Sampling, Oversampling, and Aliasing

Sam Fischmann

v1.0.5 — 08/01/2023

https://www.musikhack.com

info@musikhack.com

Table of Contents

Introduction

Aliasing distortion, an unpleasant artifact of sampling audio, and **oversampling**, a process that increases the sample rate of audio, are very popular topics in forums, product marketing, and video tutorials. It's no surprise! Here are three important reasons:

- **Aliasing** is a fundamental challenge with digital audio. It sounds bad.
- **Oversampling** can reduce aliasing until it's beneath the hearing threshold.
- When oversampling is used for marketing, the quality is boiled down into a single number, so it seems easy to compare one product to another.

As it turns out, not all oversampling is the same. You don't always need it, and developers face several trade-offs when implementing it. This paper aims to help you understand how sampling and aliasing relate to each other, how oversampling addresses aliasing, and when using oversampling is a good idea.

The paper is divided into two parts: **Understanding Sampling and Aliasing** and **How to Oversample Wisely**. If you aren't interested in the technical background, skip directly to part 2!

Part 1: Understanding Sampling and Aliasing

What is Sampling?

When I say sampling, I mean that we have a **continuous** signal (physical reality) and we want to represent it using a finite amount of data, so we sample **discrete** (countable, individual) points along that signal, evenly spaced in time. Here's a simplified description of how sound travels from the environment into your computer:

1. The air carries **continuous** sound waves to the microphone.
2. The air pushes the microphone diaphragm around, making it vibrate.
3. The diaphragm moves a magnet, creating a modulating electric current.
4. The current goes through an XLR cable into your audio interface, and is amplified by the microphone preamp.
5. The preamp passes audio to the analog-to-digital converter (ADC), which grabs **discrete** values from the electric current at even intervals (the sample rate).
6. The sample values are passed from your interface to your computer, which saves them as an AIFF or WAV.[1]

The opposite process occurs on the way back. The discrete samples are fed back through a digital-to-analog converter (DAC), which reconstructs a continuous electric current from the samples and sends it to your speakers, which convert it to a magnetic force to move the speaker cones. Voila: sound!

There are two degrees of precision we care about for each sample:

- **Sample rate** describes how many discrete samples the ADC takes of the continuous signal per second.
- **Bit depth** describes how precisely samples measure the strength of the signal.

[1] Modern ADC and DAC converters use a more complicated process known as Delta-Sigma modulation to do this work. The signal is sampled at an extremely high sample rate, >3 MHz at 1-bit depth. Think of the bit as "higher or lower than the previous sample." The converter then translates the stream of bits to a more typical sample rate and bit depth using purely digital techniques.

In this paper, we're discussing the effects of samples over time, rather than the accuracy of each sample, so we're going to talk about the sample rate.

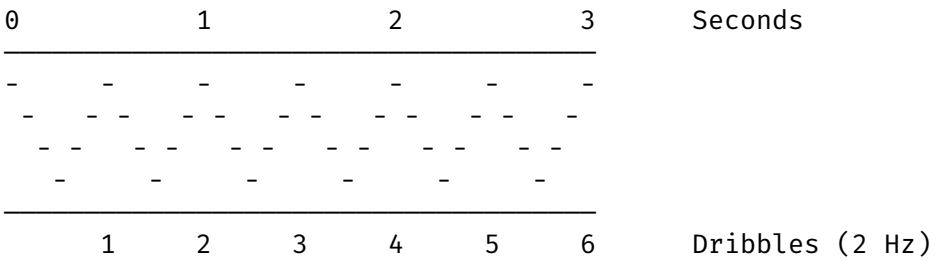
How Sampling Causes Aliasing

To recap: the sample rate determines how many discrete samples the ADC takes per second from a continuous signal. The main tradeoff when selecting a sample rate is:

