homados

I don't have a tagline yet! :O

Version 0.0.1

User Guide

Table of Contents

Overview

This is all a big work in progress! Thanks so much for making it out to version 0.0.1 of homados – the first public one and the formal embarkation of this project.

I'm not entirely certain what my aspirations for this project are yet but roughly homados exists as a way to pretty easily generate a whole bunch of different kinds and shapes of sound signals with pretty good accuracy and ease. It has no UI yet, there's a few things that are "in" but don't work yet, and I still have a whole lot more DSP type stuff to learn before I think it can really be great. But I think this is a good foot in the door and I hope this can find its way to being really useful.

Below, there is a bunch of information about how to use the thing, what it can do, some limitations, reasons why I made certain decisions, and references to some of the giants whose shoulders that this work rode on. This is all very casual right now while I publish it to get some first opinions on it before I dive too much deeper.

There will be plenty of hyperlinks colored blue for you to click on to help navigate to further reading and get more information about some topics that may be too much to cover here or better explained by someone else.

Ok, that's it I think. Enjoy!

Getting Started

Running Commands

homados works on single commands to generate its output, so everything is decided before running. A command might look something like this:

```
cargo run ~/Desktop/ "My Awesome Sound" -t pseudo_velvet -d 15 -s 96000 -w exp1_out --Param1 0.995
```

To get started, make sure you have a command-line interface handy. On macOS this might be terminal, CMD for Windows users, or maybe you have something else you like. Since homados is built in the rust programming language, we'll need to install it so we can use its built in cargo package manager to help us get our program running. Learn how to get and install it here.

The first two words "cargo run" let rust know it's time to run the program, and then we follow with the path on our computer to the output file and then the name of the file. These 4 strings of text are ***always mandatory*** when running homados.

homados uses various arguments and flags to specify anything that might affect our output, but otherwise has default values set for everything. This means you can run just the first 4 strings of the program and an output will be generated with no fuss. Specifically, this default output will be a 10 second long, mono wave file containing uniform white noise at 48kHz / 24 Bit.

Let's look at what options we have before going any further.

Getting Started

Command Options

Following our 4 strings we can add -h to have homados show us a help menu containing all the possible types of arguments and flags at our disposal to make some noise any time. Here's a list of what those options are:

	Short	Long	Description	Default Value
▪	-s	--SampleRate	Sample Rate	48000 Hz
▪	-b	--BitDepth	Bit Depth	24 Bit
▪	-c	--ChannelCount	Channel Count	1 Channel
▪	-t	--SoundType	Type of sound	"White"
▪	-d	--SoundDuration	Duration of sound in seconds	10 Seconds
▪	-f	--BaseFrequency	Base generator frequency	440.0 Hz
▪		--Param1	Generator-Specific Parameter 1	1.0
▪	-w	--WindowShape	Shape of the gain envelope / "fade window"	None
▪		--WindowCurve	Modifier for fade window curve shape	2.0
▪	-g	--Gain	Gain scalar value	1.0
▪	-v	--verbose	Display verbose output	**N/A**
▪	-h	--help	Print help	**N/A**
▪	-V	--version	Print version	**N/A**

We can add these options to a command in any order we'd like, and can omit them as desired to use default values instead. The last 3 options are flags, and act as toggles for information.

Getting Started

Option Minutiae

A few different command options are worth noting specifics regarding their behavior.

- Currently only the following bit depths are supported: 8, 16, 24, and 32 Bit Integer. 32 Bit Float support is planned but currently unavailable, and any other value will produce an error or incorrect output.

- Multichannel output ***is technically*** supported, however, currently only in an experimental and preliminary state where each channel performs the same behavior inconsistently. I am currently deciding on the best way to move forward with this – for now, it may work perfectly for your uses, but consider it like a little occasional bonus rather than a full feature.

- Regarding all the "Noise Color" types of sound: They are normalized / scaled to be approximately -18.06 dBFS ($1/8^{th}$ full-scale amplitude) RMS. This is because each different type of noise with no level scaling may be wildly different-seeming in volume from the next due to how they are generated. This ensures that no matter which noise color you pick, there will be a consistent and standard level. For "white" noise specifically, you can easily get -1.0 to 1.0 amplitude peak by picking the random uniform noise type, which is used by most all the other noise color functions as a uniform "white" noise source.

