SAMPLING-RATE-AWARE NOISE GENERATION

Henning Thielemann
Institut für Informatik, Martin-Luther-Universität Halle-Wittenberg, Germany
henning.thielemann@informatik.uni-halle.de

ABSTRACT

In this paper we consider the generation of discrete white noise. Despite this seems to be a simple problem, common noise generator implementations do not deliver comparable results at different sampling rates. First we define what we mean with "comparable results". From this we conclude, that the variance of the random variables shall grow proportionally to the sampling rate. Eventually we consider how noise behaves under common signal transformations, such as frequency filters, quantisation and impulse generation and we explore how these signal transformations must be designed in order generate sampling-rate-aware results when applied to white noise.

1. INTRODUCTION

Noise is an ubiquitous kind of signal: Often it is an annoying artefact of signal transmission, conversion, or processing, but it is also an essential part of sounds like wind, breaking water waves, wind instruments, drum sounds, and fricatives in speech. In our paper we explore the generation of the latter kind of noise.

1.1. A motivating example

Imagine a sound designer who works on a collection of synthesised instruments that shall be used in software synthesizers. For instance he tries to match the sound of a panpipe by mixing a sine oscillator and white noise, that is filtered by a resonant low-pass filter as illustrated in Figure 1. Then the sound designer goes to implement the simple flow diagram in the software synthesis package Csound as shown in Figure 2. Since he intends to use the instruments in music for compact discs, he chooses to render his sound at 16 bit resolution and 44100 Hz. He achieves this with the following command line.

```
csound -o panpipe.wav \
   panpipe.orc panpipe.sco \
   --sample-rate=44100 --control-rate=441
```

Our artist will later add an envelope, frequency modulation and other enhancements that make the sound more natural. He will also design several more instruments.

After his collection of instruments has grown to a considerable size he decides to also prepare preview sounds at a lower sampling rate for his web site. To this end he starts Csound with the option `--sample-rate=11025`. To his surprise some instruments sound quite different at the lower rate. He expected a worse quality, but he assumed that instruments would sound essentially the same. He goes back to the simple panpipe prototype algorithm and finds out that the noise portion of the sound is considerably louder at the low sampling rate than at the high sampling rate.

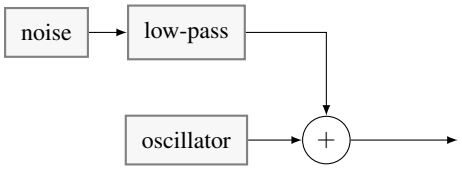

Figure 1: *Flow diagram (abstract signal processing algorithm) for a very simple sound that resembles a panpipe.*

Orchestra file `panpipe.orc`

```
nchnls = 1

instr 1
iamp = 30000
anoise   noise    0.3*iamp, 0
acolored lowpass2 anoise, 440, 10
aosci    oscils   0.5*iamp, 440, 0
out      aosci+acolored
endin
```

Score file `panpipe.sco`

```
i 1 0 2
e
```

Figure 2: *The flow diagram in Figure 1 translated to a Csound program. The line containing `lowpass2` applies a low-pass filter with resonance at 440 Hz to the previously generated white noise. The opcode `oscils` generates a sine wave also with frequency 440 Hz. The score file specifies that our instrument with number 1 starts at second zero and stops after two seconds of playing.*

Thus the carefully chosen mixing ratio of noise and sine wave is lost at the low sampling rate.

Before filing a Csound bug the sound designer wants to find out, what is precisely the problem. He checks that the white noise has the same amplitude at the low and the high sampling rate. The same is true for the sine wave. He replaces the noise generator by an oscillator and observes that filtering a tone yields the same result at both sampling rates. That is, it seems to be the combination of noise and filtering that introduces the unintended volume dependency on the sampling rate. This is very strange.

Now he becomes curious whether that problem also occurs in other sound synthesis systems. He translates his Csound algorithm to the real-time synthesizers SuperCollider and ChucK and confirms that they behave exactly the same way. Finally he gives up and decides to just downsample the sounds he rendered at

44100 Hz. The downsampled wave files actually sound fine.

However our artist is still uncomfortable with the observation that his signal algorithms depend in a non-obvious manner on the sampling rate. So far he thought, that his algorithms abstract from the sampling rate. He had seen his algorithms as an analogy to scalable vector graphic formats, such as PostScript, PDF or SVG that can be rendered at any device at any resolution while achieving the maximum possible quality. He even expected that he could use the same algorithm both for discrete and analogue signal processing. What is the point of sound synthesis compared to sound sampling if not flexible adaptation to changing sound parameters and also to the sampling rate?

He is not quite satisfied with downsampling sounds from 44100 Hz to 11025 Hz in order to get sampled sounds at 11025 Hz. This indirection means that he must invest four times of the computation time of rendering immediately at 11025 Hz plus time for resampling. When rendering music directly at 11025 Hz he could generate 4 times as many channels for spatial effects or 4 times of the polyphony of music rendered via the indirection through 44100 Hz. Vice versa: How can he produce sampled sounds with maximum possible quality at 96000 Hz from his algorithms that he designed for 44100 Hz sampling rate?

