EPRO/IAS

Bachelor Studiengang Medientechnik
Fachhochschule St.Pölten

Patrik Lechner

Wien, December 16, 2016

Introduction

Contents

Part I.

Semester 3

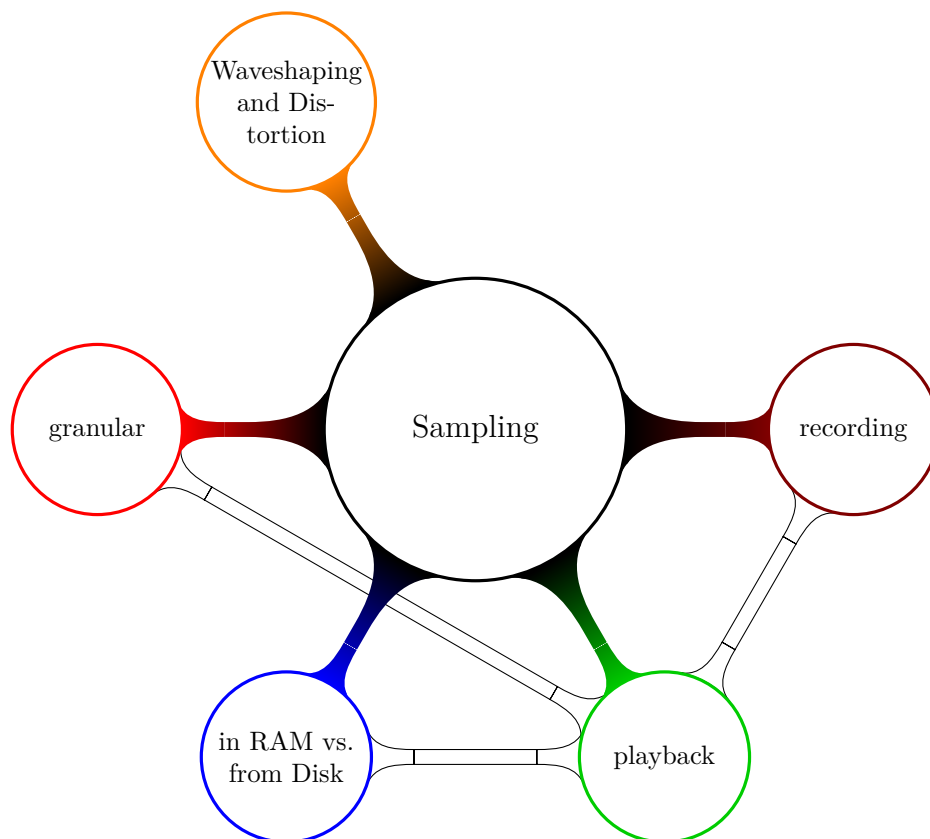# 1. Sampling, waveshaping, and non-linearity

Figure 1.1.: Lecture Contents

1.1. Waveshaping

Wikipedia quote, page "wavesahper":

„The mathematics of non-linear operations on audio signals is difficult, and not well understood." Waveshaping means distortion. It adds overtones, take a look at figure 1.2.
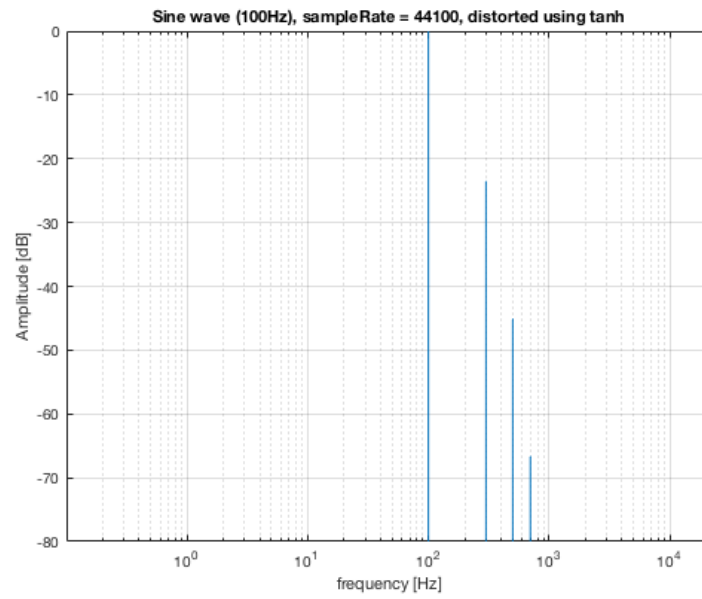
Figure 1.2.: A sine wave has been generated and waveshaping was applied to add overtones.
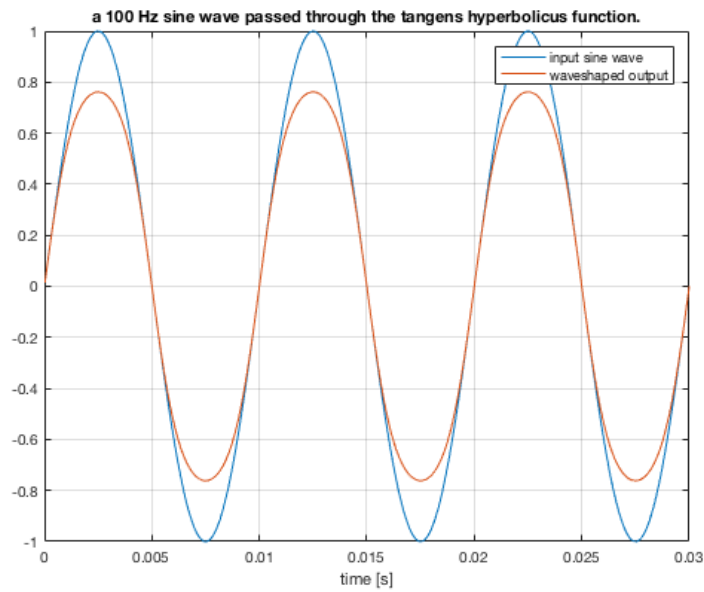

