Bachelor Thesis

Sample Rate Conversion for the Audio Doppler Effect

conducted at the

Signal Processing and Speech Communications Laboratory
Graz University of Technology, Austria

by

Jörg Hermann Müller

Supervisor:

Dipl.-Ing. Bernhard Geiger

Graz, September 16, 2011

Acknowledgement

At this point, shortly before finishing the thesis, I'd like to thank the contributors who were directly or indirectly involved in the creation of this thesis.

First of all I'd like to thank my family for being appreciative that I spent so much time with the project and had less time for them. Moreover they supported me also with any expenses and other things that came up during this time.

Then I want to thank my mentors Bernhard Geiger and Martin Poirier who quickly answered all my questions and helped with words and deeds with the thesis and the Google Summer of Code project.

Moreover I thank the Google Open Source Programs Office of Google Inc. for operating the awesome Google Summer of Code program, which enabled me to work on this project during this summer.

Last but not least I want to thank the Blender Foundation and the worldwide and local Blender community (notably Philipp Gosch) for the support around the Blender development like testing and other feedback.

Abstract

This bachelor thesis discusses sample rate conversion algorithms suitable for high quality audio Doppler effects calculation to be used in the open source 3D application Blender. The challenge is to have an algorithm that is able to dynamically change the resampling ratio over time. Apart from low order methods, an algorithm developed by Julius O. Smith is suitable for achieving this goal. The filter design necessary for this algorithm is discussed in detail. The introduced methods are then compared regarding different quality criteria. Finally, a practical experiment using the Doppler effect is implemented using Blender.

Contents

[Kurt looking at Sheldon who is dressed as the Doppler Effect]
***Kurt**: So what are you, a zebra?*
***Sheldon**: [to Leonard] Yet another child left behind.*

The Big Bang Theory - The Middle-Earth Paradigm

1 Introduction

1.1 Scope

During the Google Summer of Code 2011, my project was to implement 3D audio functionality for the open source 3D graphics program Blender (`http://www.blender.org/`). The basic 3D audio effects include distance-based attenuation, angle-based speaker output and the Doppler effect. For computer games these 3D audio effects are used, for example, through OpenAL, a realtime audio library. The goal of this project, however, was to be able to render 3D audio for computer generated animations, which in contrary to games have no realtime requirement and in return require a higher quality.

1.2 The Doppler Effect

The Doppler effect named after the Austrian scientist Christian Andreas Doppler is perceived as the change of sound pitch when the relative velocity between sound source and listener changes. Imagine an ambulance car with enabled siren emitting spheres at a constant rate which represent the wavefront of the siren sound and grow in size with the speed of sound. Now when the ambulance starts moving forward the distance between the spheres' surfaces gets smaller in the front of the car and bigger at the opposite side as shown in figure 1.1. So if someone stands in front of the car the sound waves of the car approaching get compressed, resulting in a higher frequency and if the car departs, the sound waves get stretched, sounding at a lower frequency. It's important to note that not only the frequency of the sound changes, but also its duration. If we imagine the siren doesn't create a continuous sound but only a short signal represented by only two or three spheres, it's not hard to understand that fact, if the first sphere represents the start and the last the end of the sound.

The relative frequency difference is expressed as [2]

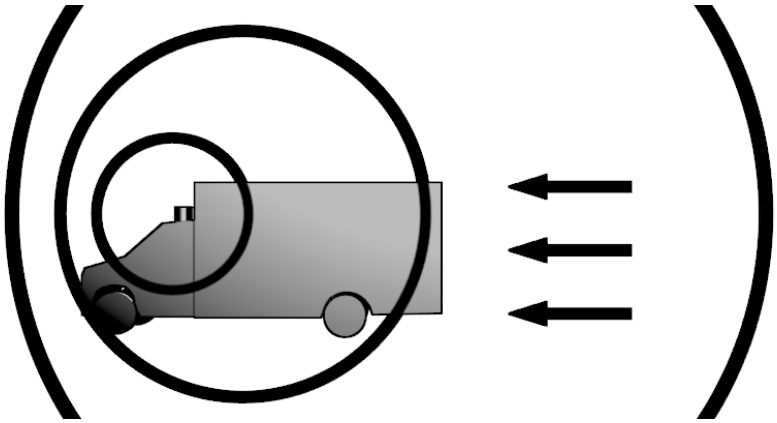


Figure 1.1: Visualisation of the Doppler effect.

$$\Delta f = \frac{c_{sound} - v_{ls}}{c_{sound} - v_{ss}}, \tag{1.1}$$

where c_{sound} denotes the speed of sound which is around 343.3 $\frac{\text{m}}{\text{s}}$ in air, depending on temperature and other physical parameters, and v_{ls} and v_{ss} denote the length of the velocity vector of the listener ($\vec{v_L}$) or the source ($\vec{v_S}$) projected on the vector from source to listener ($\vec{SL}$), evaluated by

$$v_{ls} = \frac{\vec{SL} \cdot \vec{v_L}}{\left|\vec{SL}\right|}, \tag{1.2}$$

$$v_{ss} = \frac{\vec{SL} \cdot \vec{v_S}}{\left|\vec{SL}\right|}, \tag{1.3}$$

as shown in figure 1.2.

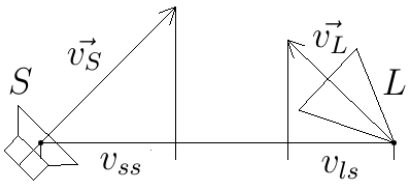


Figure 1.2: Visualisation of $\vec{v_S}$, $\vec{v_L}$, v_{ss} and v_{ls} of a source S and a listener L.

The perceived frequency f_p is the original frequency f_o multiplied with the relative frequency change Δf for which a value of 1 obviously means no change.

$$f_p = f_o \cdot \Delta f \tag{1.4}$$

In computer generated worlds like in movies or games this effect is computed to improve realism. Sound in the digital world is a discrete band-limited signal with

typical sampling rates of 44.1, 48, or 96 kHz for example. Changing the frequency (and as already explained also the length) of the signal – like one has to for the Doppler effect – equals changing the sampling rate ρ by the same factor Δf as one would change the frequency. Unfortunately typical audio output devices require a constant output sampling rate so applying the Doppler effect to audio we effectively have to use sample rate conversion. The main problem here is that the Doppler shift depends on the source and listener position and velocity as equation 1.1 shows. So a requirement for the used SRC algorithm is to support arbitrary time resampling which means that the original signal can be sampled at any required position (not necessarily in equidistant time positions), resulting in a fixed resampling ratio like typical resampling algorithms use.

