Processing Chain Explanation

Edoardo Di Pietrantonio and Giorgio Povegliano
edoardo.dipietrantonio@studenti.unipd.it
giorgio.povegliano@studenti.unipd.it

1 Introduction

This document aims to provide an in-depth explanation of the workflow for our implementation of the granular synthesis algorithm. The purpose is to explain the processing chain and the data flow within the model used to design the application "AGSynth - Granular Synthesizer for Automotive." To achieve this, we will refer to the Simulink model underlying the application, taking each simple block and explaining its functionalities. For the tutorial regarding commands and the operational guide of the application, please refer to "Granular_Synthesis_Guide."

2 Processing Chain

The main Simulink model is called "merge_variable_grain.slx." All other models and MATLAB code present inside the folder are subfunctions used by the main model. When opening the model, you will see what is depicted in Figure 1. We start by noting that the blue box on the left is merely a display for the values of speed and RPM, which is not particularly relevant for the purpose of this document. All other blocks will now be explained in detail, following the path of the data from where it is generated to where it is output.

2.1 Modeling of the Car, Actuators and Sensors

The first set of blocks we will explain is represented in Figure 2. These blocks compute the physical model of the car and manage the sensors and actuators within the models.

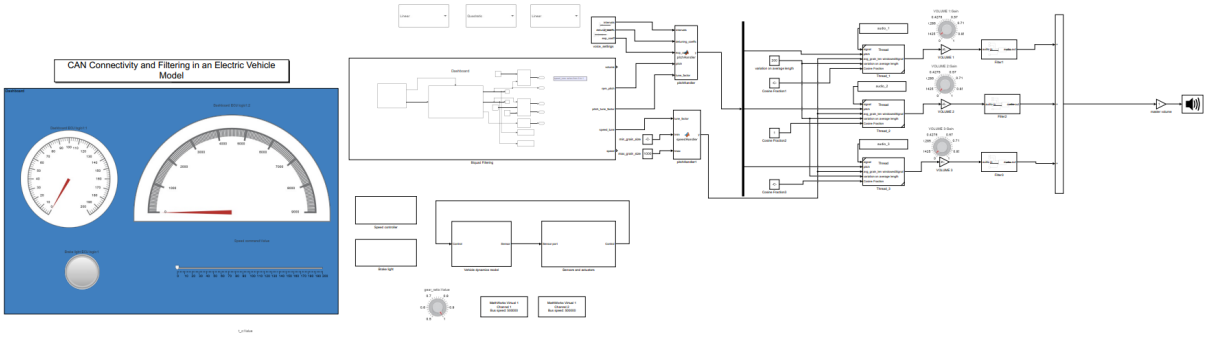

Figure 1: General View of the Model

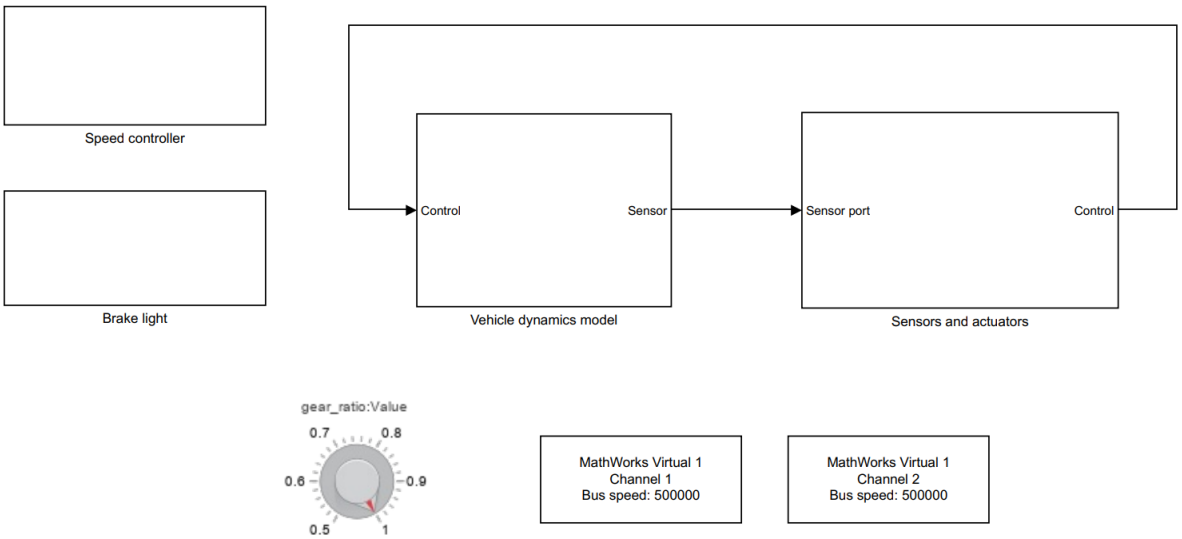

Figure 2: Physical Model of the Car, Sensors and Actuators

1. The **vehicle dynamics model** depends on lots of fixed variables, the most important ones are:

   (a) Mass of the chassis.

   (b) Mass of the wheels.

   (c) Wheel radius.

   (d) Gear ratio.

   For additional informations you can open this block and read the matlab function that simulates the car.
   inputs: control variables throttle/brake coming from actuators
   outputs: motor RPM, speed in mph

2. **Sensors and actuators** block is responsible of simulating the acquisition of signals and physical values coming from the vehicle model, computing and transmitting the control action and logging data into CAN Bus.
   inputs: Throttle / Brake command coming from CAN Bus (eg coming from Speed controller block), motor RPM, speed in mph