Figure 1.3.: The same as the spectrogram above, but in the time domain. We can see the input sine wave and the slightly distorted output. It may look like just the amplitude has changed, but the sine's actual *shape* has changed slightly

1.1.1. The simplest case: a linear Transfer function.

See 1.4. A linear transfer function is used as a lookup table for a sinosoidal input.
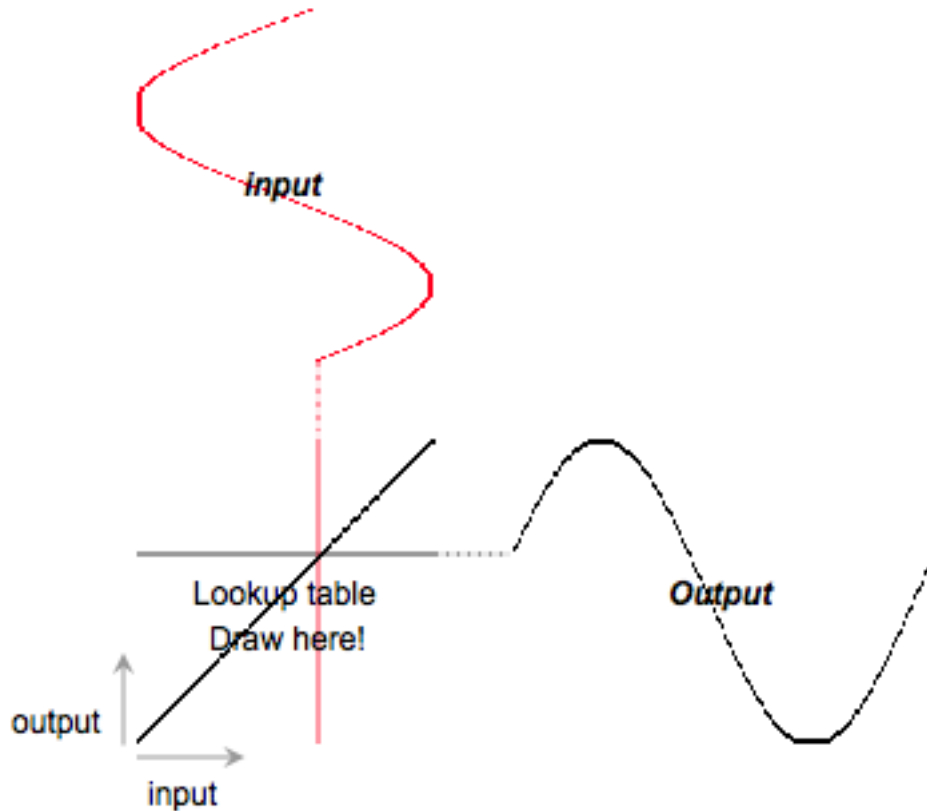
Figure 1.4.: Linear Transfer function

A transfer function in the sense of a waveshaper (a "transfer function" might also mean frequency response in other contexts) is a simple look-up function. Waveshaping means to use an input wave to *look up* values in a table or function. A linear transfer function, let's call it l, can result in no change, for example, it might return $l(x) = x$. This means, that whatever value we pass in, we get the same value out. Other linear transfer functions might *only* change the amplitude. For example $f(x) = x \cdot \frac{1}{2}$. That doesn't seem very interesting. But it might explain the term "linear". A transfer function is linear if it looks like a line if we plot it. Look at figure 1.5.