I visualize a time when we will be to robots what dogs are to humans. And I am rooting for the machines.

Claude Elwood Shannon, Omni Magazine (1987)

2 Sample Rate Conversion

2.1 Introduction

Sample Rate Conversion or resampling is the process where we have a discrete-time signal with a specific sample rate and we would like to have the same signal sampled at a different sample rate. The theoretical way to do this is quite easy to understand: Create a continuous signal out of the discrete by *somehow* interpolating between two samples, filling the gaps and then sample again at the new rate. The difficult part for sure is the interpolation. Interpolation can be expressed mathematically as

$$x(t) = \sum_{n=-\infty}^{\infty} x[n] \cdot I(t-n) \tag{2.1}$$

where $I(t)$ is a continuous time kernel function that has to fulfill the condition

$$I(t) = \begin{cases} 1 & \text{if } t = 0 \\ 0 & \text{for all integers } t \neq 0 \end{cases} \tag{2.2}$$

to go exactly through all sample points. As $I(t)$ is time-invariant regarding $x(t)$ this interpolation is called local interpolation, contrary to a global interpolation like the Lagrange interpolation is. This is based on the assumption that the input signal is infinitely long with the result that a global interpolation is impractical to compute.

The ideal kernel function is the sinc function; the problem is, that the infinite sum is not practical to calculate, so a kernel function has to be found which is limited to a specific area around $t = 0$ (that means the function value is zero outside this area) so that the practical computation is possible [7].

2.2 Zero-Order-Hold

The most simple kernel function is the *rect* function, which is defined as

$$\mathrm{rect}(t) = \begin{cases} 1 & \text{if } |t| < \frac{1}{2} \\ \frac{1}{2} & \text{if } |t| = \frac{1}{2} \\ 0 & \text{if } |t| > \frac{1}{2} \end{cases}. \tag{2.3}$$

This method of interpolation is called Zero-Order-Hold (ZOH) and in graphics also known as nearest neighbor. In practice this interpolation mode may be altered a little to

$$I_0(t) = \begin{cases} 1 & \text{if } 0 \leq t < 1 \\ 0 & \text{otherwise} \end{cases}. \tag{2.4}$$

This essentially makes the interpolation filter causal, so the output of the interpolator is always the last input sample and jumps as soon as a new sample arrives at the input.

2.3 First-Order-Hold

An order higher than ZOH is First-Order-Hold, also known as linear interpolation, with the kernel function

$$I_1(t) = \begin{cases} 1 - |t| & \text{if } |t| < 1 \\ 0 & \text{otherwise} \end{cases}. \tag{2.5}$$

The result is a continuous signal without jumps, graphically the samples are connected with straight lines, but unfortunately the result still is not *smooth* (infinitely continuous differentiable) as there are hard corners at the original sample points.

2.4 Piece-wise Polynomial or Spline Interpolation

Raising the order of the interpolator and as such the interpolation polynomial further smooths the resulting signal approaching the *ideal* sinc-interpolator at an infinite order. Interpolators at a higher order than one are typically called piecewise polynomial interpolators, where the piecewise polynomial is also called spline. For symmetry reasons the order is typically chosen to be odd and most commonly cubic interpolation is used.

There are several different Spline types, like Hermite, Bezier and B-Splines, but here only cubic hermite splines will be discussed[1].

A cubic spline is described by a polynomial of 3$^{\mathrm{rd}}$ order, as the name says:

$$P(u) = a \cdot u^3 + b \cdot u^2 + c \cdot u + d, \tag{2.6}$$

[1] Spline basics are teached in basic computer graphics courses.

where $u \in [0, 1]$ denotes the position between two samples. It's possible to differentiate a polynomical infinitely often, so where we have to pay attention, are the transition points, where one spline is connected to another. The first-order differentials at the spline end points have to be the same as those at the adjacent ones. These requirements result in a system of four equations and four variables.

$$\begin{aligned} P_n(0) &= d &&= x[n] && (2.7)\\ P_n(1) &= a+b+c+d &&= x[n+1] && (2.8)\\ P_n'(0) &= c &&= m_n && (2.9)\\ P_n'(1) &= 3\cdot a + 2\cdot b + c &&= m_{n+1} && (2.10) \end{aligned}$$

Rewritten in matrix form

$$\begin{pmatrix} 0 & 0 & 0 & 1 \\ 1 & 1 & 1 & 1 \\ 0 & 0 & 1 & 0 \\ 3 & 2 & 1 & 0 \end{pmatrix} \cdot \begin{pmatrix} a \\ b \\ c \\ d \end{pmatrix} = \begin{pmatrix} x[n] \\ x[n+1] \\ m_n \\ m_{n+1} \end{pmatrix} \tag{2.11}$$

the equations are easy to solve:

$$\begin{pmatrix} a \\ b \\ c \\ d \end{pmatrix} = \begin{pmatrix} 2 & -2 & 1 & 1 \\ -3 & 3 & -2 & -1 \\ 0 & 0 & 1 & 0 \\ 1 & 0 & 0 & 0 \end{pmatrix} \cdot \begin{pmatrix} x[n] \\ x[n+1] \\ m_n \\ m_{n+1} \end{pmatrix}. \tag{2.12}$$

There are several ways of defining the slope m_n, for example

$$m_n = T_n \cdot (x[n+1] - x[n-1]), \; T_n \in [0, \infty) \tag{2.13}$$

describes the family of Catmull-Rom-Splines, while

$$m_n = \frac{1 - c_n}{2} \cdot (x[n+1] - x[n-1]), \; c_n \in [-1, 1] \tag{2.14}$$

describes the family of Cardinal Splines. Now the slope

$$m_n = \frac{1}{2} \cdot (x[n+1] - x[n-1]) \tag{2.15}$$

will be used which is part of both families ($T_n = \frac{1}{2}$ or $c_n = 0$).