   outputs: actuators output directed to **vehicle dynamics model**, motor RPM and speed in mph into CAN Bus

3. **Speed controller** block computes the proper Throttle / Brake CAN Bus command to feed into the **Sensors and actuators** block. It implements both cruise control and normal driving mode. The controller of choice for cruise control is a simple discrete PI controller.
   inputs: Actual Speed and Speed Command from CAN Bus
   outputs: Throttle / Brake command into CAN Bus

4. **Brake light** block is not important

5. the **Gear ratio knob** regulates the ratio between engine shaft and wheels. Gear ratio equals to 1 is the default and suggested value.

2.2 From Sensors Data to Input Data

The **Biquad Filtering** (see figure 3) block is responsible for converting data from the CAN Bus into useful variables for the Granular synthesis.
inputs: Speed, RPM and Throttle command from CAN Bus
outputs: volume (not used), pitch, pitch tune factor, speed, speed tune.
The two blocks on the right (**speedtotune** and **rpmToPitch** are responsible for converting and adjusting the output of the previously presented block so that the data is meaningful for the pitch handlers, outside the Biquad Filtering block.

- Speed Tune, which is a variable ranging from 0 (when the car is at full stop) to 1 (when the car is at full speed).
- Rpm_pitch is a variable that ranges from a base pitch to a certain range that is specified in the gui.
- Pitch_tune_factor is a variable ranging from 0 to 1 that increases with linearly with the RPMs.

Outside the Biquad Filtering block we find the following blocks:

1. **Voice settings** block which is responsible for generating the pitchHandler parameters via
2. **pitchHandler** block that controls the final pitch to feed into the Granular synthesis system.
   inputs:
   - intervals: offset from base pitch
   - detuning_coeffs: detuning related to pitch_tune factor
   - exp_coeff: define hoe pitch increases with rpm (linear, quadratic, cubic)
   - pitch: coming from biquad filtering block
   - tune_factor: pitch_tune factor coming from biquad filtering block

   output: pitch for Granular Synthesis
3. **speedHandler** block that controls the average grain length based on the speed.
   inputs: speed_tune, tmin and tmax (which are min and max duration of the grain in milliseconds)

In this context, the Pitch is modified according to the user's preference: an initial offset