1.2. Basic considerations

The noise generators found in Csound and other packages do what certainly everyone would do in order to produce white noise: They run a standard pseudo-random number generator in order to fill an array with random values from the interval $[-1, 1]$ according to a uniform distribution. Now let us see, what this actually means when this is performed at different sampling rates.

In Figure 3 we have white noise of the same duration both at 11025 Hz sampling rate ($\mathrm{render}(11025\ \mathrm{Hz}, \mathrm{noise})$ in the top-left-corner) and at 44100 Hz sampling rate ($\mathrm{render}(44100\ \mathrm{Hz}, \mathrm{noise})$ in the top-right-corner). The noise at the higher sampling rate looks more dense than that at lower rate, of course. We also hear clearly the additional high frequencies in the high rate noise. However we have the impression, that the low frequencies of the noise are louder in the low rate noise and softer in the high rate noise. This auditory impression becomes even visual if we apply frequency filters to this noise. We have applied a first order low-pass filter in the second row of the signal table and a resonant second order low-pass filter in the third row of the table. We clearly see that the signals in the right column have considerably smaller amplitude than those in the left column.

Why do the amplitudes of filtered noise depend on the sampling rate? An intuitive answer can be found in the frequency spectra that are depicted in the bottom row of the table: Since the frequency spectrum of the high rate signal covers a larger frequency range, the energy of the high rate noise is spread over a larger frequency interval.

In order to verify, that the problem is actually the noise and not the filtering, we have inserted a centre column, where all signal processes are performed on a low rate noise, that was converted to a higher rate by simply replicating all sample values of the low rate noise four times. The audio impression is the same as for the sounds in the left column. The important difference between the upsampled noise in the centre column and the high rate noise in the right column is, that the high rate noise consists of *independent* random values whereas the random values in the upsampled noise are *equal* within blocks of four values.

1.3. Contributions

With our paper we want to contribute the following aspects for resolving the sampling rate dependence of white noise:

- Develop a criterion for judging whether a signal generator or modifier performs similarly in different sampling rates in Section 2.1.
- Explore some ways of adapting noise to the sampling rate in Section 2.2.
- Consider several signal modifiers like frequency filters, quantisers, click generators and how they can be made aware of noise input and sampling rates in Section 2.3.
- Discuss in Section 2.4 by what parameters a sampling-rate-aware noise generator should be controlled.
- Give a small guide on choosing a random distribution in Section 2.5.

2. MAIN WORK

2.1. Comparability across sampling rates

In natural sounds there is no such thing as a time quantisation and a sampling rate. Thus natural signals are commonly modelled by real functions. But when it comes to signal processing in a digital computer we need time (and value) discretisation. Nonetheless we do not want to think about discretisation when designing a signal processing algorithm. We like to pretend that there is no sampling and thus an algorithm without a reference to sampling can be used both for analogue synthesis and for digital synthesis at any sampling rate.

1 Definition (Abstract signal processing algorithm)**.** We like to call a signal processing algorithm *abstract*, if it does not contain any reference to discretisation or a sampling rate. All quantities in such an algorithm shall be physically meaningful, e.g. time values must be given in seconds but not as numbers of sampling periods. An example is the Csound algorithm in Figure 2.

Since an abstract signal processing algorithm neglects sampling, we can use it to describe real functions. Interpreting an abstract signal processing algorithm at a given sampling rate means, that we approximate these real functions by discrete signals. E.g. if we describe a frequency filter as the solution of a differential equation, then this is an abstract algorithm. In the digital computer we compute a corresponding difference equation and this is the interpretation of the abstract algorithm for a given sampling rate. We measure the quality of the difference equation solver by its closeness to the solution of the according differential equation.

For investigation of noise, real functions are not of much use as a model, since real functions with stochastic values are neither continuous nor integrable. In contrast to that, there is no problem in computing differences or sums in discrete noise. We may be able to model noise using stochastic processes, stochastic differential equations and generalised measurements of the degree of approximation between a discrete signal and stochastic function. But we think that the following approach is easier:

We accept the lack of a discretisation-free model that we can adapt our discrete computations to. Instead we ask for comparable results, when interpreting the same abstract signal processing algorithm for different sampling rates. That is, increasing the sampling rate for rendering shall improve the audio quality but it shall not alter the timbre of the sound signal.

