MAP
Final Project Report

Jack Walters
j.h.walters@se19.qmul.ac.uk

Tuesday 30th June, 2020

Abstract

This report presents a wind chime and wind modelling program written in C++ for use on the Bela board. The wind chimes are modelled through modal synthesis, whereby the principle modes of the chime are recreated through a set of non-linearly decaying sinusoids in variable amplitude envelopes. The wind is created by processing white noise through four parallel computed low-pass filters, and these two halves of the system interact on the basis of a sigmoid probability function.

1 Introduction

Spectral modelling has been a topic I have been interested in implementing in C++ with the low-latency capabilities of the Bela board. A central source of inspiration for this project was Teemu Lukkari and Vesa Välimäki's 2004 paper *Modal Synthesis of Wind Chime Sounds with Stochastic Event Triggering*[2]. In this paper Lukkari and Välimäki outline a parametric synthesis model through which a stochastic-based wind model interacts with sum of exponentially decaying sinusoids. This technique of modal synthesis was also covered in Xavier Serra and Julius Smith III's *Spectral Modeling Synthesis: A Sound Analysis/Synthesis System Based on a Deterministic Plus Stochastic Decomposition*[3]. This paper explores both the analysis and re-synthesis of sound as a collection of resonant modes functioning as the *deterministic* component, and also the emulation of the residual noise which exists between the modes, or the *stochastic* component. After investigating the analysis element of how these chimes will sound after being struck by the clapper which sounds them, I found the next question to naturally be regarding how they would behave in a real-world environment. After considering how wind chimes behave in the presence of gusts of wind, designing a natural-sounding unpredictable system through which wind is generated and interacts directly with the chimes seemed like an appropriate context in which the modelling wind chimes would be placed. Lukkari and Välimäki present an altered sigmoid function, through which the "energy" of the system is converted into a probability of one of five chimes sounding. This function is used to provide a natural sounding behaviour to the architecture, and demonstrate the variable spectral qualities wind chimes exhibit when placed in a natural environment.

Chime0	Mode 1	Mode 2	Mode 3	Mode 4	Mode 5	Mode 6	Mode 7	Mode 8	Mode 9
Frequency (Hz)	829.56	2224.63	4191.66	6586.79	9306.54	10353.6	10570.4	10892.7	11328.3
dBV	-6.94	-4.42	-13.61	-15.08	-19.08	-18.98	-19.13	-19.73	-20.17
V	0.449	0.601	0.209	0.176	0.111	0.112	0.110	0.103	0.098

Chime1	Mode 1	Mode 2	Mode 3	Mode 4	Mode 5	Mode 6	Mode 7	Mode 8	Mode 9
Frequency (Hz)	1118.3	2959.48	5521.3	8594.23	10371.8	10701.2	10701.2	11956.1	11956.1
dBV	-3.78	-6.92	-16.24	-16.99	-17.78	-17.78	-18.33	-18.33	-20.07
V	0.647	0.451	0.154	0.141	0.129	0.134	0.121	0.118	0.099

Chime2	Mode 1	Mode 2	Mode 3	Mode 4	Mode 5	Mode 6	Mode 7
Frequency (Hz)	1636.85	4268.98	10246.1	10392	10551.2	11814.2	11814.2
dBV	-4.68	-11.26	-19.09	-19.09	-17.68	-20.02	-19.47
V	0.583	0.274	0.111	0.125	0.131	0.1	0.106

Chime3	Mode 1	Mode 2	Mode 3	Mode 4	Mode 5	Mode 6	Mode 7	Mode 8	Mode 9
Frequency (Hz)	1333.49	3466.18	6483.16	9951.58	10421.5	10855.8	11448.7	13482.7	14975.5
dBV	-4.03	-9.41	-16.14	-16.69	-16.69	-18.28	-17.53	-20.62	-23.76
V	0.629	0.338	0.156	0.146	0.156	0.122	0.133	0.093	0.065

TABLE 1: Frequency, decibel volt and volt readings for the four wind chimes

2 Design

2.1 Modal Synthesis

As a reference against which to model the wind chimes, a set of four wind chime recordings were downloaded from user '3bagbrew' on CC licensed audio sample site **freesound.org**[1]. These recordings were loaded into audio analysis application **Sonic Visualiser** and their principle modes were recorded. Judgments on which modes were to be recorded and later modeled were made visually based on their dBV value relative to surrounding frequencies, as shown in Figure 1.