With this slope the resulting polynomial is calculated by

$$\begin{aligned} P_n(u) = &\left(-\frac{1}{2}x[n-1] + \frac{3}{2}x[n] - \frac{3}{2}x[n+1] + \frac{1}{2}x[n+2]\right) u^3 \\ &+ \left(x[n-1] - \frac{5}{2}x[n] + 2x[n+1] - \frac{1}{2}x[n+2]\right) u^2 \\ &+ \frac{1}{2}\left(x[n+1] - x[n-1]\right) u + x[n]. \end{aligned} \tag{2.16}$$

To get the complete continuous signal $x(t)$ the splines have to be connected. Important is to remember that $u = t - n$ is limited to $[0, 1]$ for each polynomial.

$$x(t) = \sum_{n=-\infty}^{\infty} P_n(t-n) \cdot \mathrm{rect}(t - n - \frac{1}{2}) \tag{2.17}$$

Equation 2.17 doesn't look like equation 2.1, but the latter is a convolution between the discrete-time $x[n]$ and the continuous $I(t)$. To get $I(t)$ out of equation 2.17 it's possible to evaluate it with a Dirac impulse $\delta[n]$ as input to get

$$I_3(t) = \begin{cases} \frac{1}{2}(t+2)^3 - \frac{1}{2}(t+2)^2 & \text{if } -2 \leq t < -1 \\ -\frac{3}{2}(t+1)^3 + 2(t+1)^2 + \frac{1}{2}(t+1) & \text{if } -1 \leq t < 0 \\ \frac{3}{2}t^3 - \frac{5}{2}t^2 + 1 & \text{if } 0 \leq t < 1 \\ -\frac{1}{2}(t-1)^3 + (t-1)^2 - \frac{1}{2}(t-1) & \text{if } 1 \leq t < 2 \\ 0 & \text{otherwise} \end{cases} \tag{2.18}$$

as kernel function, which can also be interpreted as impulse response of a filter.

2.5 Polyphase Filter

Getting away from equation 2.1 now, we take a closer look at other sample rate conversion algorithms. The standard sample rate converter with a resampling ratio of $\frac{L}{M}$ tought in digital signal processing courses consists of three parts: A factor L upsampler (signal processing literature uses the term interpolator here, which just means increasing the sample rate by filling up unknown samples with zeros), a lowpass filter with gain L and cutoff $\min\left(\frac{\pi}{L}, \frac{\pi}{M}\right)$ and a factor M downsampler (also called decimator, simply throwing away samples) [5].

Using multirate signal processing it is possible to design fast polyphase interpolation filters. The problem this design principle has is that the resampling ratio has to be a fixed rational while Doppler audio effects require a dynamically changing ratio with which it's possible to resample to any arbitrary time of the signal possibly with a different rate for each sample.

2.6 Discrete Fourier Transform

Equally important is the sample rate conversion in the frequency domain. This algorithm uses the N-point DFT and the $N_1 = \frac{LN}{M}$-point IDFT to do the resampling in the frequency domain. This technique requires some block processing and overlapping as well as FFT algorithms for the N-point DFT and N_1-point IDFT to be fast and practical [1]. However, this technique suffers from the same problem as the polyphase filter technique, not being suitable for the Doppler effect use case.

2.7 Julius O. Smith's Algorithm

Julius O. Smith has developed an algorithm that is able to resample at arbitrary time values t [8].

For a first explanation we will go back to our beloved equation 2.1 and the ideal sinc lowpass filter as kernel function. The reason why we cannot use this equation in practice is that the sinc function is supported from $-\infty$ to ∞, so what we have to do is to window the sinc function to get a kernel function with limited support.

If one implemented the algorithm so far, he would have noticed that the algorithm cannot compete with any of the others regarding speed, because the evaluation of the windowed sinc is too slow to create a practically usable filter. For this reason we use a look-up-table (LUT) for the filter coefficients, which luckily only has to have half the length, as the filter is symmetrical. Between two entries in the LUT the value is interpolated linearly.

The resulting resampler is a high-quality arbitrary time resampler perfectly suitable for the Doppler effect simulation.

Algorithms are the computational content of proofs.
Robert Harper, Benjamin C. Pierce et al. Software Foundations

3 Implementation

3.1 Smith's Algorithm

3.1.1 Definitions

- $x[n]$ is the input signal indexed by the positive integer n; $x[m] = 0 \; \forall \; m < 0$.
- $h[l]$ is the look-up-table with the filter coefficients indexed by the positive integer l, where $h[l] = 0 \; \forall \; l > l_{\max}$.
- L is the count of look-up-table entries representing the same length as between two input samples, so a length of L in the filter table equals a length of 1 in the input signal. For the ideal sinc function L would denote the number of samples between zero-crossings, resulting in $h[l] = \text{sinc}(\frac{l}{L})$.
- ρ is the resampling ratio $\rho = \frac{F_{s,\text{out}}}{F_{s,\text{in}}}$.
- $\{v\}$ denotes the fractional part of the real number v, while $\lfloor v \rfloor$ denotes the truncated part of v, resulting in $v = \lfloor v \rfloor + \{v\}$.
- o is the truncated part of the positive input sample time t, $o = \lfloor t \rfloor$.
- P is the fractional part of the input sample time t, so $t = o + P$ and $P = \{t\}$.