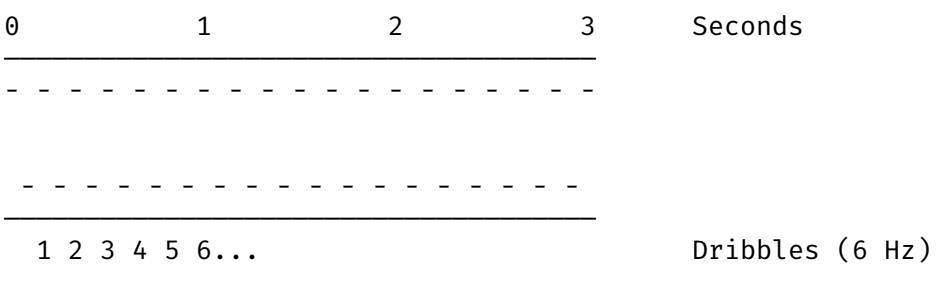
- **Lower rates** use fewer samples and are easier to process and store.
- **Higher rates** use more samples and resources, but can represent higher frequencies in the signal.

The highest frequency that can be played back at a given sample rate is called the **Nyquist frequency** (we'll use Nyquist, for short). Lucky for us, the math to find Nyquist is easy: it's 1/2 the sample rate! For 44,100 samples per second (44.1 kHz/CD quality), the highest frequency we can represent is about 22 kHz. Engineers picked this rate in the 1970's because its Nyquist frequency is slightly higher than the highest frequency humans are known to hear. So how does any of this relate to aliasing?

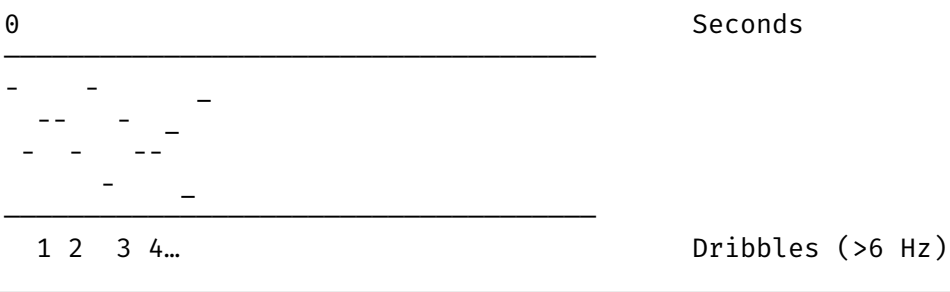
Imagine we're taking a video of somebody dribbling a basketball twice per second (2 Hz). If we're filming at 12 frames per second (12 Hz), we could see the dribbles easily, twice per second:



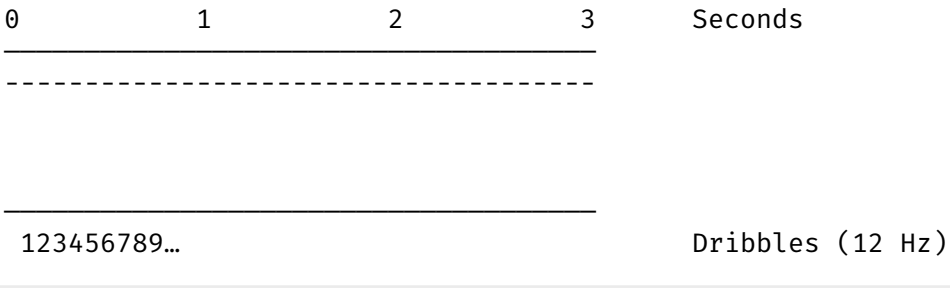
Even if the player started dribbling at 6 Hz, we could record the pattern accurately:



But now we have a problem: as soon as the player dribbles any faster than 6 Hz, our second sample will happen after the ball has bounced and is headed back up. And the third might be after it went back up and is on the way down:



Sampling does not record the direction of the ball, only its position in the air, so we don't know how to reconstruct the movement of the ball. At sample 2, is the ball still going down towards the ground, or is it bouncing up? Even worse, if the player dribbles at exactly the same speed as the sample rate, the ball doesn't appear to move at all:



This is very similar to watching a wheel that spins faster and faster. Your brain can't process it fast enough, and you start to see the wheel slow down, stop, and eventually

spin backwards… This is called the “wagon wheel effect”: it’s aliasing happening **directly in your brain**!

Similar to the wheel, audio aliasing results in frequencies that “bounce off” or “fold back over” the Nyquist frequency during sampling. Frequencies in the continuous signal immediately above Nyquist cause aliasing distortion in the sampled signal that is too high pitched to hear. As the frequencies in the continuous signal get higher and higher, they eventually fold back into the audible range as **inharmonic, metallic ringing** in the sampled signal. That’s the sound of aliasing! If the sound source is complex and the aliasing is bad, it fills in more and more frequencies until it becomes **noise**.