Table 1 exhibits how the dBV readings taken in Sonic Visualiser were converted to linear voltage readings V that could be used in the mode synthesis. The frequency and voltage readings are loaded into `chime.cpp` and combined with with Lukkari and Välimäki's T60 decays times for different modes [2], these values are used to generate amplitude envelopes for between 7 and 9 sinusoidal functions, dependent on which chime is being generated. The attack time is assumed to be `0.0001` for all modes, and is generated linearly from 0V to the amplitude of the respective mode. The decay envelope is generated with two `log()` functions to simulate the non-linear decay times of the principles modes outlined by Lukkari and Välimäki[2]. Line 5 of Listing 1 shows how the index at which the envelope will have finished decaying is set, and lines 8 to 10 calculate this logarithmically decreasing envelope sample by sample and store it in `currentLevel_` at mode index `j`. To ensure that we index the correct number of samples, lines 12 to 16 check to see if the `currentSampleIndex_` has surpassed the mode-dependent `nextStateSampleIndex_`; if it does envelope state is set as off as the mode has finished decaying.

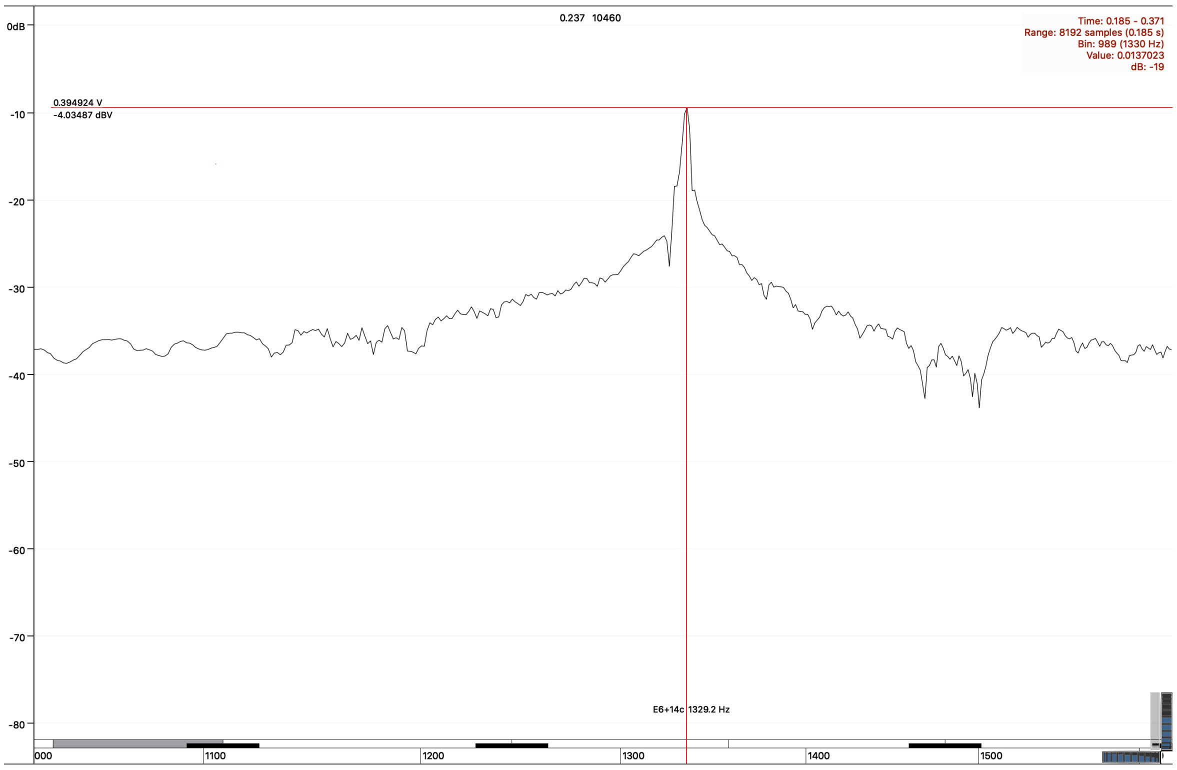

FIGURE 1: Meauring the first mode of chime3 in Sonic Visualiser

LISTING 1: Decay envelope generation

```
else if (currentState_[j] == envelopeStateDecay)
{
    currentSampleIndex_[j]++;

    nextStateSampleIndex_[j] = stateValue_[envelopeStateDecay][j]
        * sampleRate_;

    float multiplier_ = 1.0 + (log(minimumLevel_) - log(currentLevel_[j]))
        / (nextStateSampleIndex_[j]);
    currentLevel_[j] *= multiplier_;

    if (currentSampleIndex_[j] >= nextStateSampleIndex_[j])
    {
        currentState_[j] = envelopeStateOff;
        currentSampleIndex_[j] = 0;
    }
}
```