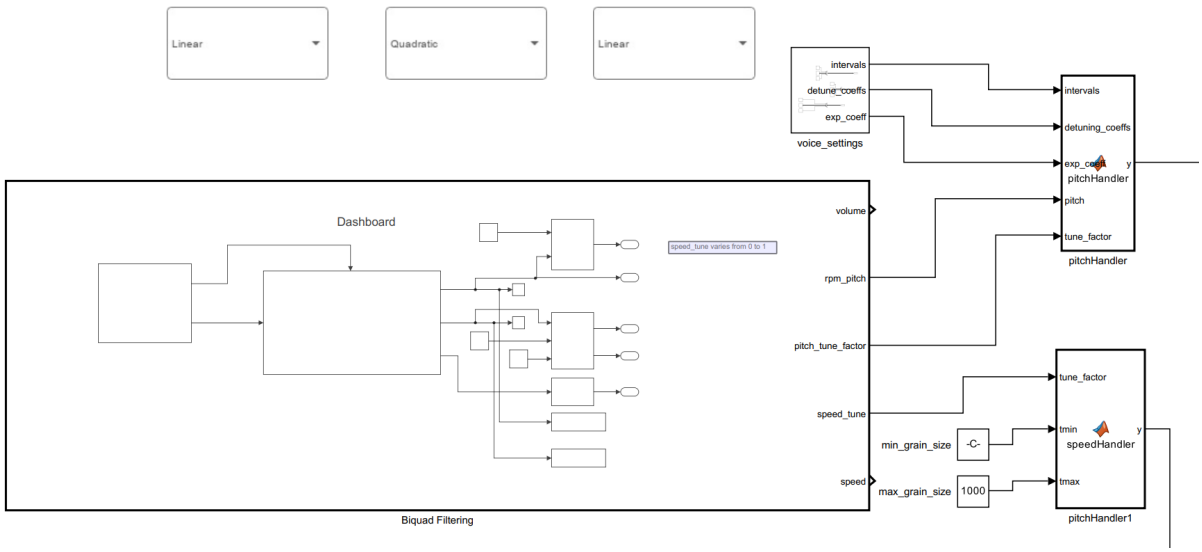

Figure 3: Blocks that convert Sensor Data in Input Data for the Granular Synthesis Algorithm

is applied, detuning is incrementally applied using the Speed Factor, and the Pitch could be raised to the power of two or three based on the user's preference. Also the average grain length is computed using the Speed Factor. Now the two inputs of the granular synthesis algorithm are ready: Pitch and Grain Length.

2.3 Granular Synthesis

Let's now discuss the implementation of the granular synthesis algorithm itself. The blocks involved in the process of generating and reproducing grains are shown in Figure 4. The block that implements the algorithm itself is the **Thread** block. In our algorithm there are three **Threads**, each operating independently.
inputs:

- average grain length, variation on average length (common to all **Threads**)
- cosine fraction, audio file, pitch (specific for each **Thread**)

outputs:

- buffer of 1024 samples of the windowed and repitched grain

2.4 Threads

We now take a deeper look at what happens inside each Thread. As we can see from Figure 5, each Thread is made up of 8 voices summed together, we now discuss how the output of each voice is generated.
Referring to Figure 6 the **handler** block is responsible for selecting the grain length and its cue point (starting sample) inside the audio file.
inputs:

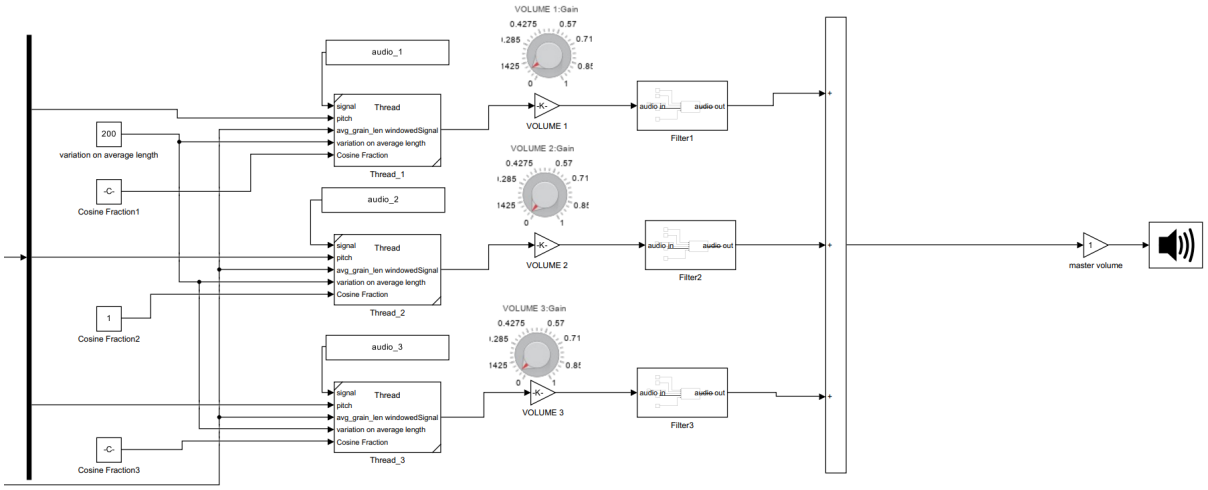

Figure 4: Granular Synthesis Algorithm

- average grain length, variation on average length

outputs:

- randomly selected grain length, associated cue point of the grain (in the range from 1 to audio_file_length - selected_grain_length)

The block called **Signal Processing** handles the extraction of the samples from the audio file, the repitching process, and the windowing of the grain.
inputs:

- cue point, grain size (output of the **handler** block)
- audio file, pitch, cosine fraction

outputs:

- buffer of 1024 samples of the windowed and repitched grain

The grain is extracted from the audio file and the window associated with it is generated; both are stored in a persistent variable. The Pitch Shift is applied to the whole grain right after its extraction. Samples are then extracted from the grain, 1024 at a time for each iteration, until the grain ends. The window is also applied incrementally, 1024 samples at a time. If the grain's length is not a multiple of 1024 (which is often the case), the last buffer of 1024 samples, which is not full, will be appended at the beginning of the first buffer of the successive grain. In this way, we can guarantee a fixed output of 1024 samples at the same sampling frequency as the input. The output of each thread at each iteration is the sum of all the 1024-sample buffers of all the 8 voices.

After each thread, there is an option for the user to adjust the **voice gain**. Each thread has its own customizable **filter**, where the only parameter that can't be adjusted in real time by the application is the Stopband Attenuation, which is set at 60 dB. The outputs of each

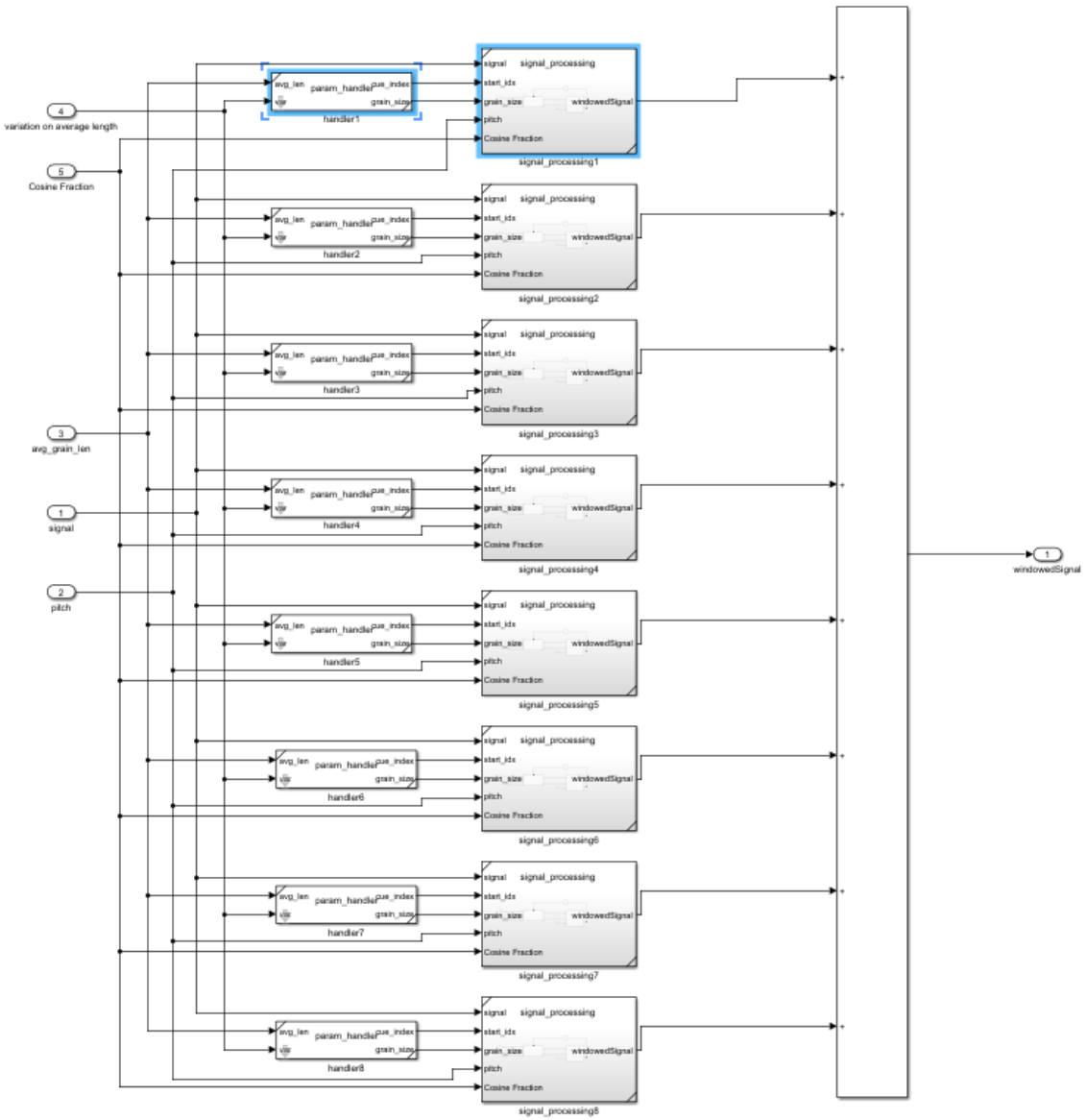

Figure 5: Granular Synthesis Thread

thread are then summed together, and there is an option to adjust the **master gain**. At the end of the chain, there is the **device writer**, which allows the sound to be played back through the speakers.

2.5 Scheme of the Processing Chain

To sum up the general structure of the model and provide a clearer explanation of the data flow, we provide the following schemes. The first one in Figure 7 represents the car physical model, it better explains the blocks presented in Figure 2.

In Figure 8 we have the diagram representing the Granular Synthesis algorithm implementation, it resembles the general structure of the Simulink model depicted in Figure 1.

Finally, in Figure 9 we show the schematized version of a single voice inside a Thread, recalling the blocks presented in Figure 5.