3.1.2 Algorithm

The algorithm evaluates equation 2.1 by the use of a symmetric FIR lowpass filter in a linearly interpolated look-up-table. Looking at the convolution, which is a sum of scaled and shifted impulse responses (which is the kernel function) it's obvious that a single

output sample is calculated by a sum of scaled filter coefficients. As the look-up-table only has to save half of the symmetric filter, the kernel function turns into

$$I_{\mathrm{JOS}}(t) = h\left[\lfloor |t| \cdot L \rfloor\right] \cdot (1 - \{|t| \cdot L\}) + h\left[\lfloor |t| \cdot L \rfloor + 1\right] \cdot \{|t| \cdot L\}\,. \tag{3.1}$$

With the definitions above we get

$$x(t) = \sum_{n=-\infty}^{\infty} x[n] \cdot I(t-n) = \sum_{n=-\infty}^{\infty} x[n] \cdot I_{\mathrm{JOS}}(o + P - n). \tag{3.2}$$

To speed up the computational evaluation of the above equation, we will first split up the sum into two sums, to get rid of the absolute value bars during the evaluation of $I_{\mathrm{JOS}}(t)$, resulting in

$$x(t) = \sum_{n=-\infty}^{o} x[n] \cdot I_{\mathrm{JOS}}(o + P - n) + \sum_{n=o+1}^{\infty} x[n] \cdot I_{\mathrm{JOS}}(n - o - P). \tag{3.3}$$

With index transformations $m = o - n$ for the first sum and $m = n - o - 1$ for the second, this turns into

$$x(t) = \sum_{m=0}^{\infty} x[o-m] \cdot I_{\mathrm{JOS}}(m + P) + \sum_{m=0}^{\infty} x[m + o + 1] \cdot I_{\mathrm{JOS}}(m + 1 - P), \tag{3.4}$$

which reveals a reason for splitting up t into $o + P$, as only the relative index m is important for the calculation of the local kernel $I_{\mathrm{JOS}}(t)$.

3.1.3 Aliasing

During downsampling ($\rho < 1$) we have to avoid aliasing. The first idea might be to calculate a new filter look-up-table, which is really impractical. But as the kernel function is a continuous function, we can use the time scaling Fourier transform property [6]

$$f(at) \leftrightarrow \frac{1}{|a|} F\left(\frac{\omega}{a}\right), \tag{3.5}$$

which is exactly what we need, as our ideal sinc interpolator is an ideal lowpass in the frequency domain with a normalized cutoff frequency of 1. Scaling the frequency axis by the factor ρ results in the cutoff frequency being moved to ρ which is exactly what we need to avoid aliasing.

$$\rho \cdot I_{\mathrm{JOS}}(\rho t) \leftrightarrow \mathcal{F}(I_{\mathrm{JOS}})\left(\frac{\omega}{\rho}\right) \tag{3.6}$$

So for $\rho < 1$ we rewrite equation 3.4 to

$$x(t) = \rho \cdot \left(\sum_{m=0}^{\infty} x[o-m] \cdot I_{\mathrm{JOS}}\left((P+m)\,\rho\right) + \sum_{m=0}^{\infty} x[m+o+1] \cdot I_{\mathrm{JOS}}\left((m+1-P)\,\rho\right) \right). \tag{3.7}$$

3.2 Optimization

This section will discuss some optimizations specific to the C++ implementation of the algorithm, so first some properties of the implementation have to be covered: The algorithm is implemented in an audio library that uses single-precision floating point representation for the sample values. Moreover the signal can be multi-channel with sample-wise interleaved channels. For this reason the coefficient table also saves single-precision floating point values. It is also important to note, that the implementation is for modern PCs with heavily pipelined CPUs instead of micro controllers or signal processors, so some optimizations are specific to this architecture.

3.2.1 Quality Optimizations

For the calculation of the sum when using integral data types (this includes fixed-point as fixed-point is only a bit-shifted integral data type) we would have to use a bigger data type for the sum than for the summands to avoid overflows. This is basically not required for floating point data types, but we would lose precision. Thus the summation itself is done using double precision floating point.

Moreover, while the sum over integral data types is accurate in the value domain of the data type, this is not true for floating point data types, especially when adding small values to big ones (it can happen that the small values have no effect). For this reason it is better to sum small values first. We can optimize the implementation of the algorithm in this manner summing in a reversed order starting at the end of the LUT, as the values there are far lower than the values at the beginning, considering the sinc falls off.

The implementation can produce multiple output samples at once, which is faster than calling the implementation for each single output sample repeatedly. However, the resampling ratio can only be chosen once per call. So an optional optimization is to linearly interpolate the resampling ratios between succeeding calls of the implementation, which results in a smoother frequency change for the targeted Doppler effect.

3.2.2 Speed Optimizations

A first optimization which has been suggested by Smith is to use a second LUT which saves the difference of two succeeding filter coefficients to optimize the linear interpolation of the filter coefficients. However, this doubles the memory requirements, so one might not want to implement this optimization for modern PCs as the improvements will be hardly observable due to the pipelining CPUs.

Looking at equation 3.1 and equation 3.4 for $\rho \geq 1$, we can see that during the sum over m the linear interpolation factors $\eta_1 = \{(m + P) \cdot L\}$ and $\eta_2 = \{(m + 1 - P) \cdot L\}$ are evaluated. As m and L are integers, this can be optimized, for example to

$$\eta_1 = \{m \cdot L + P \cdot L\} = \{P \cdot L\}, \tag{3.8}$$

a value that doesn't change during the evaluation of the sum.

While the summation, signal and filter coefficient values are all floating point in the implementation, the values used for calculating the LUT index doesn't have to, so we can optimize this by using fixed point calculation here, which is not only faster, but also saves us from expensive floating point rounding/conversion operations, which is a simple bit operation for fixed point.

Moreover as fixed point calculations are accurate as stated above, the related multiplications with the summation index that would have to be evaluated often during the summation can be turned into incremental additions, which are a lot faster.

At one point however, the floating point values have to be converted to fixed point ones. For this it is important to use a fast floating point to integer conversion that doesn't clear the floating point pipeline of the CPU by changing the rounding mode. This function is called `lrint`, a C99 standard [4] function which rounds using the current rounding mode.

Last but not least important, the innermost loop of the implementation runs over the different channels in the signal. A huge optimization can be achieved by having specialized implementations for frequently used channel counts like mono and stereo, as the overhead produced by looping over a dynamic channel count is quite big.

3.3 Filter Design

Using the look-up-table for the algorithm it's possible to use different filters. However typical optimal discrete FIR filter design algorithms like the Parks-McClellan algorithm are not usable, due to the high filter order that would be required.

So what remains is the window filter design method, where we can choose from many windows available, like rectangular, Bartlett, Hamming, Blackman-Harris, Poisson, and Hann-Poisson. As we would like to build the filter based on a design specification we will examine Kaiser and Dolph-Chebyshev windows.

