Real-Time Implementation of PhISEM Percussion Synthesis

Rasmus Nuko Jørgensen
Aalborg University, Copenhagen
rjorge22@student.aau.dk

1. INTRODUCTION

In 1997 Perry Cook published an article called "Physically Informed Sonic Modeling (PhISM): Synthesis of Percussive Sounds" [1]. In this paper, Cook presents two percussive audio synthesis algorithms, namely "PhISAM: Physically Informed Control of Modal Synthesis" and "PhISEM: Physically Informed Stochastic Event Modeling". In this paper we will show a real-time Python [1] implementation of the PhISEM synthesis algorithm. We will go over explain the algorithm presented by Cook, what we have changed in order to make the algorithm work in real-time.

The values for synthesizing different percussion instruments is presented by Cook in the article. We have implemented most of them. All code can be found in this GitHub repository, and should run on all MacOS systems with Python3 and the necessary modules [2] [3] [4] installed.

2. ORIGINAL IMPLEMENTATION

We will explain the original implementation from a physical stand point, aiming to give some understating to how the algorithm works. Then we explain the actual implementation in two steps: Noise generation and filtering.

2.1 Physical model

Since we are looking at a physical modeling algorithm, it makes sense to try to understand the implementation from physical view point. The PhISEM algorithm can synthesize percussive instruments such as maraca, sekere, cabasa, guiro. In essence this is achieved by filtering noise bursts, modeling the excitation (noise) of some body/gourd/bell (resonant band-pass filters). What separates the synthesis of these different instruments is how long these noise bursts last, the amount of exponential decay, what band-pass filters are applied, and how resonant these band-pass filters are. I.e. Sleighbells are higher pitch than maracas and the bells ring out for longer than the wooden gourd of a maraca, therefore the synthesis for sleighbells will have band-pass filters higher in the frequency spectrum that are more resonant than the filters for the maraca.

2.2 Implementation

The code from Perry Cook's article can be seen in Figure 9.

Noise generation

The actual sound being generated is noise, and what is interesting is the envelope for the noise. This envelope can be broken into two parts:

shakeEnergy The variable `shakeEnergy` can be seen as the energy in the instrument as a percussion is shook once. We have illustrated it's value over time in Figure 1. The value of `shakeEnergy` is increased by $1 - cos(X)$ at each sample, where X goes from 0 to 2π over a specified 'shake time'. This shake time is 50ms in Cook's code. There is a exponential decay called `SYSTEM_DECAY` that is multiplied to `shakeEnergy` as each sample.

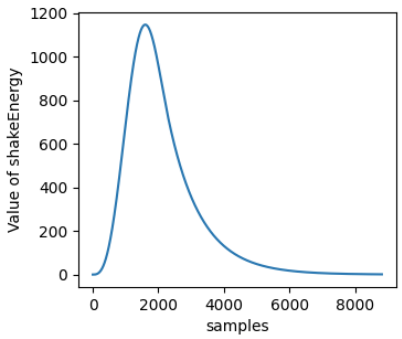

Figure 1. The values of `shakeEnergy` over 250ms, 'shaken', as in Perry Cook's code, for 50ms with `SYSTEM_DECAY` of 0.999

sndLevel The second variable for the envelope is `sndLevel`. At any sample there is a probability that `shakeEnergy` is added to `sndLevel` and at each sample the exponential decay `SOUND_DECAY` is multiplied onto `sndLevel`. We have illustrated a possible outcome for `sndLevel` in Figure 2.

Noise is generated by 'sending noise through the envelope' by calculating:
`input = sndLevel * noise_tick()`, an example output can be seen in Figure 3.