Following the triggering of a chime, calculating 9 sets of `log()` function pairs for every sample is taxing on the CPU and tended to cause buffer underruns, so in order to preserve the natural sounding non-linear decay behaviour of each mode, while also allowing control over individual modes in the general architecture, every enveloped sinusoid that constitutes a mode is stored in a wavetable when the `setup()` function is run. This ensures that each `log()` function is only run once throughout the entire duration of the program, but also allows malleability in terms of individual mode amplitude in `render()` (see 2.3). When a chime is triggered, the system energy is converted into individual amplitudes for each mode. System energy variable `windStrength` is used in an `exp()` function in `render()` to amplify upper modes exponentially in relation to lower ones. This leads to the chime models sounding brighter and louder when the generated wind sounds stronger. Furthermore, for enhanced efficiency, the `math_neon` library was used in sin function emulation, as it makes use of the Bela board's inbuilt NEON SIMD (single instruction multiple data) extension (see 2.2 for further details).

2.2 Procedural Wind Generation

In order for the modes to be excited in the natural environment this design attempts to emulate, there needed to be some natural sounding energy impulse into the system. The wind design is constituted by two principle components: white noise generators and a bank of low-pass filters. White noise is generated by a random number generator inside of `util.h`, whereby a random float value between -1 and 1 is calculated every sample. This white noise is processed by four separate low-pass filters, the coefficients of which are calculated in `filter.cpp`. In order to model the random behaviour of wind, once a second, these 4 filters are assigned random random frequency and Q values in a given range. Not only do the four filters behave independently to each other, they also operate in different frequency ranges. The cutoff frequency of `LPFilter[0]` operates in the range of 0Hz to 750hz, while the cutoff of `LPFilter[3]` varies between 1200Hz and 3000Hz. Low-passed white noise cutoff at an increasingly higher frequency begins to sound more like a gale-force storm than a light breeze. From a design standpoint, this means that greater system energy can be represented by the stronger presence of filters low-passing white noise at higher frequencies. Listening to wind, it is always in a constant state of transition between amplitudes and frequencies; in order to model this, `Ramp` objects are used to transition between random cutoff frequency and Q values which are generated once a second. Irrespective of whether the ramps have reaches their target, the intermediate values they return are copied into arrays `b0_arr` through `a2_arr` for use in assembly language function `NEONFilter.S`.

The Bela board's ARM Cortex-A8 processor contains the NEON unit, the SIMD architecture of which allows for simultaneous processing of multiple elements of the same data type. It stands to reason that utilising this architecture would allow for faster processing of four low-pass filters than in conventional C code. After the `NEON_IIRFilter` function is passed a random float between -1 and 1 as an input argument, it stores the memory addresses of coefficient and previous state arrays in `r` registers before loading their contents into `q` registers. Ensuring that registers vital to general system function are pushed to the stack, the assembly language then utilises the NEON capabilities of the ARM processor to parallel process each filter's multiplication of the coefficients with previous state values in the manner of the 2$^{\text{nd}}$ order low-pass differential equation. Due to the finite number of NEON `q` registers, the `q` registers are reassigned when they are no longer needed. Each filter's final output value ends up in their own respective `s` register inside a `q` register; in order to 'funnel' them into one value, two pairwise vector addition `vpadd.f32` functions are used to add `s` registers with themselves until one culmulative `s` register remains.