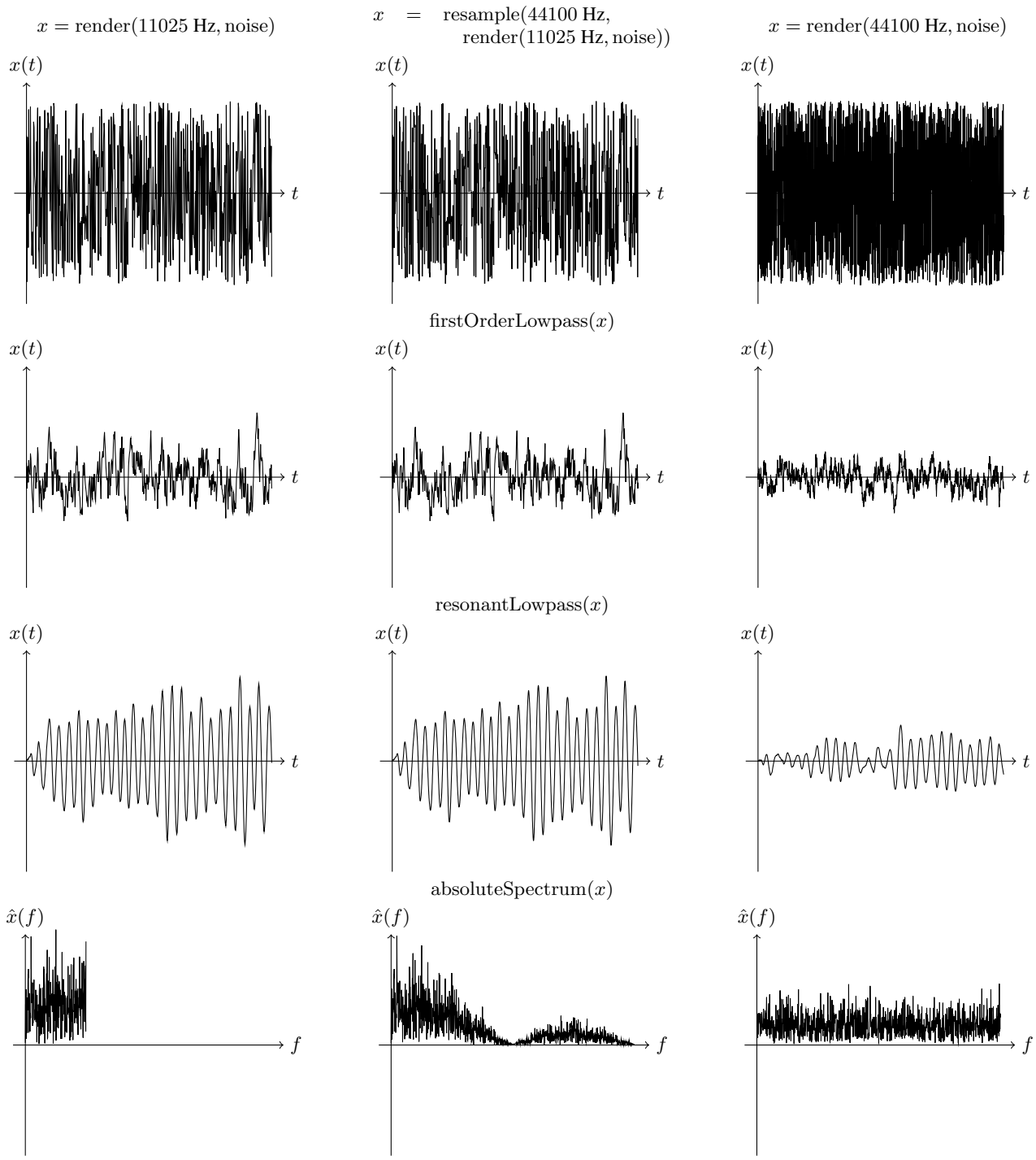

Figure 3: *Table of three signals and the result of various transformations applied to them. The three initial signals are noise at sampling rate 11025 Hz, noise at 11025 Hz upsampled to 44100 Hz by constant interpolation, noise at 44100 Hz. The duration of the sounds is 50 ms. The initial signals are depicted in the first row of the table. The row below contains the results of applying a first order low-pass filter with cut-off frequency of 500 Hz. The third row contains the results of a resonant low-pass filter from a state-variable filter with resonance frequency 500 Hz. The last row contains frequency spectra of the initial sounds.*

How can we check, whether a particular discrete interpretation of an abstract signal processor generates comparable results for different sampling rates? We have to convert between the sampling rates. We cannot add information by upsampling a signal from a low sampling rate, but we can discard information by downsampling a signal from a high to a low sampling rate. Downsampling should act as a projection: It shall maintain the information, that can be represented at the lower rate and it shall discard the remaining information.

In order to write the comparability requirement a bit more formally, we like to define $\mathrm{render}(r, A)$, that denotes a discrete signal, that is computed from the abstract algorithm A at sampling rate r. Think of A being the Csound orchestra definition in Figure 2, r being the number we pass to the `--sample-rate` option and render as being the `csound` command. The sampling rate r becomes part of the generated signal, such as it becomes part of the `WAVE` file generated by `csound`. Further on we like to denote the resampling of a signal x from its associated sampling rate to another sampling rate r by $\mathrm{resample}(r, x)$. Now we can state:

2 Criterion (comparability across sampling rates)**.** The discrete interpretation (expressed by render) of an abstract signal processing algorithm A is called *comparable across sampling rates* if

$$\forall r_0 \forall r_1 \quad r_0 \le r_1 \Rightarrow \\ \mathrm{render}(r_0, A) \approx \mathrm{resample}(r_0, \mathrm{render}(r_1, A))$$

If $\mathrm{render}(r_0, A)$ computes a band-limited version of $\mathrm{render}(r_1, A)$ and resample performs perfect resampling, then "$\approx$" could be replaced by "$=$". However most actual implementations of discrete signal processing only approximate this ideal world. For noise it is even worse, since we can hardly create "the same" noise at different sampling rates. Thus we have to interpret "$\approx$" even weaker as an equivalence of some stochastic characteristics. Although the above criterion is in no way mathematically precise, it turns out to be a very useful guide for design decisions in the following sections.

