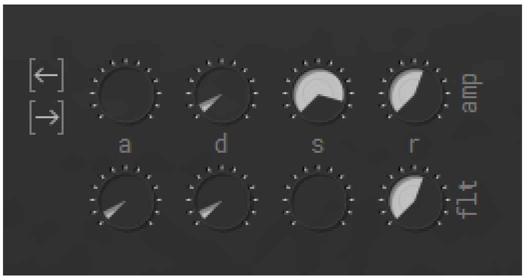


flt follows note

flt freq *= playrate

flt freq shift

flt q flt envelope



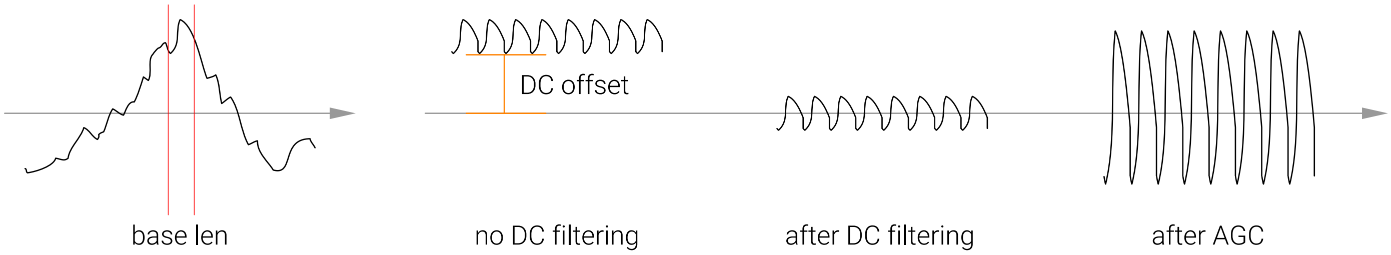
flt env A flt env S

flt env D flt env R

To the right of the filter block there are filter envelope controls. Its behavior is the same as that of the main voice envelope. See Envelopes section for details.

AGC & DC

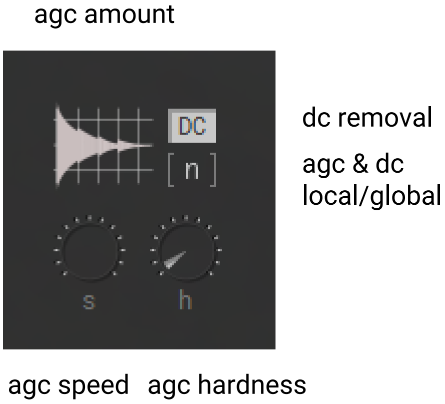
AGC stands for Automatic Gain Control. And DC stands for Direct Current, also called DC offset, or DC bias. It basically is just a very low or even zero frequency present in a signal. Often grains are cut so that the resulting waveform is too quiet and has an offset above or below the zero level. This situation gets increasingly more probable as the base length gets smaller. It can be cured by filtering out the DC and increasing the gain of the signal. This picture illustrates it:



You could apply AGC without removing the DC, and it would be fine sometimes. But usually it's better to filter out the inaudible low frequencies and only then increase the gain, because DC can take up a noticeable part of the dynamic range in the signal, and it will get only worse after applying AGC.

Maxing out the gain makes the dynamics of a signal almost flat. It doesn't always sound nice. **Agc amount** param can help you find the sweet spot. Usually just a little bit already produces the desired effect.

You can apply AGC and DC on a per-voice basis or globally using the **agc & dc local/global** param. Global mode takes up less CPU resources and makes signal a bit more saturated.



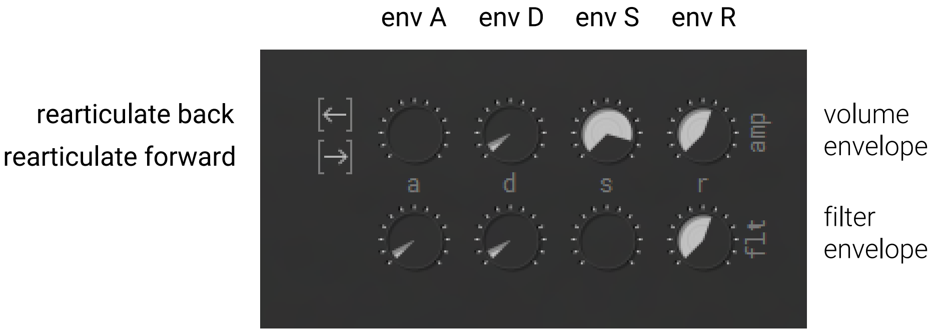
Agc speed param, which value is paradoxically measured in Hz, controls at what time interval inside a signal AGC scheme looks to decide whether the signal has a desired gain. By default it is set to auto and relies on internal parameters such as the factual base length. In SIMPLE mode it is set to 50.

Agc hardness is named after the perceptional effect it creates. The bigger the value, the more distorted it sounds. But It actually controls how fast the gain adjustments are being applied. Together with non-automatic **agc speed** it can create an interesting non-linear bouncy effect. The bounciness will depend on rapid changes inside the signal too.