[1] https://python.org/
[2] https://numpy.org/
[3] https://scipy.org/
[4] https://people.csail.mit.edu/hubert/pyaudio/

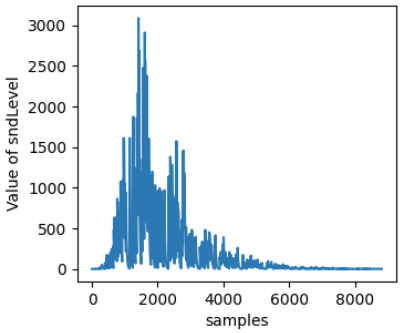

Figure 2. The values of `sndLevel`. Total duration is 250ms. 'Shaken' for 50ms with `SYSTEM_DECAY` of 0.999, and `SYSTEM_DECAY` of 0.95

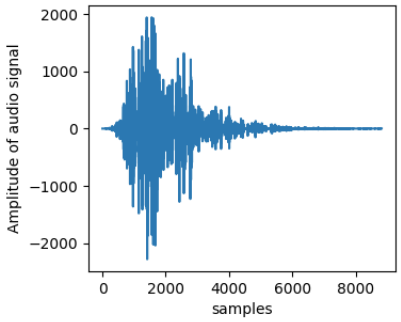

Figure 3. Example of the amplitude of audio signal before filtering. This is the result of running noise through the envelope shown in Figure 2. Total duration is 250ms. 'Shaken' for 50ms with `SYSTEM_DECAY` of 0.999, and `SYSTEM_DECAY` of 0.95

Filtering

The resonant band-pass filter is implemented using two coefficients (`coeffs[0]` and `coeffs[1]`), a 1-sample (`output[0]`), and a 2-sample delay (`output[1]`). The coefficients are calculated from the desired frequency and resonance of the band-filter.

The bass-pass filter is calculated by taking `input` and subtracting `output[0] * coeffs[0]`, and subtracting `output[1] * coeffs[1]`).

See Figues **??** for examples of audio from Figure 3 being run through to different set of filters.

3. REAL-TIME PYTHON IMPLEMENTATION

Our implementation deviates from Cook's original algorithm in some different ways. First and foremost the algorithm runs in real-time.

3.1 Real-Time

Using PyAudio, we get a callback function that allows us to write to a buffer in order to play audio in real-time. Most of the computation for the synthesis is done in the callback function, and is main idea from PhISEM of sending noise through a envelope and filtering it is what is running in the callback function.

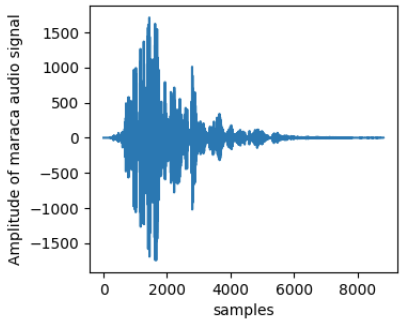

Figure 4. Example of the amplitude of audio signal after filtering. This is the result of running Figure 3 through the `maraca_conf` found in configs.py

.

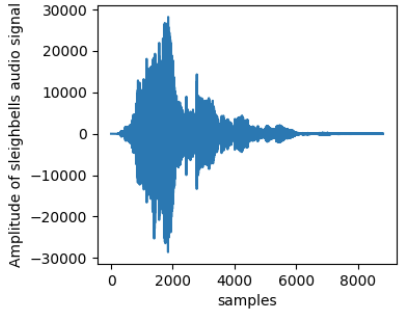

Figure 5. Example of the amplitude of audio signal after filtering. This is the result of running Figure 3 through the `sleigh_conf` found in configs.py

.

A shaker object has been implemented. This object is global in the code, and there can be accessed and stays the same between the different function calls that are being made to the callback function.

The things that are not computed in the callback function is the computation of the coefficients for band-pass filters, and computing how loud a given configuration will get in order to gain it accordingly.

3.2 Changes

We ran into problems using the filter Cook used. Instead we used another resonant filter implementation as shown in [2] [5] . Several of these filters are used in parallel in order to implement some of the instruments shown by Cook in Table 1 from [1].

After gaining a better understanding of how the PhISEM algorithm works, our way of 'predicting' how loud some configuration might get does not make much sense any more, but it gives a rough estimate. After tweaking the estimation, the results are good where the percivable loudness difference between configurations is not large.

Table 1 from [1] where Cook shows values of different parameters to achieve different percussion instruments, he also has "Zeros?" that describes whether there should be a zero at 0 and/or at π or none. We have implemented

[5] The code at the bottom of the link.

the zero at 0 by subtracting a 1-sample delay ($b_1 = -1$ shown in Figure 6) and zero at π by subtracting a 2-sample ($b_2 = -1$ shown in Figure 7). These values were found using ZtransformApplet [6].