2.2. Adapt noise to sampling rate

A simple way to provide noise that behaves similar across different sampling rates, is to upsample noise from a low rate. Say, we are satisfied with the range of frequencies contained in discrete noise at 11025 Hz sampling rate. For sounds at 44100 Hz we can just upsample that noise from 11025 Hz to 44100 Hz. This approach trivially generates comparable noise signals for all sampling rates above 11025 Hz, even with "$\approx$" replaced by "$=$" in Criterion 2 when we use pseudo-random numbers with the same seed for all sampling rates. But there are two disadvantages:

- This way we cannot generate comparable noise for rates below 11025 Hz.
- We want to increase rendering quality by increasing the sampling rate. For noise, we like to read "higher quality" to mean a larger range of random frequencies. However, with the upsampling approach we do not automatically get higher frequency portions in the noise, when we switch to higher sampling rates.

If we generate pseudo-random numbers at the target sampling rate, then we automatically fill the entire available frequency space. However, as we have seen in the introduction, we have to somehow adapt the noise amplitude in order to provide equal frequency amplitudes. We still have to live with the drawback, that this kind of sampling-rate-aware noise is comparable across sampling rates only with respect to stochastic parameters but not in terms of actual approximations.

In the following sections we will derive the necessary amplitude adjustment and we will see how other signal processes must be adapted in order to work nicely with noise.

2.3. How to further process noise

2.3.1. Frequency Filter

An important way of modifying white noise is frequency filtering. In analogy to electromagnetic oscillations of light, filtered noise is called coloured noise. Pink noise, i.e. low-pass filtered noise, can be used as control curve. White noise filtered by resonant low-pass filters can produce sounds of wind, echo sounding, or fricatives.

We want to investigate how to adapt noise to sampling rates such that it behaves similar with respect to frequency filters. That is, according to Criterion 2 we want to achieve

$$\forall r_0 \forall r_1 \quad r_0 \le r_1 \Rightarrow \\ \mathrm{render}(r_0, \mathrm{filter}(\mathrm{noise})) \\ \approx \mathrm{resample}(r_0, \mathrm{render}(r_1, \mathrm{filter}(\mathrm{noise}))) \quad . \quad (1)$$

Let us start with the simple example of a moving average filter, where the arithmetic mean of w successive values is computed. We model white noise as a sequence of random variables, that all have the expectation value 0 and the same variance. The technical term for such a sequence is *discrete stochastic process*. The expectation value corresponds to the direct current offset, whereas the standard deviation (root of the variance) is the measure of the noise volume. We start with filtering white noise X_{low} at sampling rate 11025 Hz.

$$\mathrm{render}(11025\ \mathrm{Hz}, \mathrm{filter}(\mathrm{noise})) : \\ Y_{\text{low},k} = \frac{1}{w} \cdot \sum_{j=k}^{k+w-1} X_{\text{low},j}$$

In order to perform the same filter at the higher sampling rate 44100 Hz and an according white noise X_{high}, we have to increase the number of averaged values to $4w$.

$$\mathrm{render}(44100\ \mathrm{Hz}, \mathrm{filter}(\mathrm{noise})) : \\ Y_{\text{high},k} = \frac{1}{4w} \cdot \sum_{j=k}^{k+4w-1} X_{\text{high},j}$$

We observe that

$$\sigma(Y_{\text{low},0}) = \frac{1}{w} \cdot \sigma(X_{\text{low},0}) \\ \sigma(Y_{\text{high},0}) = \frac{1}{2 \cdot w} \cdot \sigma(X_{\text{high},0})$$

that is, for equal standard deviations of the white noises the standard deviations of the filtered noises are not equal. From $\sigma(X_{\text{low},0}) = \sigma(X_{\text{high},0})$ it follows $\sigma(Y_{\text{low},0}) = 2 \cdot \sigma(Y_{\text{high},0})$.

How to resolve this inconsistency? For the filtered noise the resample operation in (1) is essentially a matter of keeping every fourth value. That is we can require $Y_{\text{low},k} \approx Y_{\text{high},4k}$. We like to read this as

$$\forall k \quad \sigma(Y_{\text{low},k}) = \sigma(Y_{\text{high},4k})$$

To achieve this, we have to set the standard deviation of X_k proportional to the square root of the sampling rate. Note, that downsampling of white noise cannot be done simply by picking values at a coarser grid, since this skips the necessary limitation of the frequency band.

$$\sigma^2(X_k) \sim r$$

Now we move on to general frequency filters. They become most simple in the frequency spectrum (just a weighting of the spectral values) and also downsampling is only matter of shortening the spectrum. Thus we like to interpret "$\approx$" in (1) as comparing the frequency spectra.