In audio, we rarely have a simple signal like we did in our ball example. We have very complex waveforms! There’s some brilliant math called Fourier’s theory, which shows us that we can think of any complex, periodic signal as a sum of many component sine waves, each with different frequencies, volumes, and phases. Depending on the literature and process being described, these components might be called:

- **bands** (we’ll use this term in a minute, take note)
- bins
- harmonics
- overtones
- timbre (when describing the sound of all components at once)

While the math of Fourier’s theory is beyond the scope of this paper, it’s important to know that thinking about signals as sums of sine waves is very helpful. By thinking about signals this way, we can reason about what happens to a very complex signal by examining what happens to individual sine wave components, which are much easier to run through pure mathematical functions and analyze. For instance, we can think about aliasing like so: when a complex signal is sampled, the components below the Nyquist frequency of the sampling rate are preserved, but higher frequencies alias back towards zero.

Removing Aliasing During Sampling with Anti-aliasing Filters

Here's a new question. If I blow a dog whistle, even though I can't hear it, won't that get into the DAC and put aliasing nonsense into my sampled signal? Yes it will! There are all sorts of high frequencies we can't hear that mics pick up or synths create. If they are above Nyquist, all of them will alias and make our recording sound terrible. Enter: **anti-aliasing filters**. Their sole job is to attenuate frequencies that are above the Nyquist frequency of the sample rate and let lower frequencies pass through (they are **low-pass** filters). Let's update our digital recording steps from earlier:

1. Air vibrates
2. Microphone diaphragm picks up vibrations
3. Diaphragm creates electric current
4. Current goes through preamp in your audio interface
5. Preamp passes to ADC
    I) ADC applies an analog **anti-aliasing filter**[2]
    II) ADC samples the filtered signal
6. The interface passes the samples to the computer, which prints a file

Because all the frequencies above Nyquist have been greatly attenuated by the filter, aliasing is no longer audible! We call the resulting signal **band-limited**, because the frequencies you'd use to mathematically describe it are **limited** to **bands** that are below the Nyquist frequency. Here's the takeaway:

> *Anti-aliasing filters must be applied in either the continuous, analog domain or higher sample-rate signal **before sampling down** to a lower rate, because aliasing is a mathematical problem that occurs during sampling itself. You cannot remove aliasing after you've sampled down!*

[2] In Delta-Sigma modulation, the ADC uses a digital filter for this purpose while converting your signal from a very high sample rate at 1-bit to a lower sample rate at 16 or 24 bits. Different method, same issue!

Reconstructing an Analog Signal from Samples

So far, we've been walking familiar ground for battle-hardened engineers and producers, but here's something about digital audio a lot of people don't think about: how does the DAC turn a series of discrete samples back into a continuous signal? How do these dots get connected? Does it just… make them up? It might surprise you to know that the answer is (usually) another analog low-pass **reconstruction filter**. Here's the way a simple DAC might work:[3]

1. The DAC converts each sample into a flat electrical pulse.
2. The pulses move through an analog low-pass filter.
3. The output of the filter is an electrical signal that moves smoothly between pulses like a wave. This is the signal that is eventually passed to your speakers.

If you have a system with both an ADC and DAC, it's possible they use identical filters for anti-aliasing on the way in and reconstruction on the way out! Just like their anti-aliasing counterparts, reconstruction filters vary in type and quality. This is another big takeaway:

> *Digital audio is reconstructed into analog audio through a low-pass filter **on your listener's hardware**. You don't know the characteristics of this filter, but most of them are decent enough. When audio manufacturers, developers, or standards bodies make claims about precision that might affect your processing or purchasing decisions, remember that you must always consider that **your listener's DAC may render those claims unimportant**, or even wrong.*

[3] A Delta-Sigma DAC filters twice on the way out: a digital low-pass filter to convert the 24-bit 44.1 PCM stream into a 1-bit > 3 MHz+ stream,, then a tamer analog low-pass filter to remove the quantization noise that creeps in during the digital conversion.

How Much is Lost in Sampling and Reconstructing Signals?