Non-liner transfer functions behave differently. They map their input to other values, such as $f(x) = x^2$. It may seem trivial, but if we put 2 into f we get 4 as an output. You can also look at figure 1.6 in order to understand what's happening. We again see a linear transfer function but also a non linear one.
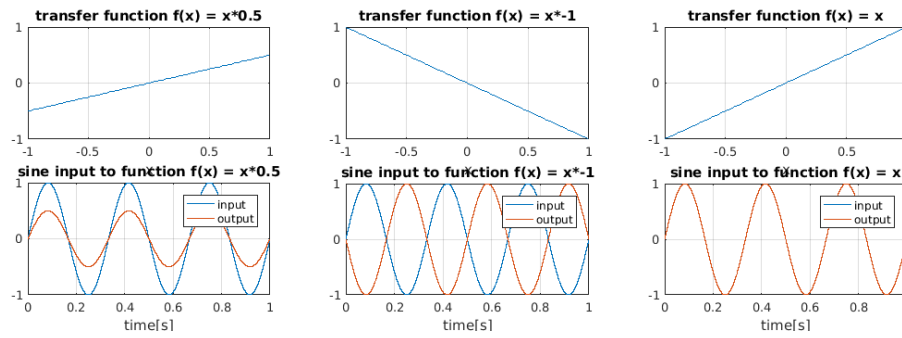
Figure 1.5.: A couple of linear transfer functions and their corresponding effects demonstrated using a sine wave. From left to right: multiplication by 0.5, so attenuation by about 6dB, inversion, and the "do-nothing"-function.
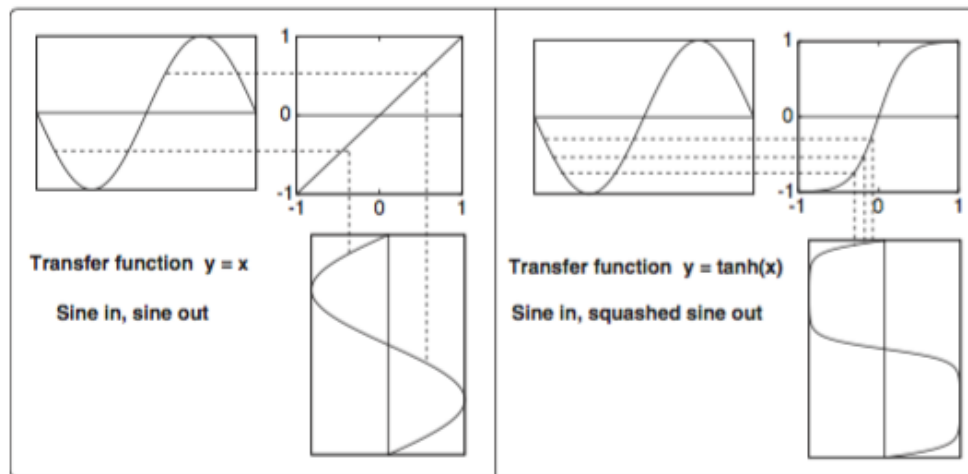

Figure 1.6.: A waveshaping visualization taken from Farnell (2010)

But let's get back to our square function, since it's simpler and we will find some surprising results when analyzing it. Let's first simply plot it too.
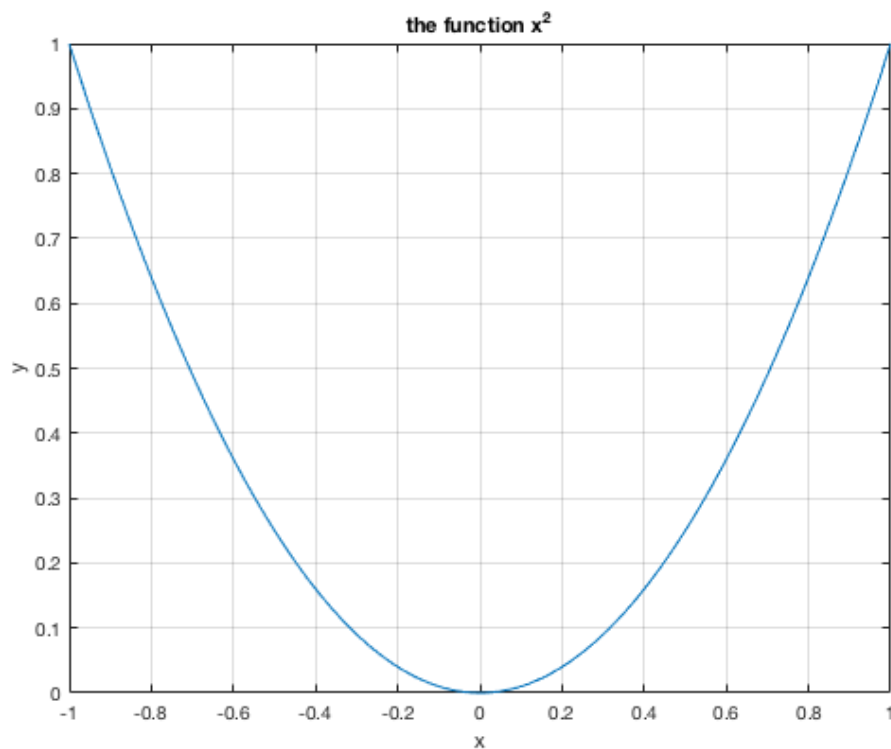
Figure 1.7.: The function $f(x) = x^2$

1.1.2. Simple non-linearity: X^2

So let's analyze what happens if we use this function for waveshaping. Here it is again:

$$f(x) = x^2 \tag{1.1}$$

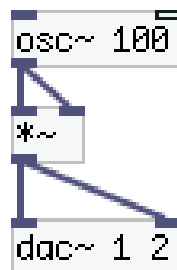Let's simply listen to what's happening, building it in pd:

Figure 1.8.: The square function in pure data, using a 100Hz sine as a test signal. What do you hear?