Let X be a sequence of n random variables, that all have the expectation value 0 and the variance y^2. The noise sampling rate is r and it may have a physical unit such as Hz. The discrete frequency spectrum $\mathrm{DFT}^{-1}(X)$ is defined by

$$\mathrm{DFT}^{-1}(X)_k = \frac{1}{r} \cdot \sum_{j=0}^{n-1} X_j \cdot \exp\left(2\pi i \cdot \frac{j \cdot k}{n}\right) \quad .$$

We have to interpret "$\approx$" in (1) as the equality of the standard deviations of the FOURIER coefficients, because we cannot expect similarity of observed frequency amplitudes. Since the random variables in X are independent, their variance is additive.

$$\begin{aligned} \sigma^2(\mathrm{DFT}^{-1}(X)_k) &= \frac{1}{r^2} \cdot \sum_{j=0}^{n-1} \sigma^2(X_j) \\ &= n \cdot \frac{y^2}{r^2} \\ \sigma(\mathrm{DFT}^{-1}(X)_k) &= \sqrt{n} \cdot \frac{y}{r} \end{aligned} \tag{2}$$

Since n depends on the sampling-rate via $n = l \cdot r$, and we must compare signals of the same length l, we have to substitute n.

$$\sigma(\mathrm{DFT}^{-1}(X)_k) = \sqrt{\frac{l}{r}} \cdot y$$

That is, if noise of duration l at sampling rate r shall have spectral values with standard deviation c (i.e. $c = \sigma(\mathrm{DFT}^{-1}(X)_k)$), then we have to choose

$$y = \sqrt{\frac{r}{l}} \cdot c \quad . \tag{3}$$

The amplitude of the noise is proportional to the square root of the sampling rate.

The part in (3) that does not depend on the sampling rate is $\frac{c}{\sqrt{l}}$. We like to call that the *noise voltage spectral density value*. Usually spectral density is a function defined for real signals. In the following definitions we want to adapt the required terms from real signals to discrete ones.

3 Definition (Wide-sense stationary discrete stochastic process)**.** A discrete stochastic (or random) process X, where all elements have expected value 0 is called *stationary in a wide sense* if the covariance between its elements depends only on the distance but not on the time point.

Expressed in formulas:

$$\begin{aligned} \forall k \qquad & \mathbb{E}(X_k) = 0 \\ \forall k \forall d \quad & \mathbb{E}(X_0 \cdot X_d) = \mathbb{E}(X_k \cdot X_{k+d}) \end{aligned}$$

The white noise signals, that we consider in this paper, and also filtered white noise signals are always discrete stochastic processes in a wide sense.

4 Definition (Autocovariance function)**.** For a wide-sense discrete stochastic process X we define the *autocovariance* function R_X (often called *autocorrelation*) as the covariances between signal values depending on their distance.

$$R_X(d) = \mathbb{E}(X_0 \cdot X_d)$$

This captures all possible values of covariances between signal values, because the wide-sense stationarity warrants time-invariance of the covariances.

5 Definition (Noise power spectral density)**.** The noise spectral density of a wide-sense discrete stochastic process X is the spectrum of the autocovariance function of X.

$$\mathrm{NSD}(X) = \mathrm{DFT}^{-1}(R_X)$$

Since the signal values of white noise are independent, the autocovariance function is an impulse at time point zero with height $\sigma^2(X_0)$. Its spectrum is a constant function with value $\frac{\sigma^2(X_0)}{r}$. According to (2) that is equal to $\frac{c^2}{l}$. We like to call this value the *noise power spectral density value of white noise*. The voltage spectral density is the square root of the power spectral density. We want to use this as the parameter, that describes the amplitude of white noise in a sampling-rate-aware way, and thus give it a symbol, namely VSD.

$$\mathrm{VSD} = \frac{\sigma(X_0)}{\sqrt{r}} \tag{4}$$

2.3.2. Quantisation

Quantising noise in time direction is a way, to give noise a pitch characteristic, when using small quantisation periods, and is useful as control curve for large quantisation periods.

For reasons of simplicity we will consider quantisation with fixed quantisation periods that are integral multiples of the sampling period. For a discrete input signal x with sampling rate r and quantisation period t, and d being the quantisation period measured in units of the sampling period, that is $d = t \cdot r, d \in \mathbb{N}$, we could simply define

$$\mathrm{quantise}(x)_k = x_{k-(k \bmod d)} \quad . \tag{5}$$

This would yield a constant amplitude of $\mathrm{quantise}(x)$ for constant quantisation period t and varying sampling rate r if the amplitude (standard deviation) of x would not depend on r. However if we quantise sampling-rate-aware noise as described in Section 2.3.1 this way, then the amplitude of the quantised noise with respect to a constant quantisation period will increase proportional to the square root of the sampling rate. In this respect an amplitude that increases with the sampling rate is not good, since the quantisation period acts like an artificial low sampling rate represented at a high sampling rate, and this quantisation period does not depend on the actual sampling rate.

We can avoid a growing amplitude by averaging over the quantisation period.

$$\begin{aligned} \mathrm{quantise}(x)_\kappa &= q_{\lfloor \kappa/d \rfloor} \\ \text{with } q_k &= \frac{1}{d} \cdot \sum_{j=k \cdot d}^{(k+1) \cdot d - 1} x_j \end{aligned} \tag{6}$$

In the following proof we show that the amplitude of quantised sampling-rate-aware white noise X with spectral density as in (4) does not depend on the sampling rate. That is, we check the criterion in Criterion 2 where we interpret "$\approx$" as comparing the standard deviation (= the amplitude) of the quantised noise.

$$\begin{aligned} Q_k &= \frac{1}{d} \cdot \sum_{j=k\cdot d}^{(k+1)\cdot d-1} X_j \\ \sigma^2\left(Q_k\right) &= \frac{d}{d^2} \cdot \sigma^2(X_{k\cdot d}) \;= \frac{1}{t\cdot r} \cdot r \cdot \mathrm{VSD}^2 \\ \sigma\left(Q_k\right) &= \frac{\mathrm{VSD}}{\sqrt{t}} \end{aligned}$$

We see that the amplitude of the quantised noise grows proportionally to the square root of the quantisation frequency $\frac{1}{t}$. This is compliant with our sampling-rate-aware white noise generation, where we want that noise with more frequency content is also louder. In fact quantisation can be seen as downsampling, that includes an appropriate low-pass filter, with subsequent upsampling by constant interpolation.