2.3 Probability Mapping

In order for the wind and chimes to convincingly interact with each other, Teemu Lukkari and Vesa Välimäki's altered sigmoid function shown in equation 1 is used in `eventTrigger.cpp` to non-linearly translate system energy into a probability between 0 and 1.

$$p(E_n) = \frac{1}{1 + c\,\exp(-2E_n)} \tag{1}$$

Where $p(E_n)$ is the probability that arises, given system energy input E_n and the constant c is chosen by Teemu Lukkari and Vesa Välimäki to be 99, in order to give a conveniantly low value to $p(0)$[2]. The total wind strength of the system is between 1 and 99, so before this equation this range is mapped between 0 to 4.59, which are the upper and lower limits of the equation respectively before the floating point precision becomes inadequate and displays either `0` or `1.0`. In such a system, having a certain probability a chime will either sound or not is unnatural, hence $p(E_n)$ of either `0` or `1.0` are avoided. Following the calculation of $p(E_n)$, a 2nd float value between `0` and `1.0` is randomly generated as a point of comparison with $p(E_n)$. If $p(E_n)$ is larger than this comparative value (which is more likely if a larger p value is calculated as a result of higher system energy), then a random integer is generated between `0` and `3`, which corresponds to the index of which chime is to be called.

3 Results

Figures 2 and 3 show the original and synthesised versions of chime 1 after it has been reverberating for 0.4 seconds. To the eye and ear the synthesised chime is reminiscent of original chime sample. Both exhibit similar reverberation times and the 'ringing' quality of the chimes is preserved. The generated wind is also convincing: it sweeps and fluctuates in a natural manner and sounds stronger when the system energy is higher. When the wind sounds stronger the chimes trigger more often as hoped, but the interval in which they sound is still unpredictable, as a result of the architecture's layered stochasicity. The chimes are all able to sound in close succession without too much strain on the CPU due to their encapsulations in wavetables, and the assembly language processing of the four filters means that CPU usage is always moderate when the program is running. The only noticeable spike in CPU use is when the program is first run: this is due to the fact that the wavetables containing the sinusoidal modes of the chimes are calculated in `setup()`. An examination of Figures 2 and 3 shows that both display similar deterministic modal behaviour, exhibited by their peaks at around 1000Hz, 2900Hz, 5700Hz and 8400Hz. The stochastic residual noise between the peaks, defined by Serra and Smith [3], appears similarly random between the peaks. The noise floor shown in the synthesised chime shows little voltage variability from from DC to the Nyquist frequency, while the residual noise