And we can simply plot what happened if we apply the function before also trying to understand analytically:
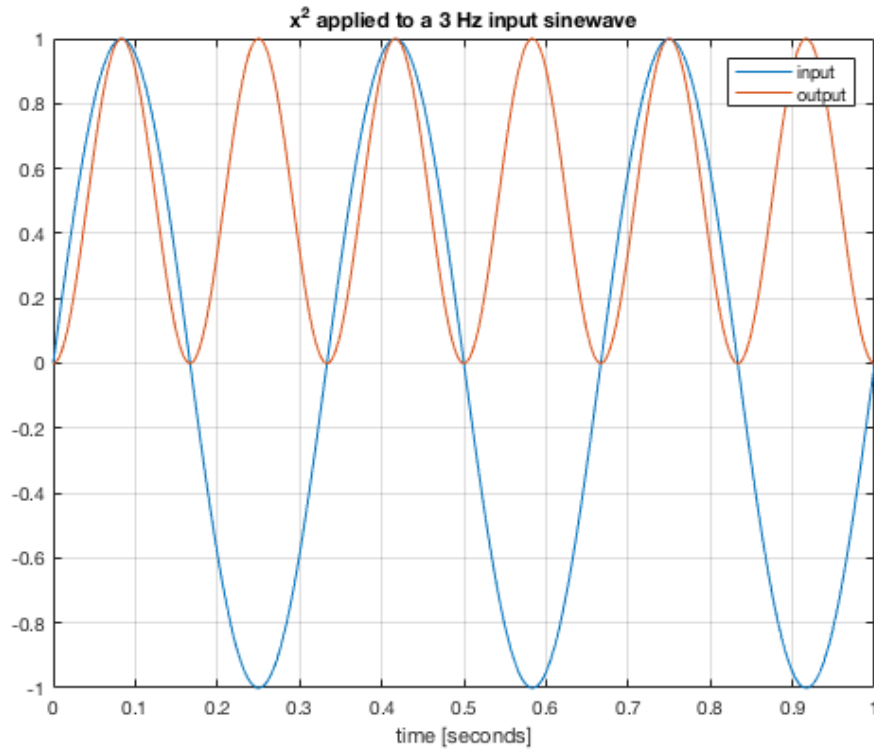

Figure 1.9.: Applying the square function to an input sine wave.

Weird, the input seems to double in frequency (did you hear that?). Let's try to understand what's happening.

So we calculate what happens if we send a cosine through this function, so let's take:

$$x = cos(\omega) \tag{1.2}$$

wit arbitrary ω. We can just ignore ω here for a while. Usually, there should be some indexing variable in the cosine function if we want to describe an oscillator that moves over time, but let's also skip that.

So applying our square function we of course get:

$$y = cos(\omega)^2 \tag{1.3}$$

This again results in:

$$y = cos(\omega) \cdot cos(\omega) \tag{1.4}$$

So far so trivial. Note that a multiplication of two oscillators is called *Amplitude*

Modulation (actually, in this case we encounter "Ring Modulation", but let's ignore that also), and we know things about Amplitude modulation, namely:

> When multiplying two oscillators, we get sum and difference of the two input frequencies. (And the whole output is attenuated by 6dB)

The above statement in equation form:

$$cos(a) \cdot cos(b) = \frac{cos(a+b) + cos(a-b)}{2} \tag{1.5}$$

We could also have looked up this *trigonometric identity.* This means for our experiment with our cosine squared:

$$y = \frac{cos(\omega + \omega) + cos(\omega - \omega)}{2} \tag{1.6}$$

So:

$$y = \frac{cos(2 \cdot \omega) + cos(0)}{2} = \frac{cos(2 \cdot \omega)}{2} + \frac{1}{2} \tag{1.7}$$

We arrive at the same result! **But is this true for every input? That would mean we just built a frequency shifter, did we? No.** Waveshaping is much more complicated, which is immediately obvious when we try to do the same t two oscillators:

$$x = cos(\omega_1) + cos(\omega_2) \tag{1.8}$$

then

$$y = (cos(\omega_1) + cos(\omega_2))^2 \tag{1.9}$$

$$y = cos(\omega_1)^2 + cos(\omega_2)^2 + 2 \cdot cos(\omega_1) \cdot cos(\omega_2) \tag{1.10}$$

And finally:

$$y = \frac{cos(2 \cdot \omega_1)}{2} + \frac{1}{2} + \frac{cos(2 \cdot \omega_2)}{2} + \frac{1}{2} + 2 \cdot (\frac{cos(\omega_1 + \omega_2) + cos(\omega_1 - \omega_2)}{2}) \tag{1.11}$$

1.1.3. How can waveshaping be implemented?

Take a look at figure 1.10. What do you think is happening? On the left side, we see waveshaping as we did it above, using a mathematical function, in this case the tangens hyperbolicus, to distort our signal. On the right side, we see a table that contains the authors desperate attempt to draw the same function with the mouse. The results are theoretically equivalent (if the function in the table was correct), but what are the advantages and disadvantages of the two approaches? Also, be sure to understand what the addition of 1 and the multiplication with 50 does on the right side. Hint: the array has 100 points.
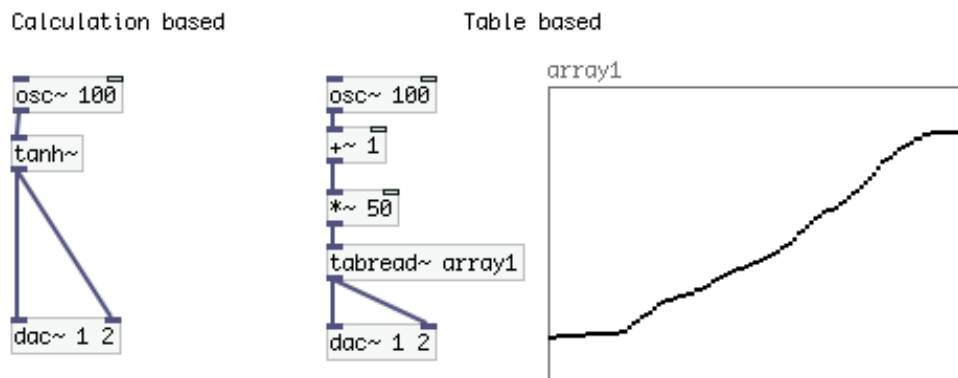
Figure 1.10.: Left: using a mathematical function to calculate the output. Right: using a table to look up the output.