A disadvantage of averaging quantisation as in (6) is that in real-time processing it delays the signal by one quantisation period t, whereas the simple quantisation is in (5) does not cause such a delay.

2.3.3. Random clicks (impulse noise)

There is another important kind of sounds that is based on randomness: Randomly occurring impulses. By subsequent processes like frequency filters we can change the characteristic to several natural sounds. Examples are the sound of raindrops, hail, or the GEIGER-MÜLLER-counter for measurement of ionising radiation.

A simple approach to generate random impulses from white noise is as follows: From white noise at sampling rate r with samples that are uniformly distributed between $-y$ and y we want to obtain random impulses with a frequency f. We generate an impulse in the output signal whenever the white noise sample is in the interval $\left[-\frac{yf}{r}, \frac{yf}{r}\right]$.

This approach is very simple but it has several drawbacks:

- It is bound to input noise with uniformly distributed sample values.
- The threshold value $\frac{yf}{r}$ for given frequency f depends on the sampling rate r. If the input white noise is sampling-rate-aware and uniformly distributed, then its sample values cover $[-k\sqrt{r}, k\sqrt{r}]$ (i.e. $y = k\sqrt{r}$) for sampling-rate independent k and the threshold must be $\frac{kf}{\sqrt{r}}$.
- For smooth input signals (no noise) we get clusters of impulses, what in discrete signal processing means, that we get signals consisting of constant pieces rather than separated impulses.
- We can only control the overall frequency of impulses but not the degree of randomness.

All of these problems can be solved using $\Delta\Sigma$-modulation as in Figure 4. We integrate white noise with a positive direct current offset until it exceeds a threshold. At this time point we emit an impulse in the output signal and then start integrating with cleared accumulator again. We repeat this procedure in an endless loop.

In the discrete implementation of the $\Delta\Sigma$-converter the integrator $\boxed{\int}$ is a cumulative sum, the comparator $\boxed{> y}$ emits an impulse with a height related to y, if the input exceeds the threshold y

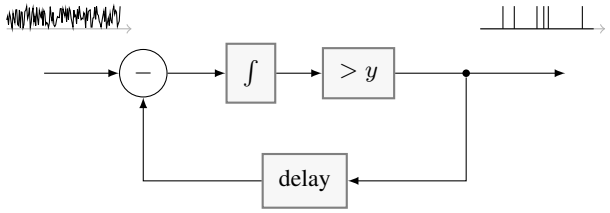

Figure 4: $\Delta\Sigma$*-modulation.*

and zero otherwise, the $\boxed{\text{delay}}$ delays by one sampling period in order to make the feedback possible, and the $\ominus$ subtracts the feedback impulse signal from the input, such that the integrator is reset after every emitted impulse.

The expectation value of the input white noise, i.e. the direct current offset, determines the frequency of peaks in the output, whereas the variance of the noise determines the degree of randomness of peak distribution.

We want to prove, that impulse generation from white noise via $\Delta\Sigma$-modulation yields comparable frequency and randomness of impulses across sampling rates. The noise generation and the integration are the only operations that adapt to the sampling rate. That is, it suffices to show that the integral over a fixed duration t of a sequence X of identically distributed random variables with standard deviations as in (4) has an expected value and a variance that does not depend on the sampling rate r. For simplicity $t \cdot r$ shall be an integer.

$$\begin{aligned} \int_0^t X &= \frac{1}{r} \cdot \sum_{k=0}^{t\cdot r-1} X_k \\ \mathbb{E}\left(\int_0^t X\right) &= t \cdot \mathbb{E}(X_k) \\ \sigma^2\left(\int_0^t X\right) &= \frac{t\cdot r}{r^2} \cdot r \cdot \mathrm{VSD}^2 \\ \sigma\left(\int_0^t X\right) &= \sqrt{t} \cdot \mathrm{VSD} \end{aligned}$$

We have still not answered the question, what kind of impulses the $\Delta\Sigma$-modulator shall generate. If we use impulses with one sampling period as duration, then the height of the impulses must be chosen, such that when fed back it resets the accumulator in the integrator. To this end let us consider the involved physical units: Let the input signal have time unit s and amplitude unit V. Then the integrated signal has amplitude unit Vs and so the threshold in the comparator must have this unit, too. Thus the impulse, that the comparator generates, must have an area equal to the threshold y, in order to clear the accumulator. Since its width is $\frac{1}{r}$, its height must be $r \cdot y$.

This way the impulses have sizes such that they represent the area of the input signal over the pauses between the impulses. This means, that smoothing the impulse train yields a signal similar to the input signal after smoothing. This property is actually the key for using $\Delta\Sigma$-modulator in digital-analogue converters.