This may come as a surprise, but theoretically, it's not lossy! We have a stellar mathematical theory that proves the case: the Nyquist-Shannon sampling theorem (shout out to Whittaker, too). The theorem proves that as long as a signal is **band-limited**, meaning its harmonic content does not exceed Nyquist, we can sample and reconstruct that signal perfectly. It's the mathematical proof for why anti-aliasing filters are necessary.

When recreating the continuous signal from your band-limited sampled signal, the theory shows that if you replace each individual sample with a whole **sinc function** centered at the same place in time as that sample, scaled by the sample value, and sum all of these sinc functions together, you can recreate your continuous signal perfectly. If you've been deep in audio technical stuff, this technique might remind you of convolution!

In reality, sinc functions run infinitely forwards and backwards in time. It's impossible for a machine to do the math, because it would require infinite calculations. So engineers approximate this process, most commonly using the method previously discussed: by sending electrical pulses to an analog low-pass filter whose impulse response closely approximates a sinc function. Whatever process your DAC is performing, it's **meant to approximate Nyquist-Shannon reconstruction**. While nothing is perfect, most DACs are close enough that we can't hear the difference. The closer they are, the higher the quality, which is why audiophiles shell out big bucks for standalone DACs!

How Aliasing is Created in DSP, After Sampling

OK, so now we have a basic understanding of how audio gets into the digital domain and comes back out without aliasing. Great! Let's talk about how aliasing is introduced into your signal **after** it has been sampled. The bottom line is this: if digital signal processing (DSP) introduces new frequencies above Nyquist, the output samples will have aliasing distortion! Luckily, we have a mathematical way of knowing which types of processes do and don't add these frequencies. We classify them with the following names:

- **Linear** – does not add new frequencies to your signal
- **Non-Linear** – does add new frequencies and may cause aliasing

In order to be **linear**, a process must follow both of the following rules:

- **Additivity** – adding two signals and then processing them gives the same result as processing each signal by itself, then adding them.
- **Homogeneity** – increasing the strength of the input proportionately increases the strength of the output.

If a process does not follow both rules, it is **non-linear**, which means it changes the shape of the signal, and therefore alters its frequencies. If new frequencies show up above Nyquist, you've introduced aliasing into your sampled audio signal.

Common Linear and Non-Linear DSP

Here are some examples of typically linear and non-linear processes:

Process	Linear	Non-Linear
Gain	X	
Clean EQ (both linear phase and minimum phase)	X	
Delay	X	
Reverb	X	
Creative EQ (like a console channel emulation)		X
Compressor		X
Saturation/Distortion		X
Tape emulation		X
Clipper		X

This table isn't comprehensive, but it's a handy starting point! For some in-depth reading material about this topic, see this article from CCRMA.

How Oversampling Lessens Aliasing in Non-Linear DSP

Here's how oversampling reduces the amount of aliasing caused by non-linear DSP:

1. Non-linear DSP adds frequencies above Nyquist, causing aliasing.
2. By increasing the sampling rate before performing non-linear DSP, we also increase the Nyquist frequency. This means that while we are at a higher sampler rate, the signal can represent higher frequencies accurately, so there is more frequency headroom for the non-linear DSP.
3. After we are done with non-linear DSP, we put the oversampled signal through a low-pass anti-aliasing filter with a cutoff set just under our original, lower sample rate. This removes the frequencies that would alias at our lower sample rate.
4. Finally, we downsample the signal back to the original rate. Less aliasing!

Sounds great in theory, but how do we do it? Let's look at some techniques.

Method 1: Zero-Padding and Filtering

Think of oversampling as taking a low resolution input and writing it onto very high resolution output. A continuous signal is the highest resolution of all because you can zoom in forever, so if a low-pass reconstruction filter works for a DAC, it will also work for oversampling:

1. Insert a 0 (zero) sample between each sample in the signal, doubling the sampling rate. This is called **zero-padding**.
2. Run the signal through a low-pass filter to fill in all the padding (**interpolation**).
3. Repeat as many times as you want. Each pass is an oversampling "stage."
4. Do non-linear processing to the oversampled signal.
5. Just like an ADC, run the high-resolution, oversampled signal through an anti-aliasing low pass filter, and remove the extra samples (**decimation**).