1.1.4. How is Waveshaping related to other techniques?

Sampling

If we take a look at figure 1.18, we see that we play a sound file by accessing a buffer (wavetable) using an index, an oscillator. This is effectively the same setup as we would build for distorting an input sound. Also take a look at figure 1.11, which showing us that waveshaping and wavetable synthesis are identical.
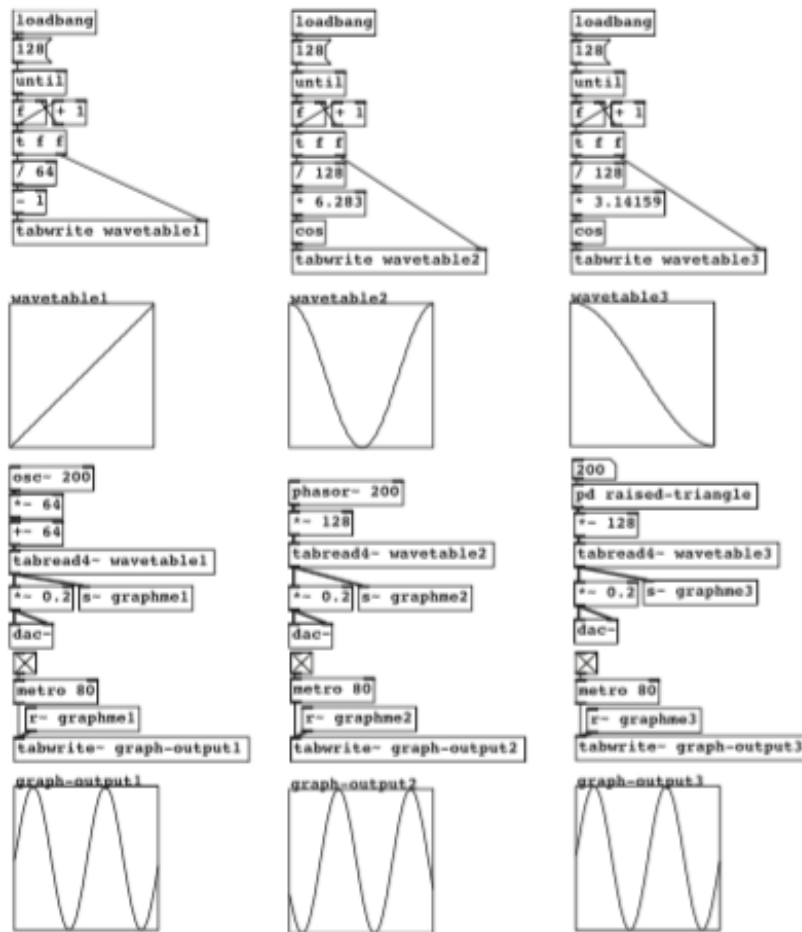
Figure 1.11.: Picture taken from Farnell (2010), showing the identity of waveshaping and wavetable synthesis

Modulation

While we will talk about modulation in a separate chapter, let's loosely define amplitude modulation (AM) as the multiplication of two oscillators and frequency modulation (FM) as varying the frequency of an oscillator using another oscillator. So, as we have also seen above, AM looks like this:

$$y = sin(a) \cdot sin(b) \tag{1.12}$$

and FM looks like this:

$$y = sin(sin(a)) \tag{1.13}$$

in practice, the a and b terms are a bit more complicated, but we will look at this later. That certain cases of AM are identical to waveshaping has been shown above, think

about the square function again. This of course does not mean that waveshaping can do everything AM can do and it does not mean that AM can achieve everything that waveshaping can. This should just show that we can understand the techniques from the perspective of another. What about FM? Well if our lookup function we use for waveshaping is a sine wave, we arrive at the exact same equation as how we defined FM above. Again, practically speaking, the results we get with these two techniques are very different, but we can see the connections.

1.1.5. Why is Waveshaping useful?

The output spectrum is dependent on the input amplitude. This makes it easy to create complex evolving spectra.

1.1.6. What are the problems with waveshaping?

Waveshaping adds overtones. When we build a waveshaper, we have to be aware of aliasing. Take a look at figures 1.12 and 1.13. Sinewaves have been amplified and clipped here.
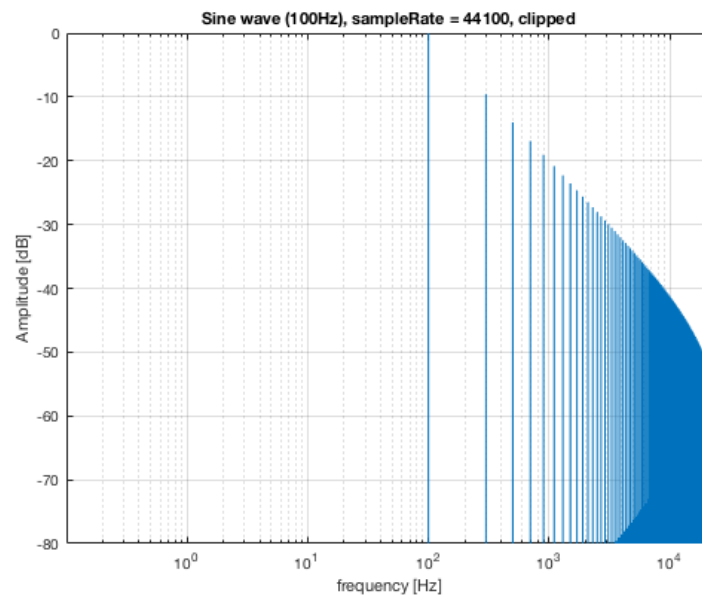

Figure 1.12.: A sine wave was generated and clipped. Clipping is a form of waveshaping which adds many overtones. Note how high frequencies fold back into the lower parts of the spectrum because they exceed the Nyquist-rate.