An alternative approach for generating random impulses is to choose the pauses between the impulses according to pseudo-random numbers with expectation value "average silence duration between impulses" and variance "degree of randomness". That is, strictly spoken it is not necessary to generate random impulses

from white noise. However, on the one hand we wanted to show, that random impulse generation from sampling-rate-aware white noise is possible in a way, that is itself sampling-rate-aware. On the other hand we wanted to point out that the use of a comparator (as in the beginning of this section) can lead to signal algorithms, that depend on the sampling rate by accident.

2.4. Noise parameters

In principle a sampling-rate-aware white noise generator with amplitude unit V and time unit s must be controlled by a parameter with unit $\mathrm{V}/\sqrt{\mathrm{Hz}}$ or $\mathrm{V}\cdot\sqrt{\mathrm{s}}$, that we called *voltage spectral density value* VSD (see (4)). However that is both unintuitive and unsupported by the usual implementations of physical dimensions in programming languages ([1, 2]) where exponents of units must be integers. It is unintuitive, because it is not simple to choose a number that yields a reasonable amplitude. E.g. we must choose $\mathrm{VSD} = \frac{1\ \mathrm{V}}{\sqrt{44100\ \mathrm{Hz}}} \approx 4.67\ \mathrm{mV}\cdot\sqrt{\mathrm{s}}$ in order to get standard deviation 1 V when rendering at sampling rate 44100 Hz. We avoid the factional powers in units using the squared parameter, that is the *power spectral density value* with unit $\mathrm{V}^2\cdot\mathrm{s}$. This is even less intuitive, since doubling the noise amplitude means using four times of the power density value. In our experience a very intuitive solution is to use two parameters y and f with the units V and Hz. They mean that at sampling rate f the variance shall be y and the variance for other sampling rates shall be adjusted accordingly. Given these parameters the noise generator must compute samples of random variables X with

$$\mathbb{E}(X) = 0 \qquad \sigma(X) = y\cdot\sqrt{\frac{r}{f}} \quad .$$

2.5. Random distribution

So far we did not need to consider particular random distributions, because we only needed additivity of the variance of random variables. The choice of the random distribution does not have an effect on the shape of the frequency spectrum. If the random variables of a noise signal are independent from each other, then all spectral values have the same variance. However, since the human ear performs something more like a short-time FOURIER transform, a random distribution considerably different from normal distribution may generate single clicks, that can be heard.

Because in signal processing many operations like frequency filtering, integration, mixing involve addition, it is likely that the Central Limit Theorem applies. The result of applying signal algorithms to white noise are likely to yield random variables with random distributions close to normal distribution. For reasons of consistency we may thus prefer normal distributions from the beginning. A very simple way to approximate normally distributed random variables with variance 1 is to add three uniformly $[-1, 1]$-distributed random variables. The actual distribution has the shape of a quadratic B-spline. If speed matters, then white noise with uniformly distributed random variables is the best choice. This is what pseudo-random number generators create.

3. RELATED WORK

A wide range of the literature considers noise that arises as an undesirable artefact of signal processing. This part of the literature identifies properties that allows to compare the behaviour of electronic circuits with respect to noise and to separate noise and non-noise portions of a signal. In this literature the notion of the noise spectral density is well-known. [3, Chapter 2]

Intended generation of noise is not equally popular. A notable exception is [4], where the authors construct noise by mixing sine waves at random frequencies. Consequently they use a custom definition of noise spectral density, where the density is the number of sine waves divided by the width of the frequency band. By using the same ratio of present frequencies per band across sampling rates they can create sampling-rate-aware noise in a trivial way. The sinusoidal model allows to control the noise colour in an intuitive way, but for white noise, it is computationally more intensive, even when using a Fast Fourier Transform, than our approach of just adapting the amplitude of a random-number sequence.

Although the notion of the noise spectral density is well-known in the literature, we could not find the conclusion, that discrete white noise should be generated with a variance proportional to the sampling rate. As mentioned in the introduction it is also not implemented in common software synthesizers. We have tested Csound-5.10.1 [5], SuperCollider-3.3.1 [6], ChucK-1.2.0.8 [7]. None of these packages advertises to be sample-rate-aware, although the use of physically motivated parameters suggest that they are. However physical parameters such as time in seconds and frequency in Hertz are mixed with low-level parameters like plain digital filter parameters, e.g. Csound:`noise`:*kbeta*, SuperCollider:`OnePole`:*coef*, ChucK:`BiQuad`:*a0*.

In Csound the `noise` opcode with disabled smoothing (parameter *kbeta*=0) generates white noise. It does not adapt its amplitude to the sampling rate. This applies to all other of Csound's white noise generators, that provide different distributions of the random variables (opcodes `gauss`, `unirand`, `linrand`, `cauchy`, ...). It also applies to the white noise generators in SuperCollider (`WhiteNoise`) and ChucK (`Noise`).

If we want to get coloured noise, we can call specialised noise generators in those packages. E.g. there is the Csound opcode `pinkish` in the default mode and the SuperCollider unit generator `PinkNoise`, that generate pink noise following a multi-scale scheme (MOORE/VOSS-MCCARTNEY method). Although not strictly comparable across different sampling rates, because at lower sampling rates there are more low-frequency components, the generated noise is almost comparable across different sampling rates.