First of all we will have a look at the length $l_{\max}$ of the coefficient table, which is determined by L and the number of zero-crossings in case of a windowed ideal sinc filter with a normalized cutoff frequency at 1. As all window functions have a main lobe with a specific bandwidth, we design our sinc filter to have a normalized cutoff frequency f_c of exactly that main lobe bandwidth below the ideal cutoff frequency $f_o = 1$. The third parameter is the actual count of zero-crossings N_z at one side of the symmetric filter response, resulting in a LUT length of

$$l_{\max} = \left\lfloor \frac{L \cdot N_z}{f_c} \right\rfloor. \tag{3.9}$$

Working with digital computers having size-limited binary data types, we should consider filter design based on these to get a filter quality with an error as small as the quantisation error. Regarding this, Smith [8] has two statements for us:

1. Choice of Table Size: $L \geq 2^{n_c/2}$ where n_c is the number of bits per table entry.
2. Choice of Interpolation Resolution: $n_\eta \geq \frac{n_c}{2}$, where n_η is the number of bits available for the linear interpolation factor (the fractional part of the argument of $I_{\mathrm{JOS}}(t)$).

These statements are valid for fixed-point data types, while when using floating-point n_c can be estimated by the number of bits in the fractional part, as the lower limit and the upper limit, being the number of bits for the whole floating-point value.

But the data type size of other variables during the computation is also important, like for example the size n_P (in bit) of the fractional part of the time argument of $I_{\mathrm{JOS}}(t)$, which limits the LUT length by $n_P = n_\eta + n_l$, where n_l is the count of bits available for the LUT index l.

Last but not least the size of the actual signal data type n_s (again in bit) is important because it determines the accuracy of the sample values, where each bit adds approximately 6 dBFS (decibels relative to full scale) to the accuracy, resulting in a accuracy of

$$\mathrm{A} = 20 \log_{10} (2^{n_s}) \tag{3.10}$$

in dBFS, which basically tells the side lobe level of the windows.

Now back to the windows, the window size parameter

$$M = 2 \cdot l_{\max} - 1 \tag{3.11}$$

is equal for all window types.

3.3.1 Kaiser Window

The Kaiser window which is based on the modified Bessel function has a parameter β which controls the (positive) side lobe level A and main lobe bandwidth ω_m (in radians). The lower the bandwidth, the higher the side lobe level and vice versa.

The connection between side lobe level A in dB and β has been determined empirically as [5]

$$\beta = \begin{cases} 0.1102\,(A - 8.7) & \text{if } A > 50 \\ 0.5842\,(A - 21)^{0.4} + 0.07886\,(A - 21) & \text{if } 21 \leq A \leq 50\,. \\ 0 & \text{if } A < 21 \end{cases} \tag{3.12}$$

β is also quarter the time-bandwidth product [9],

$$\beta = \frac{1}{4} M \omega_m, \tag{3.13}$$

in radians.

We would like to know the ideal cutoff frequency $f_c = 1 - \frac{\omega_m}{2\pi}$, to reach the stopband exactly at $f = 1$. We have to use $M = \frac{2N_z}{f_c}$ instead of the M we use for the final window calculation, as equation 3.13 is valid for a discrete FIR filter, while ours basically is subsampled by the factor L.

So after some calculation we get

$$\beta = \frac{1}{4} \cdot \frac{2N_z}{f_c} \cdot 2\pi \left(1 - f_c\right) \Rightarrow f_c = \frac{N_z \pi}{N_z \pi + \beta} \tag{3.14}$$

to calculate the perfect cutoff frequency f_c for our kaiser windowed sinc filter.

3.3.2 Dolph-Chebyshev Window

The Dolph-Chebyshev window has a parameter r which controls the side lobe ripple magnitude which is equal to the side lobe level. It optimizes the frequency response of the window where all side lobes have the same magnitude allowing for the smallest main lobe width ω_m. For small main lobe widths ($\omega_m \ll 2\pi$), the equation [9]

$$M \approx 1 + \frac{4}{\omega_m} \cosh^{-1}\left(\frac{1}{r}\right) \tag{3.15}$$

links window size M, main lobe width ω_m and ripple magnitude r [9], so with the same equations for M and ω_m as with the kaiser window we get an optimal cutoff frequency at

$$f_c = N_z + \frac{1}{2} + \frac{\cosh^{-1}\left(\frac{1}{r}\right)}{\pi} - \sqrt{\left(N_z + \frac{1}{2} + \frac{\cosh^{-1}\left(\frac{1}{r}\right)}{\pi}\right)^2 - 2N_z}. \tag{3.16}$$

3.3.3 Example Filter Design

For the practical implementation we want to design the highest-quality filter possible using a Dolph-Chebyshev window. For the single-precision floating point data types we assume an accuracy of $n_c = 24\,\text{bit}$ for the filter coefficients bitdepth (so that L and n_η don't get too big) and $n_s = 27\,\text{bit}$. For n_P we have a fixed point floating point data type with 32 bit.

With all equations from this section, we can calculate the parameters:

$$n_\eta = \frac{n_c}{2} = 12 \text{ bit} \tag{3.17}$$

$$L = 2^{n_c/2} = 4096 \tag{3.18}$$

$$r = \left(\frac{1}{2}\right)^{n_s} = 7.4505806 \cdot 10^{-9} = -162.5562 \text{ dBFS} \tag{3.19}$$

$$n_l = n_P - n_\eta = 20 \text{ bit} \tag{3.20}$$

$$l_{\max} = 2^{n_l} = 1048576 \tag{3.21}$$

Now with equations 3.9 and 3.16 we can calculate the missing two parameters N_z and f_c. Both equations contain both variables, but instead of solving this equation system, we first calculate N_z with the normalized cutoff frequency $f_{\hat{c}} = 0.95$ which is definitely lower than the final f_c, for the reason of simplicity and N_z linearly influencing the runtime and memory usage of the algorithm.

$$N_z = \left\lfloor \frac{l_{\max} \cdot f_{\hat{c}}}{L} \right\rfloor = 243 \tag{3.22}$$

$$f_c = 0.97776 \tag{3.23}$$

The filter generated with these parameters is a bit exaggerated, especially N_z can be chosen smaller, having a linear influence on the runtime as already stated.