- There are two currently disabled sound types – the linear and logarithmic sine sweeps. This is because the current generators for sine are quite noisy and problematic. They work somewhat okay for static tones but produce undesirable distortion once modulated. An upcoming goal is to change these generators to cleaner phase accumulator versions, and then re-enable these (and make other new) sound types. They still exist in the code and are documented for the curious individual to play with, and to leave the infrastructure in place to easily add these back in as a feature in the future.

Signal Source Types

Simple Signals

Constant

- DC-Offset

  A constant signal at 1.0 amplitude. The gain scalar may be used to set this to any other value.

- Silence

  A constant “silent” signal. This is equivalent to DC-Offset (or any other signal for that matter) with a gain of 0.

Sinusoidal

- Sine

  A sine wave function at a constant, user-defined frequency.

- Cosine

  A cosine wave function at a constant, user-defined frequency.

- Sine Sweep: Linear ***(Currently Disabled)***

  A sine wave linearly varying in frequency between user defined endpoint values over the entire sound’s duration.

- Sine Sweep: Logarithmic ***(Currently Disabled)***

  A sine wave logarithmically varying in frequency between user defined endpoint values over the entire sound’s duration.

- Quantized Sine

  A sine wave rounded to a smaller set of possible amplitude values than what is dictated by default from the sample rate.

  Currently, the amount is ***not*** user-configurable.

  In a future update this will be addressed as more quantization modes are added – for now it remains as a proof-of-concept.

Section Heading

Simple Signals

Basic Shapes

- Saw

  A square wave function at a constant, user-defined frequency.

- Square

  A square wave function at a constant, user-defined frequency.

- Triangle

  A triangle wave function at a constant, user-defined frequency.

Impulses

- Unit Impulse

  Also known as the Dirac delta function, this is a single sample impulse at full amplitude.

- Dirac Comb / Impulse Train / Needle Pulse

  A Dirac comb function at a constant, user-defined frequency.

Signal Source Types

Noise

Random Noise

- Random Uniform

    A full-range sequence of pseudo-random values using rust's rand crate defaults.

    This is the base "random" source for all the other noise generators.

- White Triangular

    Random Triangular distribution noise, scaled amplitude.

- White Binary / Bernoulli

    Random Bernoulli distribution noise, scaled amplitude.

White Noise

- White Uniform ***(Default)***

    Random Uniform noise, scaled amplitude.

- White Normal / Gaussian

    Random Gaussian distribution noise, scaled amplitude.

Pink Noise

- Pink Kellet Econ ***(Default)***

    Pink noise via white noise filtered with Paul Kellet's Econ method as described here, scaled amplitude.

- Pink Kellet Refined

    Pink noise via white noise filtered with Paul Kellet's Refined method as described here, scaled amplitude.

Signal Source Types

Noise

Brown Noise

- Brown

    Brown noise via white noise filtered with an EMA Low Pass filter at 20Hz.

Blue Noise

- Blue Kellet Econ ***(Default)***

    Blue noise via white noise filtered with Paul Kellet's Econ method, then an EMA High Pass filter at 20kHz.

- Blue Kellet Refined

    Blue noise via white noise filtered with Paul Kellet's Refined method, then an EMA High Pass filter at 20kHz.

Violet Noise

- Violet

    Violet noise via white noise filtered with an EMA High Pass filter at 20kHz

Sparse Noise

- Pseudo-Velvet (Consecutive)

    Velvet-like noise implemented as a random uniform noise source with a threshold that determines how often it will be rounded to zero, 1, or -1.

    Threshold ranges from 0.0 (all values have 1.0 magnitude) to 1.0 (all values have 0.0 magnitude) – it may be thought of as a scale for how likely 0.0 is to appear.

    There is no limit on whether consecutive impulses may appear or not.

Acknowledgements

Thank you to the following people and groups:

Tomato Tribe
You know who you are, you know why

The Rust Audio Discord Server
For letting me ask a lot of really annoying questions on there regarding this project, and giving me a lot of really helpful information and talking me through some of the problems I've had along the way.

Theres more of you out there

Just gotta figure out who

Oh and also to you

The user

For using this (and helping make it better as a result)

Alright. Thanks! This was version 0.0.1 !!!!

References

[1] The Paul Kellet filtering methods used for our pink (and blue) noise were retrieved from here:
https://www.firstpr.com.au/dsp/pink-noise/

[2] The Signalsmith crossfade curve functions were derived and retrieved from here:
https://signalsmith-audio.co.uk/writing/2021/cheap-energy-crossfade/#fractional-power-law