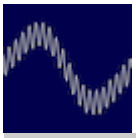

VividSynths

Welcome/Apps/ VividShaper/VividShaper reference manual

VividShaper Reference Manual

VividShaper is a wavetable AUv3 plugin synthesizer, where each wavetable consists of 128 samples. Instead of having a fixed number of wavetables to choose from, VividShaper allows you to program your own waves using the built in Lua programming language and change them over time.

The Lua-code is called many times per second. Default is 93.75 times/second, meaning it is updating after 512 frames at 48000 Hz sample rate. This can be changed as well (128 = 375 times/sec, 256, 512, 1024, 2048, and 4096 ≈ 12 times/sec). This allows the wavetable to be programmatically changed in realtime.

There are up to eight generators for polyphony (the number of generators can be set from 1 to 8). Each generator has in turn up to eight oscillators. Each oscillator has the following states that can be manipulated in Lua-code:

- volume
- volume at gate onset
- left-right panning
- note (including cent note)
- A wave table array containing 128 samples

The generator will use the volume information for each oscillator to change the output volume for each wave. Hence, you don't have to adjust the volume in the wave table.

Having a high update frequency and many generators is demanding on the CPU. Many times, a lower update frequency will give the same sound, but will require less CPU power. Also, it may not always be necessary with eight generators.

The first row is always a comment describing the name of the patch (– Patch: <name>). This name will be relapsed if the patch is saved in a different name. VividShaper will add it for you when saving the patch if you forget to add it yourself.

A simple example

In this example, we'll demonstrate how to create a basic patch using VividShaper's Lua API.

```
-- Patch: A simple example
wave[1] = VSTriangle(1,0)
vol[1] = VSADSRE(1,1,0.8,3,0,gatetimeon,gatetimeoff)
```

This example encapsulates the fundamental process of generating a wave and setting its volume envelope in VividShaper. The line **wave[1] = VSTriangle(1,0)** creates a triangle wave at the base frequency of 1 Hz and phase of 0 degrees (between 0-360 degrees). You need to provide both arguments even if you are not phase shifting the wave. The frequency here refers to the number of complete cycles of the Triangle wave form that fits within the sample (consisting of 128 sample points). A frequency of 1 Hz means one complete cycle of the triangle fits within the 128 samples.

This is not the playback frequency of the note (i.e. which note that is being played). The speed of the 128 samples is determined by note[1] for oscillator 1. We don't have to set note[1] explicitly, it is already set by the note you are playing on the keyboard. However, if we do wish to change it, for instance transposing one octave, we can do so by setting: **note[1] = notein+12**. Alternatively, if we wish to set it to a constant value, making it play back the same note even if play another note, we could do that as well: **note[1] = 40**.

The volume will then be set to an exponential volume ADSR envelope, with attack=1 sec, decay=1 sec, sustain=0.8, release=3 sec, startlevel=0. The name of the wave table will be “A simple example”.

This is all it takes to generate a simple wave. This will then be played back at different speeds, depending on the note being played.

API

List of built in VividShaper variables:

```
-- Input information
notein         -- The current input note for the active generator.
tempo          -- Current tempo (beats per second, e.g. 120).
timesignnum    -- Time signature numerator, e.g. 3 for three beats per bar (3/4). (Next version)
timesignden    -- Time signature denominator, e.g. 4 for 3/4 time signature. (Next version)
prevbeatpos    -- Previous beat position. (Next version)
beatpos        -- Current beat position.
velocity       -- The MIDI velocity of the note (normalised between 0 to 1).
gate           -- Either 1 or 0 depending on if gate is on or off.
gatetimeon     -- Time in seconds for how long the key was pressed.
gatetimeoff    -- Time in seconds since the key was released.
tick           -- The number of times the Lua code has been called since initiation (when tick=0).
cc[x]          -- Value of CC message x (between 0 to 127).
ncc[x]         -- Normalised value of CC message x (between 0 to 1).
par[x]         -- Parameter x. (Next version)
prevvol[x]     -- The volumes for each oscillator when the gate flipped from off to on.
midi[x]        -- MIDI message input array. One array of three elements for each midi message. (Next ve
playing        -- A value true/false. True value means the sequencer is running, false means it has sto

-- Output information
wave[x]        -- Wave array x (one array for each oscillator x).
panning[x]     -- Panning for oscillator x (between 0 - 1; 0 = left, 0.5 = middle, 1= right).
note[x]        -- Note output for oscillator x (default value is the same as note).
vol[x]         -- Volume for oscillator x. Default is 0.
gvol           -- Global volume, default = 1. Multiplied on the output to amplify or limit the audio.
updatefreq     -- Update frequency: 128, 256, 512, 1024, 2048, or 4096. Tells how many frames to wait b
generators     -- Number of generators (between 1 to 8).
active         -- A value true/false. A false value turns off the generator until it is turned on agair
midigenerator  -- A value true/false. A true value means only one generator will be running and it cann

-- Views settings
text           -- A text string that is printed in the waves view.
scaleview      -- The waves view is normally between [-1, 1]. Setting scaleview=0.5 changes to [-0.5, 0
wavesview      -- Which wave to show, between [1, 8].
```