When a tree falls in a lonely forest, and no animal is near by to hear it, does it make a sound? Why?

Charles Riborg Mann, George Ransom Twiss, Physics (1910)

4 Comparison

4.1 Test System

The following tests have been set up to compare the discussed sample rate converters, which were all implemented in C++.

Impulse Response Shows the impulse response and the frequency response.

Dynamic Ratio Checks how good the arbitrary resampling ratio works.

Sweep Chirp signal to check frequency response and basic aliasing.

Aliasing Downsamples a high frequency signal that should get filtered completely.

Harmonic Distortion Harmonics are frequency responses at frequency multiples of a basic frequency that are created in non-linear systems.

Intermodulation Distortion Also a result of non-linear systems, where inputs with two or more different frequencies result in harmonics that combine the original frequencies.

4.2 Method

All tests use RIFF wave files with single-precision floating point sample data. The input files were created with the open source audio editor *audacity*.

4.2.1 Impulse Response

A signal with a sample rate of 96 kHz is downsampled to 44.1 kHz. The signal $x[n]$ consists of $\frac{96000}{\gcd(96000,44100)} = 320$ unit impulses with enough space between them. The impulses are at sample positions n so that $\{n \mod 320\}$ is a permutation of $\{0, 1, 2, \ldots, 319\}$. This basically means that all fractional differences in sample positions between input and output signal are addressed exactly once. In the later analysis the output signal is upsampled (by inserting zeros) to $\mathrm{lcm}(96000, 44100) = 14.112$ MHz and the impulse responses are added to obtain a high resolution impulse response, which is also inspected in the frequency domain in phase and magnitude utilizing the DFT.

4.2.2 Dynamic Ratio

A sine wave at a frequency of 5 kHz with a sampling rate of 48 kHz is resampled to a linear chirp between 100 Hz and 20 kHz over 10 seconds simulating the Doppler effect of a passing object. For analysis the spectogram of the resulting file is plotted.

4.2.3 Sweep

A sweep signal with a linear slope of 1 kHz/s from 1 Hz to 48 kHz at a sample rate of 96 kHz is downsampled to 44.1 kHz. Again the spectogram is plotted for analysis to evaluate aliasing effects and filter cutoff.

4.2.4 Aliasing

A 23 kHz sine at -4 dBFS with a white noise floor of -150 dBFS over 30 seconds is downsampled from 96 kHz to 44.1 kHz. The resulting signal is analysed using Welch's method with 4 segments, an overlap of 50 %, and a Hann window.

4.2.5 Harmonic Distortion

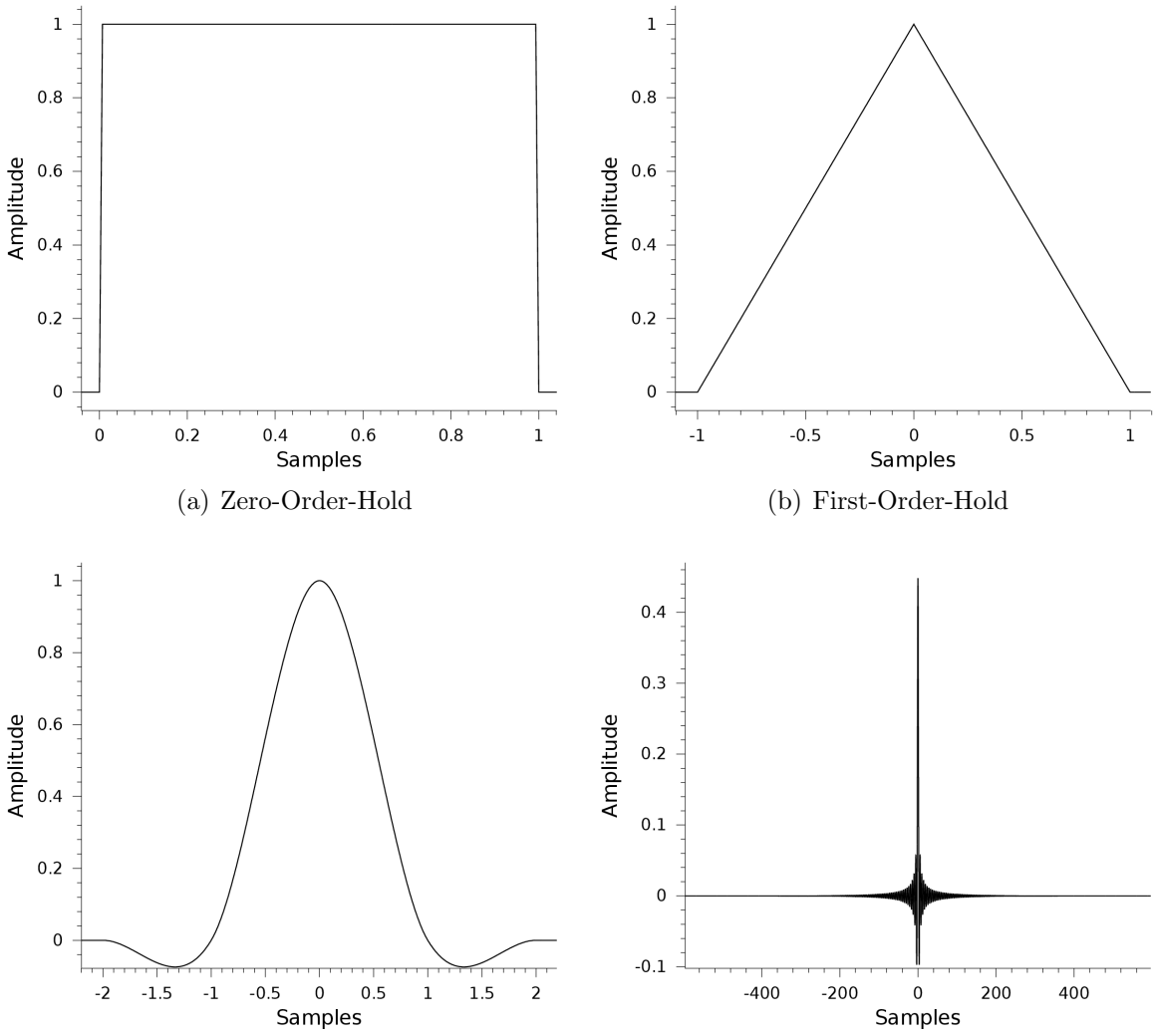
Equivalent to the Aliasing test, with the only difference that the sine has a frequency of 1 kHz.