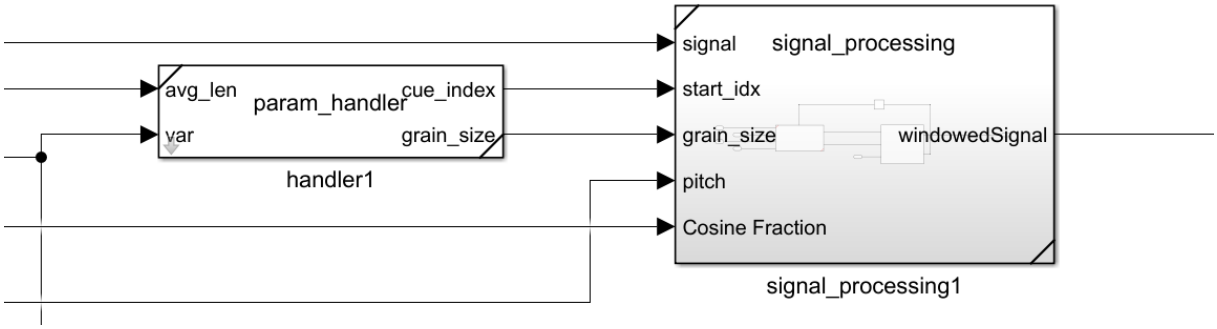

Figure 6: Single Voice inside a Thread

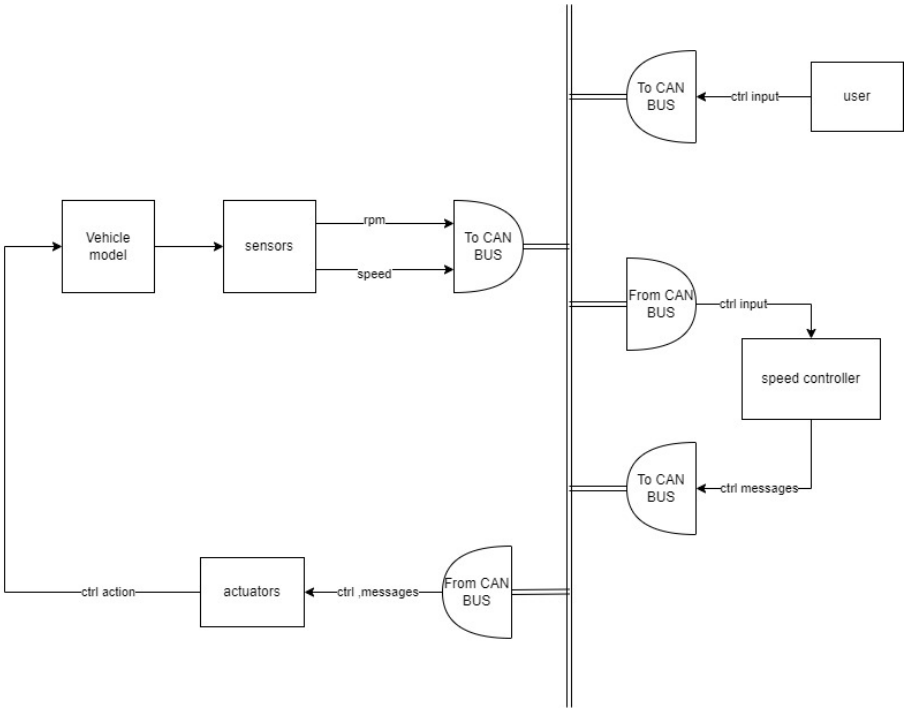

Figure 7: Diagram of the Car Model