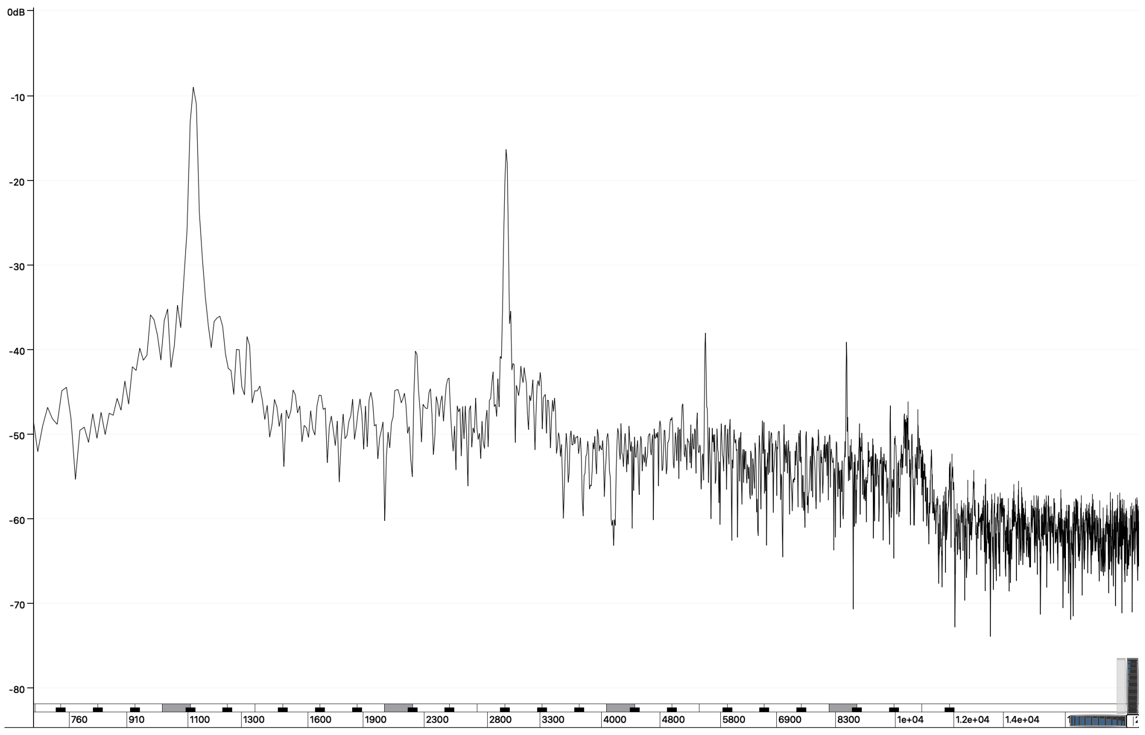

FIGURE 2: Original chime

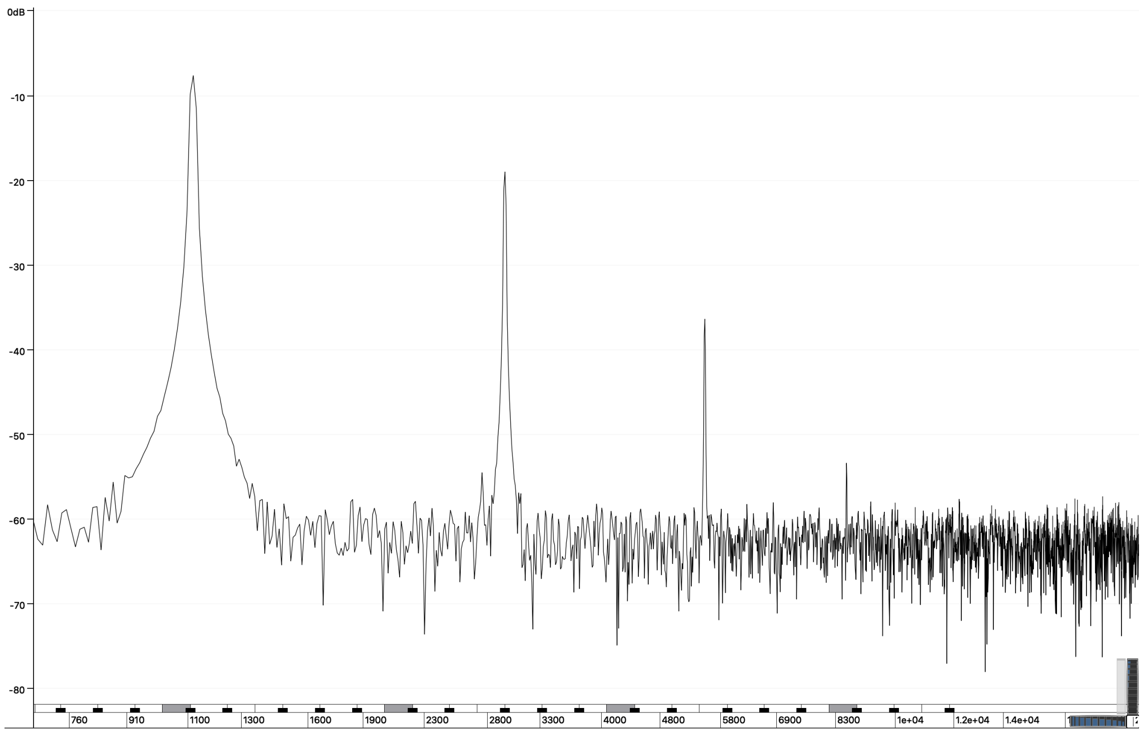

FIGURE 3: Synthesised chime

level of the natural chime decreases throughout before dropping off by about 10dB from 12000Hz upwards. Mode 1 of the original chime also has more noticeable side lobes. Through analysing the lobe dBV relationship in the two figures, the synthesised chime exhibits an evenly sloped shape between the modes, while the natural chime's modes are less even, with modes 3 and 4 being nearly at the same level. Due to the increased mode excitation and larger mode amplitude at higher system energies, the chimes also sound brighter and louder when the wind is stronger. While the sweeping nature of the wind sounds convincing, at times it appears to lack a certain resonance about certain frequencies that could be achieved with a resonant notch filter. Due to the behaviour of a low-pass filter, higher frequencies never appear to be isolated, as when higher frequencies sound all the frequencies below them are also visible.