In general, developers pick one of two different filtering methods during interpolation and decimation.

- **Infinite Impulse Response** (IIR) – cheap compute, low latency, non-linear phase, higher maximum ripple in the signal
- **Finite Impulse Response** (FIR) – expensive compute, more latency, linear phase, lower maximum ripple in the signal

Beyond that, developers might pick different filter shapes:

- **Steep (brick wall)**
  - **FIR** – Expensive compute
  - **IIR** – Sharper phase distortion
  - **Both**
    - Better aliasing reduction (for the same filter cutoff frequency)
    - Larger ripple in the signal (higher visible peaks)
    - More resonance at the cutoff near Nyquist
- **Shallow**
  - **FIR** – Cheaper compute
  - **IIR** – Smoother phase response
  - **Both**
    - Worse aliasing reduction
    - Smaller ripple in the signal (flatter)
    - Less resonance at the cutoff near Nyquist

So, you can see why somebody might argue:

> *"I can make a shallower FIR filter to get linear phase, less CPU, and less resonance around the Nyquist frequency. To do this, I'll lower my cutoff to 17,000 Hz, nobody can hear that anyway!"*

Then again, a different engineer might say:

> *"All frequencies must be preserved, but who cares about phase? I need the steepest filter with the least latency. That will keep my plugin more versatile for live use, so I'll use IIR right up at the limit."*

On top of that, there are more decisions, tricks, and ways to combine the filter and resampling mathematically to save compute power. The filters for each stage may be different, and some will eschew phases altogether, pad a bunch of zeros after each sample, and just low pass filter one time! All these approaches could handle your audio in subtly different ways, whether it's transients, imaging, aliasing reduction, or simply an inaudible, but measurable, difference.

Method 2: Windowed Sinc Interpolation

Remember the **Nyquist-Shannon** sampling theorem we discussed when talking about sampling? We said then that by turning each sample into a scaled, time-shifted sinc function and summing all the functions, we could reconstruct a continuous signal. So why don't we just do that, and then sample from that continuous signal?

We can, but we must approximate the infinitely-long sinc function, because we only have finite computing power. Here is a simplified method to accomplish this:

1. Crop the **sinc function** some number of samples (such as 32) on either side of zero.
2. Write a new polynomial function that approximates these samples, and all the values that would be between them. A polynomial function is much more efficient to calculate than sinc.
3. Multiply the result of the polynomial by a function that smooths out its edges to zero. This is called a **windowing function.** It is often named after a mathematician or two, such as "Hamming," or "Blackman-Harris."

4. Explode each individual sample into a copy of this windowed function, scaled and time-shifted to the location of each sample, then sum them up and sample them at any rate we want.

Developers have come up with all sorts of tricks for their windows and sinc approximations, each with its own trade-offs. Some even combine the traditional and windowed sinc approaches!

Do Different Plugins use Different Oversampling Methods?

Yes! If you bought a different brand of shampoo, conditioner, soap, face wash, toothpaste, deodorant, and laundry detergent, it would be hard to say that your smell is in any way intentional. Yet, that's what most people do. Or they cover the melange with perfume. Or just… never clean up. And yet, here we are doing that with oversampling: even if we are working on a single, final stereo track, if we dump a bunch of plugins on it, each will use different kinds of oversampling with different filters going up, down, reconstructing, etc.

- **On the one hand**, most oversampling methods are pretty good, and the frequencies involved are very high up, so many people don't hear changes.
- **On the other hand**, especially during final processing stages, we don't want to pass our digital signals through too much. We want purity and clarity! Sometimes, we incorrectly attribute unintentional changes caused by oversampling to intentional changes caused by a particular plugin.

In Review

There are a lot of ways developers can implement oversampling, each with different motivations. You cannot control what method each plugin uses, and it's unclear exactly how a plugin will modify your sound when you enable it. For these reasons, it's important to make good decisions about when to turn oversampling on.