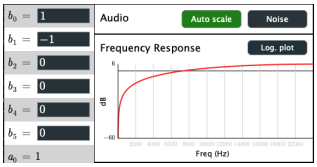

Figure 6. $b_1 = -1$ is a high pass filter as shown in ZtransformApplet

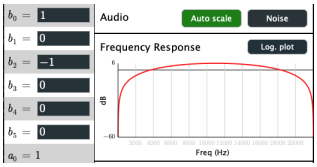

Figure 7. $b_1 = -2$ is a wise band pass filter as shown in ZtransformApplet

3.3 Configurations

Using Python dictionaries, we have an easy way to store the configuration of parameters in the PhISEM algorithm, corresponding to the synthesis of different percussion instruments. We can then load these configurations, to easy switch between instruments. When loading a new configuration, we change the configuration of the global shaker object.

```
maraca_conf = { 'num beans': 32,
                'prob': 32,
                'system decay': 0.999,
                'sound decay': 0.95,
                'zeros': ['zero'],
                'shells': [{'freq': 4200,
                            'q': 0.96}],
                'shake time': 50e-3,
                'name': 'maraca'}
```

Figure 8. Code snippet from configs.py.

4. REFERENCES

[1] P. R. Cook, "Physically informed sonic modeling (phism): Synthesis of percussive sounds." Computer Music Journal, 21(3), 1997. [Online]. Available: https://www.jstor.org/stable/3681012

[2] uh.etle.fni@yfoocs, "Resonant filter." [Online]. Available: https://www.musicdsp.org/en/latest/Filters/29-resonant-filter.html

[6] https://github.com/SilvinWillemsen/ZtransformApplet/

```
#define SOUND_DECAY 0.95
#define SYSTEM_DECAY 0.999
#define SHELL_FREQ 3200.0
#define SHELL_RESO 0.96
#define TWO_PI 6.28318530718
#define SRATE 22050.0

void main(int argc, char *argv[]) {
     FILE *file_out;
     double temp = 0;
     double shakeEnergy = 0.0, sndLevel = 0.0, gain;
     double input = 0.0, output[2] = {0.0,0.0}, coeffs[2];
     long i, num_beans = 64;
     short data;
     extern double noise_tick();    /* -1.0 < this < 1.0*/

     gain = log(num_beans) / log(4.0) * 40.0 / num_beans;
     if (file_out = fopen(argv[1],"wb")"wb")) {
          /* Initialize gourd resonance filter.  */
          coeffs[0] = - SHELL_RESO * 2.0 *
                    cos(SHELL_FREQ * TWO_PI / SRATE);
          coeffs[1] = SHELL_RESO * SHELL_RESO;
          /*  The main loop begins here.  */
          for (i=0; i<100000; i++)       {
               if (temp<TWO_PI) {
                    /* Shake over 50 msec and   */
                    /* add shake energy.        */
                    temp += (TWO_PI / SRATE / 0.05);
                    shakeEnergy += (1.0 - cos(temp));
               }
               /* Shake 4 times/second.  */
               if (i % 5050 == 0) temp = 0;

               /* Compute exponential system decay.  */
               shakeEnergy *= SYSTEM_DECAY;

               if (random(1024) < num_beans)
                    sndLevel += gain * shakeEnergy;
                    /* If collision add energy.        */

               /* Actual sound is random.              */
               input = sndLevel * noise_tick();

               /* Compute exponential sound decay.*/
               sndLevel *= SOUND_DECAY;

               /* Do gourd resonance filter calc.  */
               input -= output[0]*coeffs[0];
               input -= output[1]*coeffs[1];
               output[1] = output[0];
               output[0] = input;

               /* Extra zero for spectral shape.  */
               data = output[0] - output[1];

               /* That's all! Write it out.   */
               fwrite(&data, 2, 1, file_out);
          }
     }
     fclose(file_out);
}
```

Figure 9. PhISEM code from p. 45 [1]