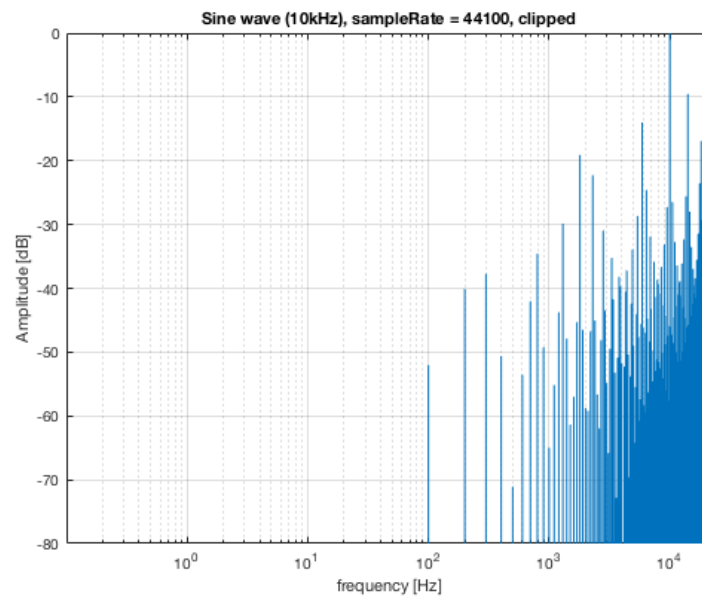In pd we could achieve this like in figure 1.14.

Figure 1.13.: Again, a sine wave, this time with a higher frequency to begin with. Extreme clipping has been applied by boosting the input amplitude. The aliased overtones are all over the place, even below the input frequency.

Figure 1.14.: Clipping an amplified sine wave in pd

What does the output look like? Let's not only look at the spectra but also at the time signal:
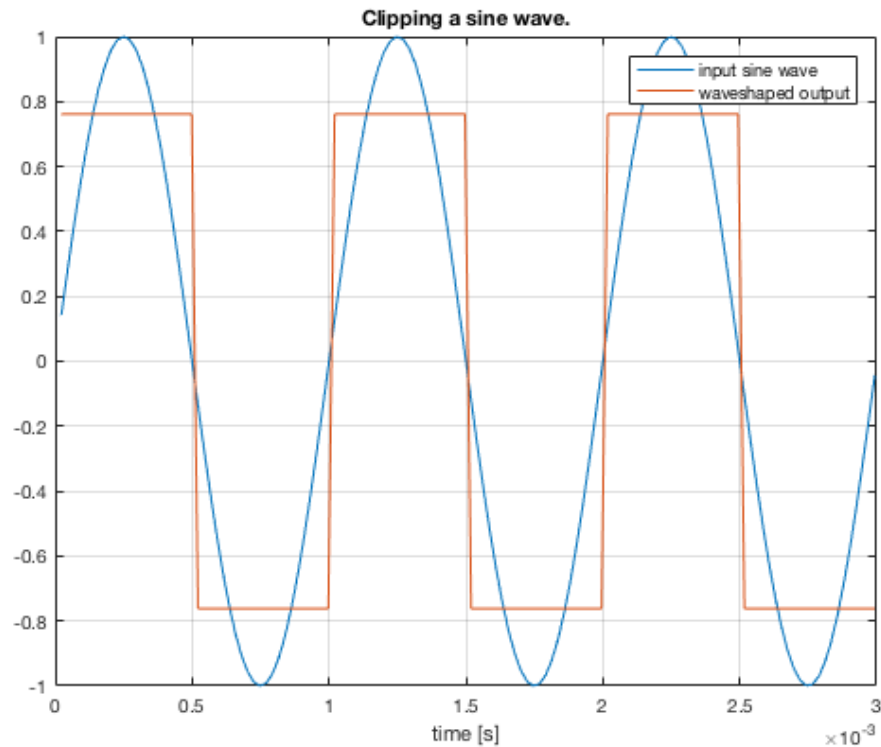
Figure 1.15.: An amplified and clipped sine wave in the time domain.

We see that we can arrive at a square-wave like result, but this square-wave is not anit-aliased.