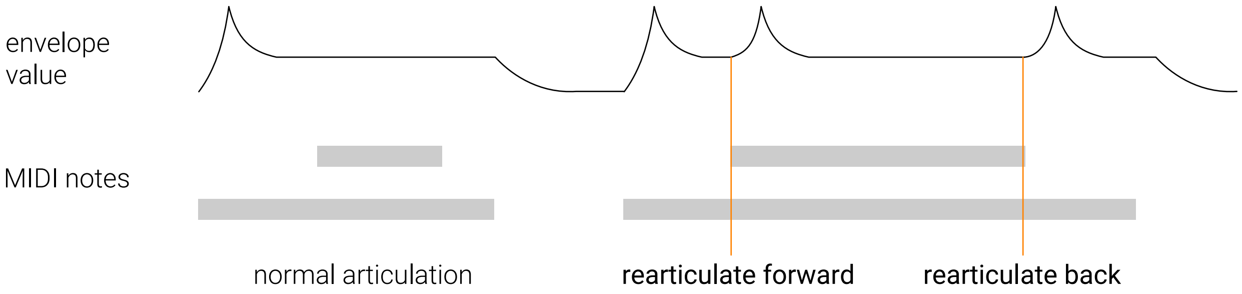
AGC and DC are applied after the filter.

ENVELOPES

Every Ribs' voice has one volume envelope (denoted as "amp" on the GUI) and one filter envelope. The envelopes have 4 stages. Attack, decay and release all have exponential character.



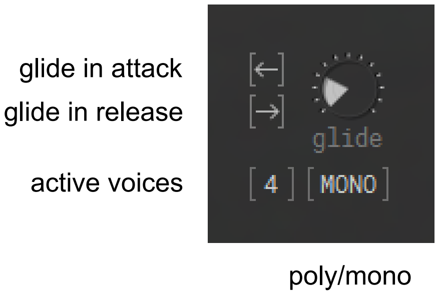
The effect of the following two params is heard in monophonic mode or during portamento.
Rearticulate back retriggers the envelopes when you release a note and have some older note still being pressed.
Rearticulate forward retriggers the envelopes when you still have a note pressed and then press a new one.
It's easy to understand by looking at the picture:



VOICING

Ribs supports sustain, sostenuto, portamento and legato MIDI CCs, as well as the pitch wheel CC.

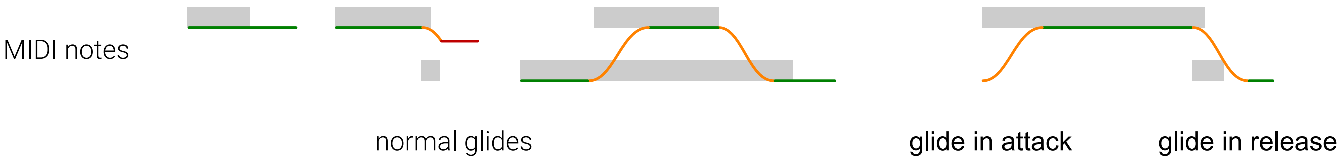
In polyphonic mode **active voices** param not only controls the CPU usage, but also can serve as an additional way to control the harmonic expression of a performance. It is especially noticeable with arpeggios. When you lower the number of active voices, the excess voices are not shut down immediately but are set to the release stage of their envelopes. Also note that this param is ignored in FX mode.



When a new note comes in and all available voices are taken, Ribs takes into account the existing harmonic diversity among the active voices as well as the **use buf for <note>** flags and **request picked buf** state and steals the most appropriate voice.

In monophonic mode **glide** parameter, as usual, controls the time it takes for a voice to change its frequency to that of a new note. When notes are played separately, this param is ignored. However, there are two modifiers for this param:

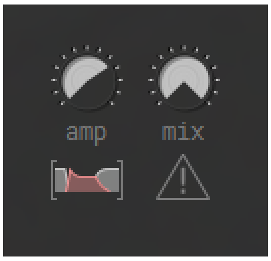
glide in attack makes a voice glide from the previous note even when it was played separately,
glide in release makes "tails" continue gliding to a new note even if the note was released before the glide was over.



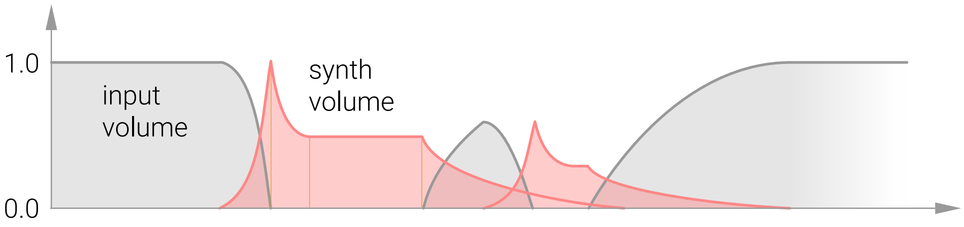
MIX

The functions of **amp** and **mix** params are obvious.

Extrude input makes the incoming audio pass through, but as soon as a voice (or several voices) starts playing, the input volume gets lowered by the growing envelope level. Decay and sustain are considered the stages during which none of the input signal is audible. If several voices are in release or attack stages and these stages are long, the input volume may be noticeably lower or even zero.



extrude input



AUTOMATION

All the peculiarities of automation of different params were described in corresponding sections, and in this section this information is just repeated.

Most of the parts of Ribs' engine are deterministic, but overall complexity practically makes it a chaotic system. This is mostly caused by the voice picking algorithm (however, you can get a hold of it to some extent by using request picked buf) and playheads' trajectories. That's why 100% precise automation recording is not guaranteed.

If you do scratching, aiming playheads or change the selection range, do remember to record the automation for **picked buffer**, **buffer size**, **selection start**, **selection end**, **aiming playhead**, **playhead position** and **select this/all** params.

While using MIDI CC do keep in mind that selection range CCs are mapped to the visible range of buffer contents, and **playhead position** CC is in its turn mapped to the selection range.

facebook.com/hvoyaaudio
vk.com/hvoyaaudio