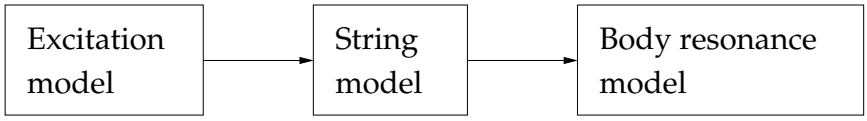
A string excitation model for audio synthesis of bowed string instruments

Volker Schatz http://volkerschatz.com/science/nonpatents/ January 2014

Keywords: musical instrument synthesis, bowed strings, string excitation

Introduction

This invention forms one component of a physical model for synthesizing the sound of a bowed string instrument. Such physical synthesis requires a model of the string excitation, of the string itself and of the instrument body, with the output of each component fed into the next, as displayed in the figure below.



An excitation model describes how the bow drives the oscillation of the string when moved across it. The string model describes the resonance of the string and usually includes a backfeed loop. The simplest string model is that of Karplus and Strong which consists of a feedback delay line with an order-2 lowpass. More complex string models take the two transverse oscillation modes of a string and their interaction at the bridge and at the bow into account. Body resonance models range from reverberation filters to physical models of the instrument body.

Description of the model

This invention was inspired by a model for a block-sliding motion [1]. It is statistically self-similar, accounting for the fact that real-world materials show similar structures at different scales [2]. As described in [1], a block sliding on an inclined surface moves in lurches, with larger lurches being less frequent according to a power law. This principle can be transferred to string excitation by generating a small plucking of the string for each lurch of the bow. Accordingly, this excitation model generates a statistically self-similar pulse train in which the intervals between pulses of a given height are smaller for smaller pulse heights.

The self-similar pulse train is generated by adding up several regular pulse trains whose intervals differ by constant factors and whose amplitudes are related to their intervals by a power law:

$$f(t) = \sum_i \ell^{i\,\gamma}\ \tilde{\delta}(\Delta\ \ell^i;\ t)\,, \tag{1}$$

where Δ is the interval of the largest regular pulse train, $\ell < 1$ the scaling factor by which the intervals of successive component pulse trains differ and $\gamma \geq 0$ the exponent of the power law between interval and height. The function $\tilde{\delta}$ describes a regular pulse train and is defined as follows in a continuous and discretised formulation of the model, respectively:

$$\text{continuous:} \qquad \tilde{\delta}(d;\ t) = \sum_k \delta(t - k\,d) \qquad \text{(Dirac comb)}$$

$$\text{discretised:} \qquad \tilde{\delta}(d;\ t) = \begin{cases} 1 & \text{if } t \in d\,\mathbb{Z} \\ 0 & \text{otherwise} \end{cases}$$

Compared to the strictly self-similar formula (1), the model introduces two kinds of randomness to make the output more natural. The intervals and pulse heights of each regular component pulse train are perturbed independently by a normal-distributed relative variation. Secondly, the component pulse trains are time-shifted randomly relative to each other so that not all first pulses coincide at $t = 0$.

When pulses coincide, their amplitudes are not added, but only the larger pulse is output. This can occur repeatedly as a consequence of rational ℓ.

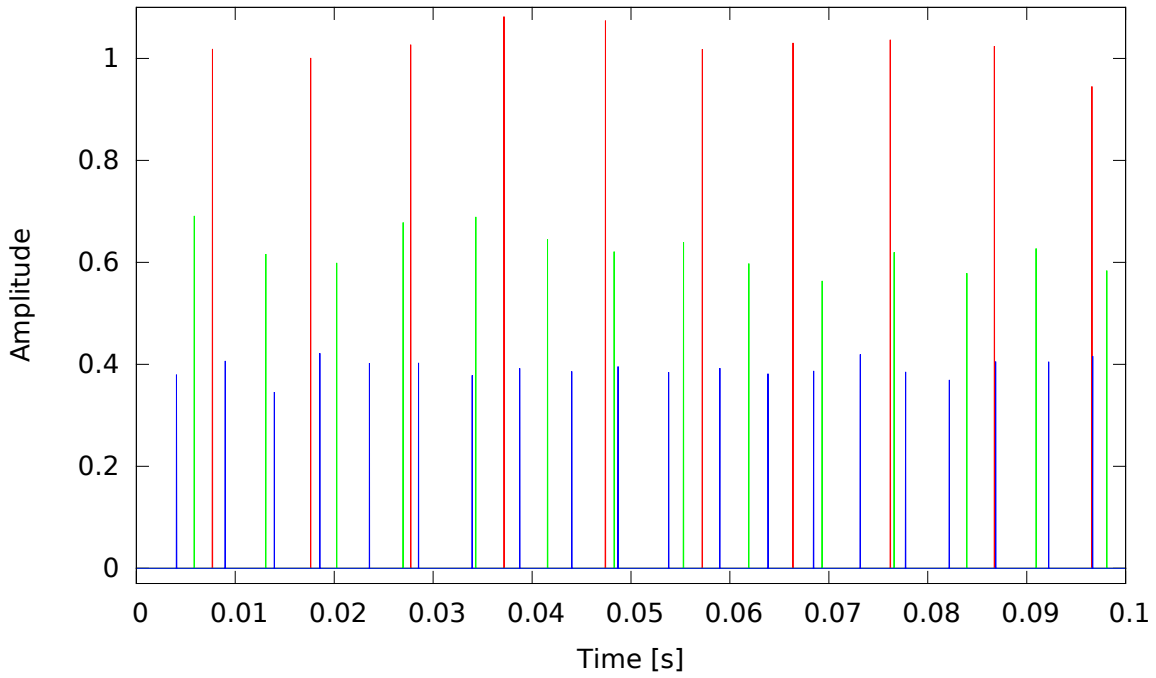
The excitation model has the following parameters:

- The range of intervals of the regular component pulse trains. This determines the largest interval Δ and the maximum i up to which the sum in (1) runs.
- The scaling factor ℓ between the intervals of a component pulse train and the next. This factor greatly influences the overall pulse density as well as the density of the distribution of pulse heights.
- The exponent γ of the power law between the interval and the amplitude of component pulse trains
- The RMS deviation of the relative random variations of the pulse intervals and heights

The figure on the next page displays an example pulse train output by the model. Its parameters are a maximum and minimum interval of 0.01 s and 0.003 s respectively, $\ell = 0.7$, $\gamma = 1.3$, and a statistical perturbation of 0.05.

Example implementation

Page 4 shows an example implementation of the string excitation model as a class in the scripting language Python. The implementation does not assume a sampling rate. Rather, its `output` function receives the time increment to the next sampling point as an argument, and a pulse is output whenever it occurred during the preceding sampling interval.



Example output of the model. The parameters for this example were a maximum interval of $\Delta = 0.01\,\mathrm{s}$, a minimum interval of $0.003\,\mathrm{s}$, a scaling factor $\ell = 0.7$, an exponent $\gamma = 1.3$, and a statistical perturbation of 0.05. The three component pulse trains are coloured differently for illustration purposes.

References

[1] Parteli, Gomes, Montarroyos, Brito: Omori Law for Sliding of Blocks on Inclined Rough Surfaces, `http://arxiv.org/cond-mat/0402577`

[2] Mandelbrot: The Fractal Geometry of Nature, Freeman and Co. 1982, ISBN 0-7167-1186-9

Disclaimer

This disclosure has been submitted to `defensivepublications.org` and is now contained in the IP.com prior art database. However, the author has not performed a search for potential preceding prior art and does not provide any guarantee of freedom from the rights of others. It is up to commercial users to perform all necessary checks before making use of the invention.

```
01: #!/usr/bin/python
02:
03: import math
04: import random
05:
06: class stringexcite:
07:
08:     def __init__(self, maxinter, mininter, scaler, exponent, meandev):
09:         if type(mininter) is not float or type(maxinter) is not float or \
10:             type(scaler) is not float or type(exponent) is not float or \
11:             type(meandev) is not float:
12:             raise AssertionError("All parameters have to be floats")
13:         self.maxi= maxinter
14:         self.scaler= scaler
15:         self.exponent= exponent
16:         self.meandev= meandev
17:         self.levels= int(-math.log(self.maxi/mininter) / math.log(self.scaler))
18:         self.time= 0
19:         self.countdown= []
20:         self.rng= random.Random()
21:         interval= self.maxi
22:         for i in range(self.levels):
23:             self.countdown.append(interval * self.rng.random())
24:             interval *= self.scaler
25:
26:     def output(self, timestep):
27:         eventlevel= -1
28:         interval= self.maxi
29:         for i in range(self.levels):
30:             if self.countdown[i] <= 0:
31:                 if eventlevel < 0:
32:                     eventlevel= i
33:                 self.countdown[i]= interval * self.rng.gauss(1.0, self.meandev)
34:             self.countdown[i] -= timestep
35:             interval *= self.scaler
36:         if eventlevel < 0:
37:             return 0.0
38:         else:
39:             return self.scaler ** (eventlevel * self.exponent) \
40:                    * self.rng.gauss(1.0, self.meandev)
```

Example implementation of the model in Python. The implementation does not explicitly account for the sampling rate; rather, the sampling interval is passed in the `timestep` argument of the `output` method.