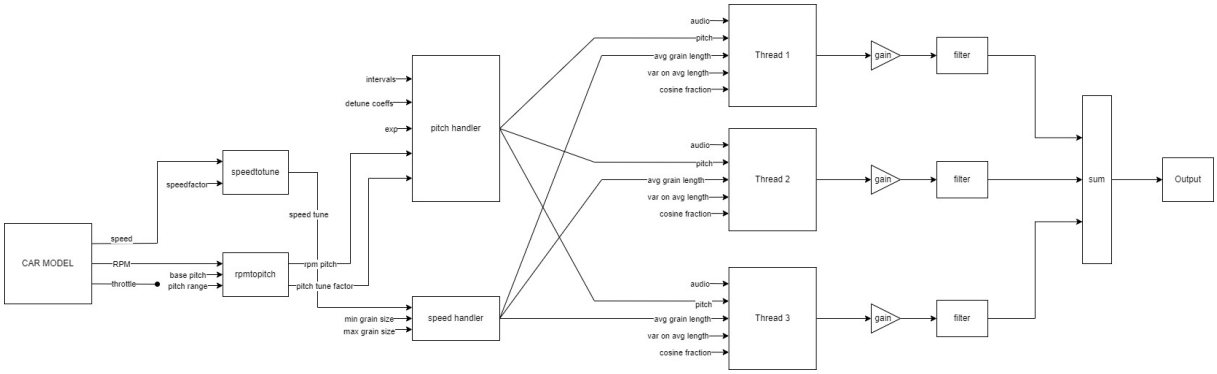

Figure 8: Diagram of the Granular Synthesis implementation

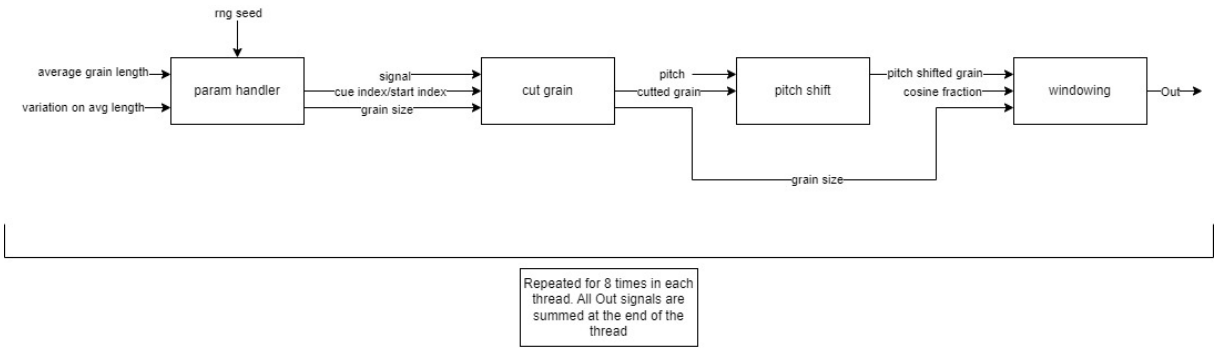

Figure 9: Diagram of one of the voices of a Thread