Sampling Algorithms for DrumGizmo

André Nusser Bent Bisballe Nyeng

July 21, 2019

Abstract

This paper suggests new sampling algorithms for DrumGizmo. First the requirements and certain problematic special cases are formulated and then several approaches are explored and turned into sample selection algorithms.

1 Introduction

Introduce DrumGizmo. Sample selection is one of the core parts of DrumGizmo as it heavily influences how "robotic" DrumGizmo sounds. What is sample selection (only consider it for one instrument)?

1.1 Terminology

Sample: One hit on the instrument.

Power: Every sample has a certain power which is given by its audio signal .

how exactly is it currently measured?

1.2 Power Value Calculation in DrumGizmo

The power value calculation of DrumGizmo works as follows. Each sample offset is detected by setting a value threshold. If a sample above this threshold is detected the algorithm goes backwards until it finds the first zero-crossing and this point is used as the sample offset.

The power, which is just the power of a signal, is regularly calculated via sum of squares and the length of it is defined by the "attack length" which is in samples (note to self, make this ms instead; also, see samplesorter.cc:96).

The power is then "spread out" by raising the value to the power of "spread", i.e.:

$$\begin{aligned} \textit{power} &= \text{sum of squares in the attack range} \\ \text{stored sample energy} &= \textit{power}^{\textit{spread}} \end{aligned}$$

In the code this is done in samplesorter.cc:99.

We should probably store the original power and attack length in the xml and perform the spread calculation in the engine instead of storing the spread-applied value in the xml.

The attack is the same across all samples within an instrument so the energies can be compared.

1.3 Sample Distribution in DrumGizmo Kits

1.4 Current Algorithm of DrumGizmo

The current sample selection algorithm of DrumGizmo works as follows. The engine gets a value $l \in [0,1]$ which gives the strength of the sample to be triggered. The power values of a drum kit are floating point numbers without any restriction. Then the value l is mapped using the canonical bijections between $[0,1]$ and $[p_{\min}, p_{max}]$ and afterwards shifted. We call this new value p.

do they have to be positive?

by which amount?

Now the real sample selection algorithm starts. We select a value p' drawn uniformly at random from $\mathcal{N}(p', \sigma^2)$, where σ is a parameter specified by the user. Now we simply find the sample s with the power q which is closest to p' – ties are broken such that the first minimal value is chosen (which is problematic as explained below). In case s is equal to the last sample that we played we repeat this process, otherwise we return s. If we did not find another sample than the last played after 4 iterations, we just return the last played sample.

Actually, it is not. It is a value specified by the user *multiplied* with the power range divided by the number of samples.

1.5 Drawbacks

I will list a number of drawbacks of this algorithm in the following. These will be an inspiration for the requirements that are formulated later.

Equal Powers. In case certain samples have the same power value. Always the first of them in the list of samples is chosen because of the way we break ties. This is obviously wrong and samples should instead be chosen either in a random way or, probably better, in a round robin way.

Middle Samples. Consider the case where there are 3 samples which almost have the same power value, and especially consider the sample which has the power value in the middle. This sample has a very low probability of being chosen by the current algorithm as it always chooses the closest sample. However, the range for which this sample is the closest sample is very small. Thus, an improved algorithm has to be robust to such small perturbations of power values.

Unequal Probabilities. More generally, we want that samples which are close, should have a similar probability of being chosen by the sampling algorithm (summed over all possible values of l).

History of Size One. Currently, we only remember the last sample that was played. This seriously limits us to select samples well. Imagine that there are several samples of similar power. We currently rely on the normal distribution solving this, even though using round robin like sampling would result in more diverse samples being chosen while not deviating significantly more from l.

1.6 Related Work

Velocity Layers. Round Robin and Random Selection. Is there actually any academic related work? What is actually the mathematical problem that we are trying to solve?

2 Requirements

Normal Distribution. The samples should roughly be drawn from a normal distribution.

Avoid Same Samples. When we have multiple samples to choose from we should always take one which was last played far enough in the past.

Randomization. To avoid patterns (like e.g. in round robin), we want some form of randomization.

Equal Probability. Locally, samples should have almost the same probability of being chosen.

3 Suggested Algorithms

3.1 Resampling

Resample from normal distribution until we find a fitting sample. This seems rather wasteful.

3.2 Objective Function

Define an objective function which depends on the history of samples and the current power requested – or better, a power which comes from the normal distribution of the power requested. The sample that we choose is then the one which minimizes this objective function. This is a nice way to balance the two contradictory requirements of sampling by normal distribution and avoiding samples that were just played.

The rough algorithm should go as follows. A sample with power l is requested. We draw one sample from the normal distribution around l and call it p. Let t_q be the time at which q was played last (the unit does not matter as it is parametrized by β anyway), and let $r(q,t)$ be a random number generator uniformly producing numbers in the range $[0,1]$. At the current time t, we now want to find the sample q minimizing the objective function

$$f(q,t) \coloneqq \alpha \cdot (p-q)^2 + \beta \cdot (t_q - t)^{-2} + \gamma \cdot r(q,t).$$

We have to ensure that $t_q \neq t$ to avoid division by zero.

4 Implementation Details

Instead of iterating over all samples and computing the objective function, we can simply do a binary search for the value closest to the requested power and then search down and upwards until we can be sure that there cannot be any better value. This is the case as soon as the first summand exceeds the best found value.

5 Experiments

5.1 Methods of Evaluation

- mean squared error to straight line from min power to max power
- histogram of distance to closest next same sample (to check that diverse samples are selected; picture!). Or maybe some other measurement, not sure.
- Histogram of how often samples were played. This should be a uniforum distribution (at least locally). Globally it might diverge from that as the sampling is worse for some powers.
- mean square error to gaussian curve (to check that we still use something similar to a normal distribution; picture!)
- Upload sound samples of the different algorithms to a server and link to them.

5.2 Experimental Evaluation

6 Conclusion

What is the best algorithm and why?