The problem of aliasing in waveshaping is usually treated by over-sampling. This does not solve the problem but lessens it significantly resulting in cleaner, arguably better sound. Oversampling means that, if we work at a sample-rate of 44.1kHz, the input is up-sampled, essentially interpolated, to be at a sample-rate of 88.2kHz. Then the waveshaping is applied, leaving room for high frequencies up to 44.1kHz. Using a lowpass filter, high frequencies over 22.1kHz are then attenuated as much as possible, in order to be able to down-sample again to reach our initial sample-rate of 44.1kHz. To state it more simple: Waveshaping is usually encapsulated in a process that runs at higher sampling rates in order to lessen aliasing.
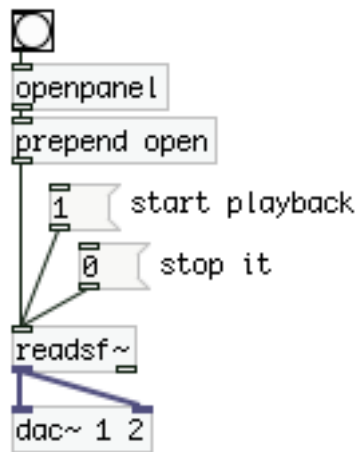
1.2. Sampler
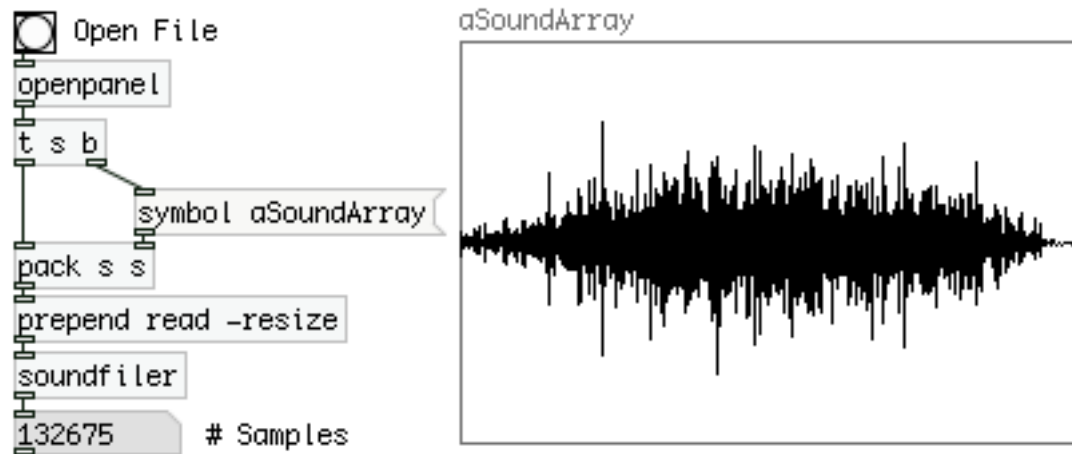
Work in progress.

Figure 1.16.: simpleSampler

Figure 1.17.: sound in Ram

Loading Audio to an Array (to RAM)
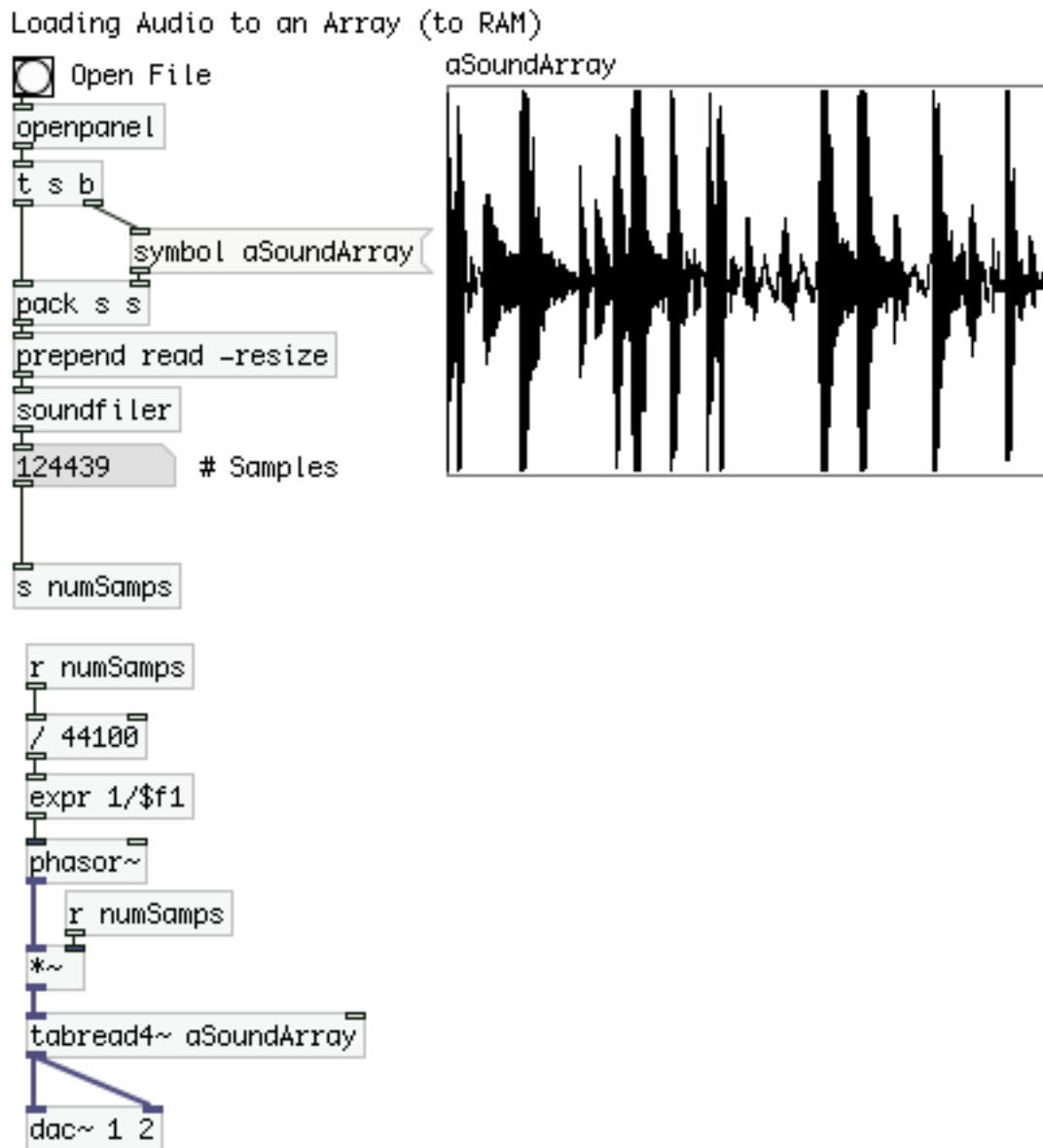
◯ Open File

