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 PhISAM 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. This paper can also be read as a blog post, where audio snippets of the synthesis can be heard. 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 three steps: Sound generation, filtering, gaining.

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 (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 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 and more resonant filters than the synthesis of the maraca.

2.2 Implementation

3. REAL-TIME PYTHON IMPLEMENTATION

3.1 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.

```
# Maraca
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 1. 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

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

```
#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 2. PhISEM code from p. 45 [1]