To our surprise we have not found time quantisation in Csound, SuperCollider and ChucK. Thus these packages have no problem in combining a noise generator with a quantisation. However Csound:`randh`, SuperCollider:`LFNoise0` and ChucK:`SubNoise` provide quantised noise at a given rate. By design these produce comparable results across different sampling rates.

We could also not find delta-sigma modulation in Csound, SuperCollider and ChucK. Since SuperCollider does not allow short-time feedback, delta-sigma modulation cannot be build from other components. Nonetheless generation of random impulses is possible with Csound:`mpulse` with `rand` input and SuperCollider:`Dust`. The frequency of impulses is sample-rate-aware, but the area of generated impulses varies across sampling rates.

There are also software synthesizers like Timidity and FluidSynth that are designed for SoundFonts. SoundFont is a format for archiving sampled sounds together with loop points, envelopes and post-processing features like frequency filters. Since SoundFont-2

Orchestra file `panpipe-adapt.orc`

```
nchnls = 1

instr 1
iamp = 30000
anoise   noise    0.3*iamp*sqrt(sr/44100), 0
acolored lowpass2 anoise, 440, 10
aosci    oscils   0.5*iamp, 440, 0
out      aosci+acolored
endin
```

Figure 5: *Modified version of the Csound program in Figure 2 that automatically adapts to the sampling rate.*

does not seem to support a noise generator, noise must be provided as a sampled sound. If resampling is implemented properly (i.e. including band-limitation), then noise is automatically adapted to the sampling rate. Of course the range of frequencies contained in noise can never be larger than the noise contained in the stored sampled sound.

Digital hardware synthesizers have a fixed sampling rate and thus they do not need to adapt to different sampling rates. The same applies to analogous synthesizers that do not have a sampling rate at all.

In [8] the authors describe an electronic circuit called "Noise Manipulator" that creates random impulses from white noise. It uses the following signal algorithm:

$$\mathrm{monoflop}(\mathrm{comparator}(y, \mathrm{pinknoise}))$$

This means: Pink noise is converted to a rectangular signal using a comparator with threshold y. Then the monoflop converts low→high jumps to impulses. The frequency and randomness of the impulses depend on the threshold y and the spectrum of the pink noise in a non-obvious way. In analogue signal processing there is no infinitesimally short DIRAC impulse and no sampling period. That is we must explicitly assign a duration and a height to impulses. It is certainly worth to also generate impulses with a definite duration in discrete signal processing.

4. CONCLUSIONS AND FUTURE WORK

In our paper we found that it is useful to adapt the amplitude of white noise proportionally to the square root of the sampling rate in order to achieve a consistent audio impression across different sampling rates. To speak in terms of the sound designer in the introduction (Section 1): He extends the amplitude parameters of all of his noise generators by the factor $\sqrt{\frac{r}{44100\ \mathrm{Hz}}}$ as in Figure 5 where r is the sampling rate. This way the noise amplitudes at 44100 Hz remain as he found them in the course of developing the signal algorithms. For other sampling rates the amplitude grows proportionally to the square root of the sampling rate.

This solves the problem for the sound designer. However the signal algorithm now contains a reference to the sampling rate. That is, according to Definition 1 it is no longer abstract. E.g. it would not be possible to translate the algorithm and its parameters to an analogue synthesizer. To this end we would need a noise generator with the noise voltage spectral density as parameter as in Section 2.4.

Additionally we have checked, that further processing steps of the noise like filtering, quantisation and impulse generation, can maintain the auditory experience across sampling rates when implemented properly.

As seen in section Section 2.3.3 it is still possible to accidentally develop signal algorithms that produce considerably different results across sampling rates. Especially non-linear operations like comparators are problematic. We have to further investigate how to reduce that risk while remaining able to implement all interesting signal processing algorithms.

5. ACKNOWLEDGMENTS

Many thanks go to my colleague Alexander Hinneburg for proof reading and discussing the draft of the paper.

6. REFERENCES

[1] Don Syme, Adam Granicz, and Antonio Cisternino, *Expert F#*, Apress, 2007.

[2] Bjorn Buckwalter, "dimensional: Statically checked physical dimensions.," `http://hackage.haskell.org/package/dimensional-0.8.0.1`, June 2010.

[3] Walt Kester, Ed., *The Data Conversion Handbook*, Analog Devices Inc. Newnes, 3rd edition, 2005.

[4] Pierre Hanna, Anthony Beurivé, and Myriam Desainte-Catherine, "Real-time noise synthesis with control of the spectral density," in *Proceedings of the 5th Conference on Digital Audio Effects (DAFX'02)*, 2002, pp. 151–156.

[5] Barry Vercoe, "CSound," `http://www.csounds.com/`, 2009.

[6] James McCartney, "Super Collider," `http://www.audiosynth.com/`, March 1996.

[7] Ge Wang and Perry Cook, "Chuck: a programming language for on-the-fly, real-time audio synthesis and multimedia," in *MULTIMEDIA '04: Proceedings of the 12th annual ACM international conference on Multimedia*, New York, NY, USA, 2004, pp. 812–815, ACM.

[8] Hans-Jochen Schulze and Georg Engel, *Moderne Musikelektronik*, Militärverlag der Deutschen Demokratischen Republik, Berlin, 1st edition, 1989.