openpanel

t s b

symbol aSoundArray

pack s s

prepend read -resize

soundfiler

124439   # Samples

s numSamps

aSoundArray

r numSamps

/ 44100

expr 1/$f1

phasor~

r numSamps

*~

tabread4~ aSoundArray

dac~ 1 2

Figure 1.18.: RamFilePlayback
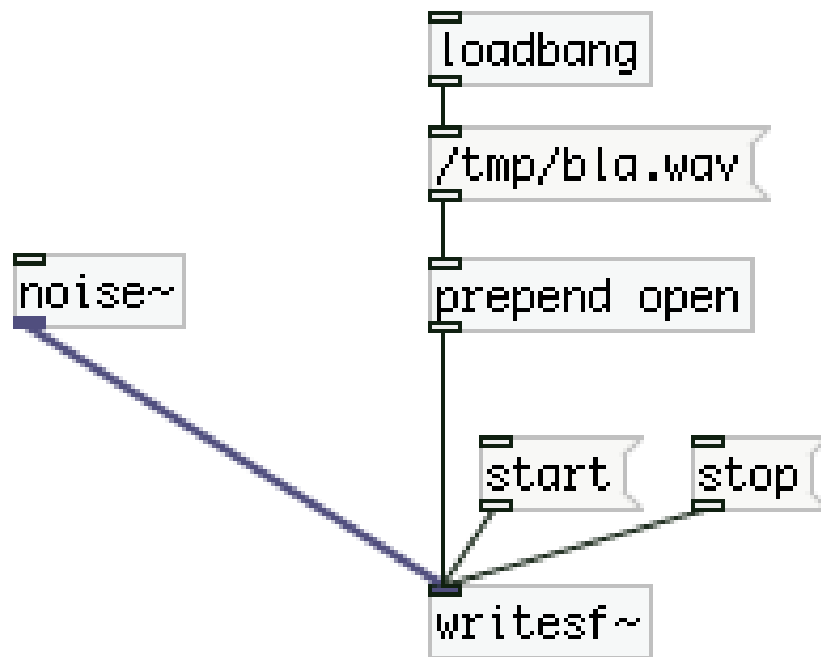
Figure 1.19.: writing Audio to disk

Loading Audio to an Array (to RAM)

◯ Open File

openpanel

t s b

symbol aSoundArray2

pack s s

prepend read -resize

soundfiler

153400   # Samples

s numSamps

aSoundArray2

r numSamps

/ 44100

expr 1/$f1

sig~

speed

*~ 1

phasor~

init 40

42   grainsize (ms)

+~

r numSamps

metrum 30 1 ⊠ ▷30

*~

expr 1000/$f1

snapshot~

phasor~

s pos

*~

mstosamps~

delta~

<~ 0

sah~

+~

tabread4~ aSoundArray2

*~ 0.5

sin~ windowing

*~

dac~ 1 2

(NOTE: this patch demonstrates granularsampling, but is, in fact, not entirely correct, since it is not implemening multiple grains.)

Figure 1.20.: moreSamplimg.pd, a simplified version of granular sampling

1.3. Hausübung

1.3.1. Testmodul

baue ein audio Testmodul mit folgener spezifikation:

- Ein audio output

- verschiedene klangquellen wählbar:

  1. White Noise

  2. Sinus (freq. einstellbar)

  3. soundfile (file wählbar)

- GUI

- verfügbar(in eurem pfad, und jederzeit abrufbar als abstraction)

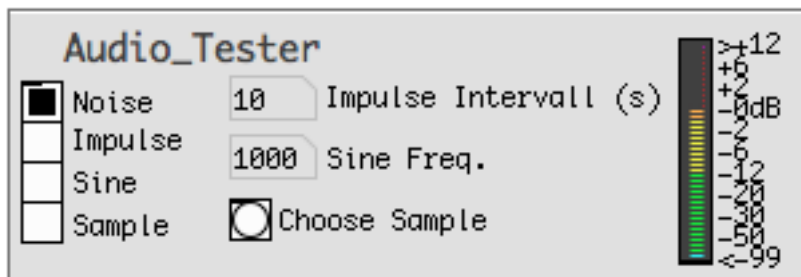- output pegel sichtbar (level meter)

Figure 1.21.: audioTester.pd, zu bauen als Hausübung

List of Figures

Bibliography

Farnell, A. (2010). *Designing sound.* MIT Press, Cambridge, Mass.