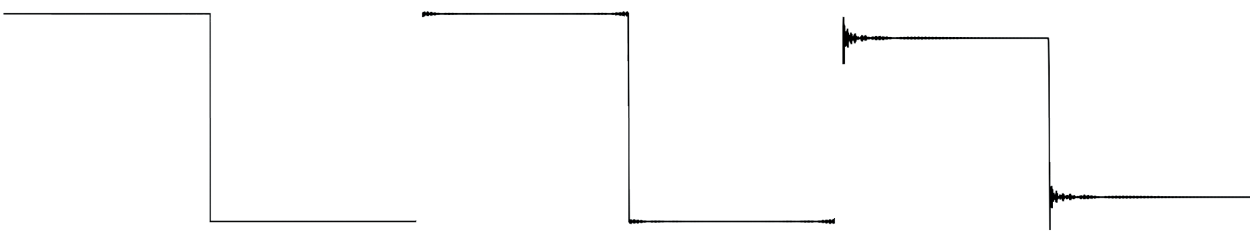
Part 2: How to Oversample Wisely

Problems with Oversampling

Now that we have good background information about aliasing and have discussed oversampling methods, let's talk about the problems that oversampling can cause.

1. **High CPU usage** — This is the most well-understood problem. For example, while oversampling by 4x does use the CPU, most of the work is done by the non-linear math after the oversampling that now needs to be calculated 4x as much. All oversampling methods will be similar in this regard.
2. **Reduced Headroom** — Oversampling filters can cause significantly higher peaks on material that has frequencies above their cutoff values, or on material that already contains aliasing frequencies. Because a ton of non-linear processing is used in effects, software synths, and sample packs, having these high frequencies is extremely common. Also, different oversampling methods cause different ripples, so you must consider each of your plugins individually.
3. **High frequency roll off** — If several of your plugins use gentle oversampling filters on the same track, you'll start to hear a high-frequency roll-off if you stack them together.
4. **Phase changes in the high end** — Some material is sensitive to phase issues very high up, but it takes a sensitive ear and sensitive material to notice this problem.

Let's talk a bit about problem **2**, reduced headroom. If you look at a square wave, or a signal with sharp edges in the shape (this indicates super high frequency content that contains aliasing), you'll see a big difference between the output of different filters: **ripples**, due to the Gibbs effect:



Input square wave | FIR oversampled and back | IIR oversampled and back

These ripples don't necessarily change the way you perceive the sound, but they reduce the headroom you have in your digital signal, and create more, and higher, peaks. This phenomenon is worse for IIR than FIR filters, because FIR filters are symmetrical, distributing the ripple before and after an edge, while an IIR filter accumulates it all after the edge. These ripples affect all processing you do **after** an oversampling plugin, such as **true peak limiting**. If the limiter has to pull back further to keep the peak under, it will reduce the dynamic range, deadening your sound![4]

If, like nearly all music, any of the material in your mix has gone through plugins, comes from software synths or samplers or loop packs, or was in any way modified after it was initially recorded, IIR oversampling may reduce your headroom.

When and How Much to Oversample

Let's boil this down into a small set of rules. Ideally, you should oversample a signal:

- only when turning oversampling on or off makes an **audible** difference
- only to an oversampling factor (2x, 4x, 8x, etc.) that makes an **audible** difference
- as **few times** as possible

Yes, I am aware **this advice is trite, boring, and requires no technical expertise**. But it is **correct**. If you can't hear the difference, you're only introducing more filters and processing into your system that you do not control, and without audible benefit.

It's most important to listen to the audio you're working on and decide if you can hear a difference. For some effects, particularly creative ones, aliasing is an intentional part of the sound. In those cases, there is no point in removing aliasing!

[4] A true peak meter on the square wave will read higher after an oversampling plugin. We can assume the same of your limiter!

If you have many tracks that you will mix together later, it's possible that while each track has inaudible aliasing, the mix will build up enough high-frequency aliasing distortion that it will bother you. If it does, oversample the effects on the individual tracks. The next section provides a simple technique to determine which processes are causing the most aliasing, using your ears.

Using a Sine Sweep to Determine when to Oversample

Here is a detailed, reproducible approach to hear aliasing in your system:

1. Run a 10-second sine sweep through your set of plugins, all the way up to 20 kHz.
2. Disable all the plugins, then enable a single one.
3. Listen to the sine sweep. When the frequency gets really high, if the plugin is aliasing, a quiet, whooshing, faster sine sweep will bounce up and down underneath the main sweep, which (depending on your age) may no longer be audible after crossing 17 kHz.
4. Toggle oversampling for the plugin (if available), and see if this helps. If it does, leave it on, raising the factor to a level where the aliasing is inaudible.
5. Repeat steps 2-4 for each plugin in the chain.
6. Enable the whole chain at once to test for any inter-plugin issues.