4.2.6 Intermodulation Distortion

Again equals the Aliasing test with the difference that the sine is replaced by two sines, one at 60 Hz, -6 dBFS and the second at 7 kHz, -18.0412 dBFS, which equals quarter the amplitude of the first sine.

4.3 Discussion

4.3.1 Impulse Response



(a) Zero-Order-Hold

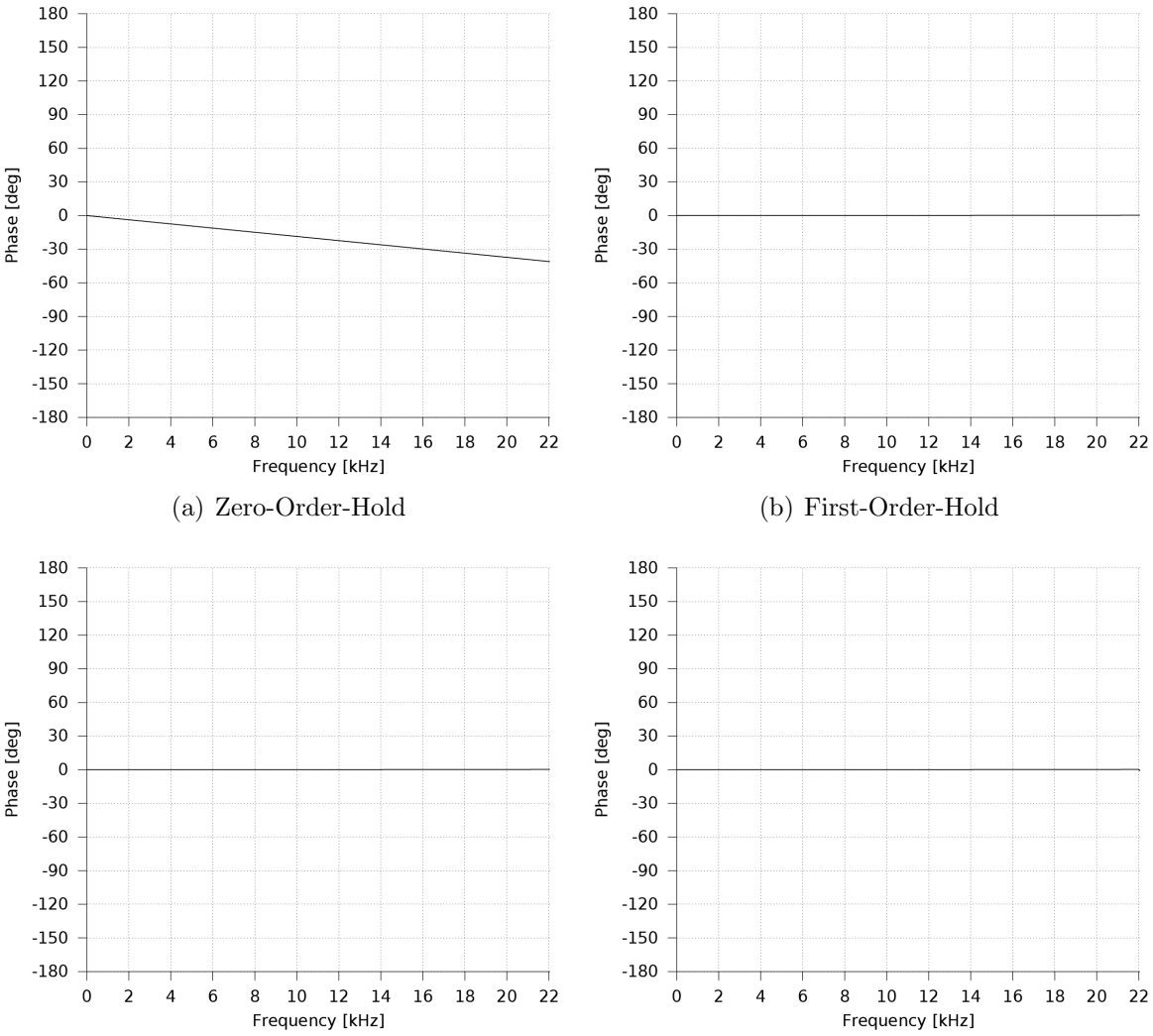
(b) First-Order-Hold

(c) Hermite Spline

(d) Windowed Sinc

Figure 4.1: Impulse response in the time domain.

Figure 4.1 shows the impulse responses in the time domain. As we can see, the higher the order gets, the more the response looks like the sinc function. It is also important to notice, that all responses are symmetric around 0, except the Zero-Order-Hold response in figure 4.1(a), which is shifted. This behaviour can also be seen in the phase in figure 4.2 where all symmetric filters are zero phase, except the ZOH filter, which is linear phase.