List of built in VividShaper functions:

```
-- Wave generators
wave[x] = VSSin(frequency,phase)           -- Sine-wave. frequency=1 is base, phase is in degrees.
wave[x] = VSTriangle(frequency,phase)      -- Triangle-wave.
wave[x] = VSSaw(frequency,phase)           -- Sawtooth-wave.
wave[x] = VSSquare(frequency,phase,width)  -- Square-wave. Width is between 0 to 1.
wave[x] = VSNoise(seed)                    -- Random generated wave, given the seed integer.

-- A neural network autoencoder has been trained with over 4000 different wavetables to be able to
-- reconstruct waves from a two dimensional "latent space".
latentspace = {0.5, -0.2}                  -- latentspace is an array of two elements in the range [-1,
wave[x] = VSAutodecoder2(latentspace)      -- Creates a new wave given the 2D coordinate in latent spac
latent  = VSAutoencoder2(wave)             -- Obtain latent space coordinate for the given wave

-- Wave manipulators
wave[x] = VSWaveFold(wave[x],amplify)         -- Wave will be folded outside the range [-1, 1].
wave[x] = VSNorm(wave[x],threshold,normvalue) -- Normalise the waveform to normval if max amplitude is

-- Wave math operators - arguments can be either arrays or scalar factors
wave[x] = VSMul(wave1,wave2) -- Multiply element-wise wave1 with wave2
wave[x] = VSMul(wave1,1.5)   -- Multiply element-wise wave1 with 1.5
wave[x] = VSDiv(1,5,wave2)   -- Divide element wise 1.5 with wave2
wave[x] = VSAdd(wave1,wave2) -- Add element wise wave1 with wave2
wave[x] = VSSub(wave1,wave2) -- Subtract element wise wave1 with wave2
```

```
-- Envelopes
-- The level is the initial volume. Normally, it would start at zero, but could also be initiated to
-- start higher.
vol[x] = VSADSR(attack,decay,sustain,release,initlevel,timeOn,timeOff) -- initlevel is initial volume
vol[x] = VSADSRE(attack,decay,sustain,release,initlevel,timeOn,timeOff) -- Exponential release

-- Filters (x = oscillator)
-- A quad filter consists of 5 coefficents: b0, b1, b2, a1, a2. The 'biquadcoeff' is an array
-- consisting of these 5 coefficients. The functions VSLowpass, VSBandpass, and VSHighpass will
-- set the coefficients given the values. When setting these filter coefficients, both the cutoff
-- frequency and the note value are considered. This is because the cutoff frequency is typically
-- specified relative to the note's frequency.

biquadcoeff = VSLowpass(cutoffFreq,resonance,notein)  -- Helper function to set the quadfilter values
biquadcoeff = VSBandpass(cutoffFreq,resonance,notein) -- Helper function to set the quadfilter values
biquadcoeff = VSHighpass(cutoffFreq,resonance,notein) -- Helper function to set the quadfilter values
wave[x] = VSBiquad(wave[x],biquadcoeff)                  -- Apply filter on wave using quadfilter setti

-- MIDI output
-- VSTick is a help function to determine if the beat position crosses a certain time interval.
-- For instance, if length=1/4 (corresponding to a quarter note), VSTick will return true every time
-- the beat position (beatpos) crosses over to a new beat. If length=1, it will return true every time
-- beatpos crosses over to a new bar.

VSMIDI(command,data1,data2) -- Send MIDI data. (Next version)
VSMIDIPlay(channel,note,velocity,length) -- Play a given note on the given channel. length=1/4 means a
VSTick(prevbeatpos,beatpos,length) -- Returns true for every new time interval with the given length. (
```