If you hear no bouncing whooshing sound underneath, congratulations! You've successfully eliminated meaningful aliasing distortion from your channel strip.

The sine sweep technique is important because it doesn't require any special technology or tools, it cannot be fooled by fancy words, diagrams, or marketeers, and it forces you to use your ears to make decisions. After all, that's why we're in this, isn't it?

Further Techniques to Reduce how Often you Oversample

When producing or mixing, the simplest way to reduce oversampling and downsampling many times is to **run your project at a higher sampling rate**. All the processing you do will have a higher Nyquist frequency, pushing aliasing further out of audible range. Then, it gets filtered away when your signal is put through the low-pass filter that determines its final output format, usually at 22 or 24 Khz. When you run your project at a high rate, you only have to downsample at the end.

When mastering, if you're given a 2-track at 44.1 or 48 kHz, use the highest-quality upsampling software you can find to raise its sampling rate to something standard, like 176.4 Khz (for music) or 192 Khz (for broadcast). I chose these rates because they are a power of two above the standard sampling rate delivered for each format. This makes them perform the best when up- and downsampling to the delivered rate. Run your project at this rate, and you should need a lot less oversampling to pass the sine sweep test described previously. You can then downsample one time when you are finished mastering, and have great results.

Some sequencers, such as Reaper, have a feature called **chain oversampling** that allows you to oversample a channel on the way in and downsample it on the way out. This is a good idea! If you have a feature like this one, the latency is usable, and it behaves well, **use it**. Because the channel performs oversampling before running the signal through your plugins, you should disable oversampling on each of the plugins in the channel. It's more flexible and convenient than manually resampling your files, making it more likely you can keep it as a good habit.

Fixing "Aliasing Creep" When Working at High Sample Rates

When working at high sample rates — whether you're using chain oversampling or simply a high sample rate — stacking a large number of plugins can still cause a little aliasing to crop up in measurements. For example, let's say you have some sonic material at 15 kHz that goes into a non-linear plugin running in a 192 kHz project. The output would have

sound at 30 kHz (the first harmonic, an octave). The sound at 30 kHz will likely be much quieter than at 15 kHz, but it's still there. If you stack the same plugin again, it will output sound at 60 kHz that is quieter still. Even though it's much quieter, you can see how this would eventually alias even at a very high sample rate. However, while you can see it, that aliasing is unlikely to be audible unless your plugin does some really nasty aliasing: each time it hops up an octave, it gets quieter.

If you hear this phenomenon or the measurement bothers you, you can put a linear low-pass filter above 20kHz after the offending plugin. This will prevent the aliasing noise from stacking, should not consume much CPU power, and does not require oversampling up and down a lot.

In the future, I'd suggest developers making chain oversampling solutions add an option that, when selected, automatically performs this process between each plugin on the chain. It will improve output characteristics without much performance or usability impact.

Are there Other Ways to Reduce Aliasing?

Yes! Oversampling is a generic way to eliminate aliasing. No matter the non-linear process, it will help. There are also ways to eliminate aliasing for specific types of processing, such as:

- Band-limited waveform generation for oscillators (BLEP, BLIT, MIP Map)
- Anti-derivative anti-aliasing

Developers can leverage these techniques with or without oversampling to mitigate aliasing problems that would show up in older designs, but by their nature, they still contain ripples from the Gibbs effect and can reduce headroom.

In Conclusion

Adding oversampling to individual audio plugins is a good, but imperfect solution to aliasing during production and mixing. Recent trends focus on oversampling as an important feature that will solve all aliasing issues without any downsides. But in reality, all solutions are imperfect, based on compromises, and a bit overhyped. By using sine sweeps to test your channel strips, you can make smart decisions about **when oversampling is necessary,** and by using a higher sample rate to begin with, you can **make oversampling less necessary** overall. Armed with these simple techniques, you can clamp down on aliasing without reducing your headroom or unnecessarily bogging down your computer. So go out there, and make some music!