4 Discussion

Overall I believe this project to be successful in its implementation. The wind chimes sound convincing and react realistically to stronger wind. Whether on manual or automatic mode, the wind sweeps and behaves in an unpredictable way and the chimes react accordingly. Although the wind chime synthesis is convincing and the modes are comprehensively recreated, there is still the stochastic element of the sound, outlined by Serra and Smith, that is not recreated in the current program[3]. In order to accurately model the stochastic element of this sound, one technique Serra and Smith outline is to subtract the spectral content of the synthesised modes from the original spectrum and use the absolute value of the resultant magnitude as an amplitude envelope for filtered white noise. This is a technique that was attempted with this project, however the effect did not function reliably, and due to the fact that a strong filtered white noise source (i.e. the procedurally generated wind) is also present whenever the chimes are sounding, the synthesised stochastic residual noise would have been inaudible. A lack of residual noise generation means that the the residual noise exhibited in Figure 3 is simply a result of the noise floor and the residual noise's amplitude between DC and the Nyquist is unchanging. In order to generate a more accurate chime model, this residual noise must have higher amplitude around modes with higher energy; this kind of behaviour is shown in Figure 2, where the residual noise amplitude is roughly proportional to mode amplitude. When comparing amplitude relationship of the modes between Figure 2 and 3, there is a clear sense of correlation between the two, however they also differ in a couple of obvious ways. Firstly, the synthesised chime's modes' roll-off is precisely evenly sloped, while the original chime is not so. Although I took voltage readings of the exact chimes the chimes in this project are modelled off, I took the T60 decay times from Lukkari and Välimäki's aforementioned conference paper and extrapolated for modes they had not covered by preserving the ratio that was present in their results. This kind of estimation will of course lead to inaccuracies, and the fact that the paper's T60 results were recorded on different wind chimes is a further cause for inaccuracy. The attack time of the chimes is also convincing (see 'Sounds' folder in this project repository for recordings of the real and synthesised chimes), with upper transient modes contributing to the perceived effect of these chimes having just been struck by a clapper in between them. Regardless it is still an assumption that the attack times will not differ between modes, which they invariably will.

In relation to upper transient mode computation, placing the modes in wave tables in `setup()` was an effective way to minimise CPU usage while the program is running, but also leads to a front heavy program that takes a while loads before it is ready to process audio. Implementing part filter process in assembly also greatly improved efficiency of processing, but there is still room for efficiency improvement. It would be worth expanding the assembly language usage to also calculate filter coefficients in parallel, and there will also be ways to streamline the mode calculation process while making maximum use of the NEON unit. Processing the white noise with a variety of filters where resonance can be controlled in different ways would also lead to the system exhibiting the fluidity and dynamism of wind in a more authentic way. Random number generation proved to be more of a difficulty than expected, and there is a scope in this project to make random number generation both more random and efficient. I used Mersenne Twister pseudo-random number generator algorithm, but there are countless others with various strengths and weaknesses that would be worth exploring in future iterations of this project. Expanding on this, the probability behaviour of the chimes could also be refined in such a manner. As the wind generation ultimately hinges on the random number generation constituting the white noise, it also stands to reason that trialing other random number generation techniques could lead to a thicker and more natural wind sound

4.1 Future Work

Aside from the previously mentioned points on random number generation and filter optimisation, in future work I hope to modally synthesise chimes and other sounds with more variability in the amplitude of their modes. Modelling the stochastic element of the chime sound would also help to create a more accurate chime simulation, and this will be the focus of future work. It would also be worth recording T60 times myself for the specific object I am modelling, as this would undeniably lead to more accurate imaging. Furthermore, examining attack times more closely will help improve the behaviour of the both the more resonant modes lower in frequency, and particularly the upper more transient modes. Finally, I plan to overhaul the thread management system, whereby further spectral/ behavioural calculations are carried out while the program is running, leading to enhanced sonic accuracy and program function.

References

[1] 3bagbrew. *barnchime*, 3 May 2010. `https://freesound.org/people/3bagbrew/`.

[2] Teemu Lukkari and V Valimaki. Modal synthesis of wind chime sounds with stochastic event triggering. In *Proceedings of the 6th Nordic Signal Processing Symposium, 2004. NORSIG 2004.*, pages 212–215. IEEE, 2004.

[3] Xavier Serra and Julius Smith. Spectral modeling synthesis: A sound analysis/synthesis system based on a deterministic plus stochastic decomposition. *Computer Music Journal*, 14(4):12–24, 1990.