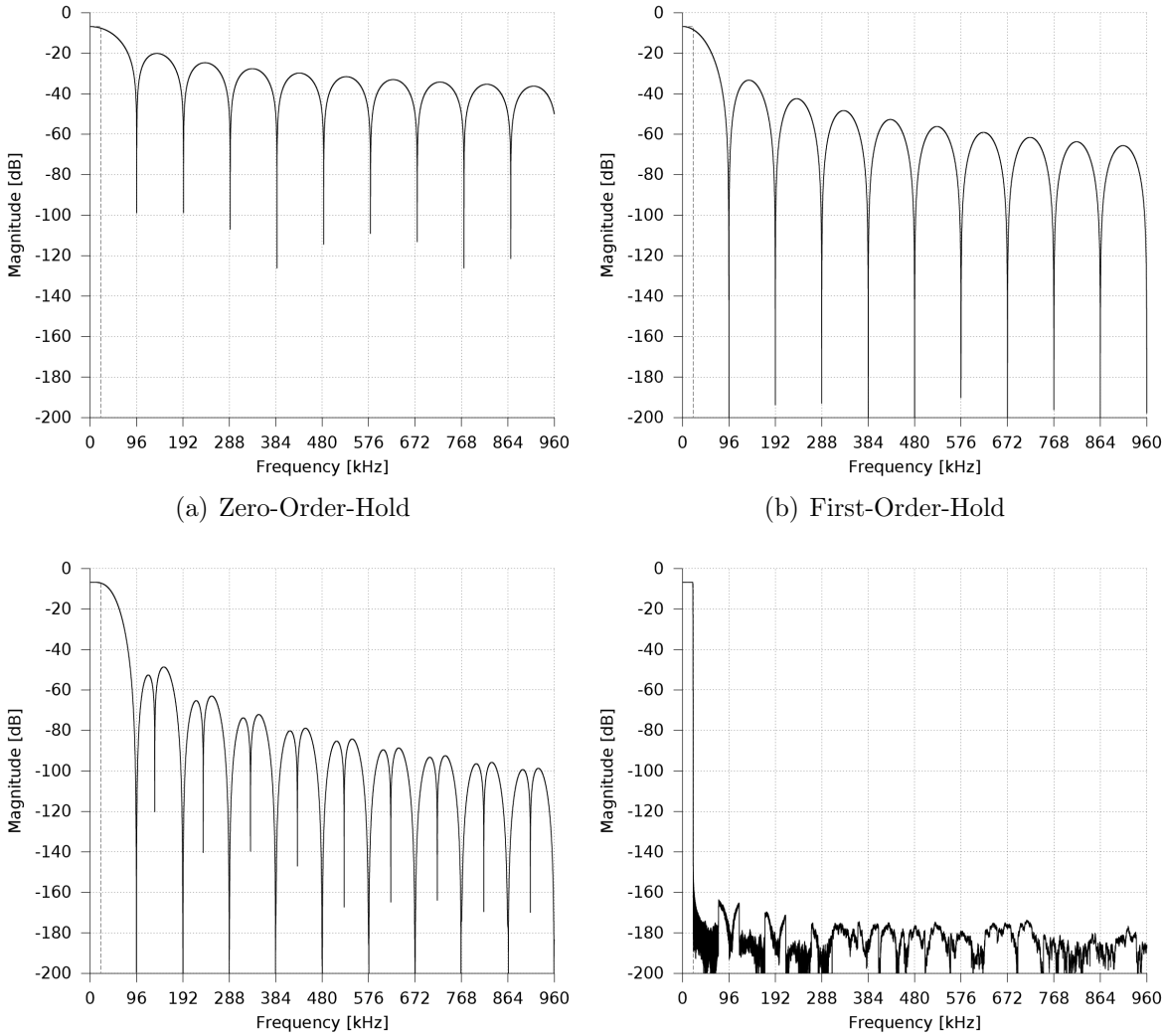
(a) Zero-Order-Hold

(b) First-Order-Hold

(c) Hermite Spline

(d) Windowed Sinc

Figure 4.2: Phase of the frequency response.



(a) Zero-Order-Hold

(b) First-Order-Hold

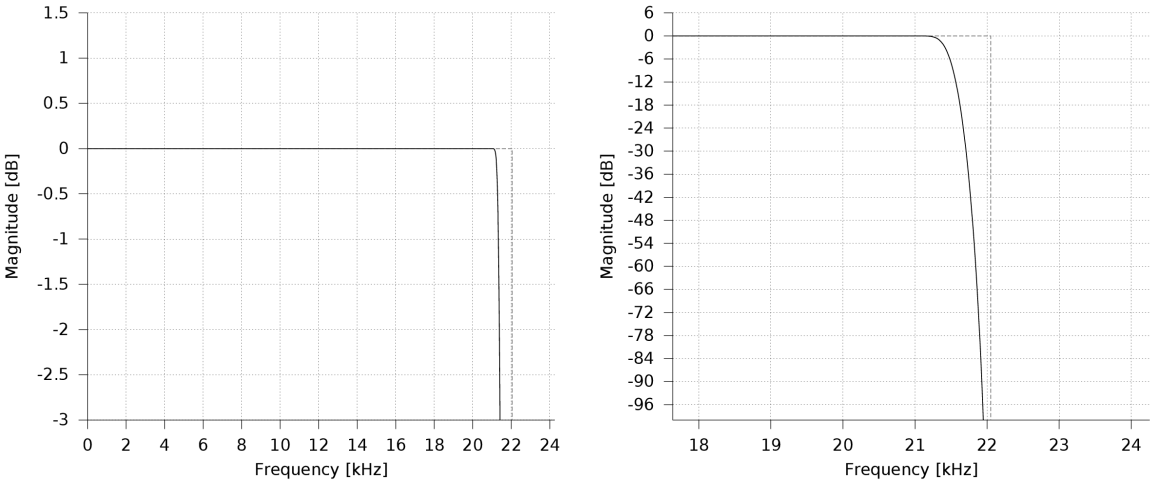
(c) Hermite Spline

(d) Windowed Sinc

Figure 4.3: Magnitude of the frequency response.

The magnitude of the frequency response is displayed in figure 4.3, where we can see that the first three filters have a very gentle overall fall-off, so we can expect quite some aliasing with these filters. In contrary to that, the high-quality algorithm has a very good fall-off to a stopband of approximately $-160\,\mathrm{dBFS}$. For the lower-order filters we notice the roots at multiples of the sampling frequency. These are easily explained since the Fourier transform of the ZOH rect window is a sinc with its roots exactly at the multiples of the sampling frequency. Also notice that the higher the order gets, the better is the fall-off for higher frequencies.

Taking a closer look at the magnitude of the high-quality filter in figure 4.4 we can see that there are no significant ripples in the passband and a quite steep fall-off in the transition band which starts around $21.25\,\mathrm{kHz}$.



(a) Passband (b) Transition

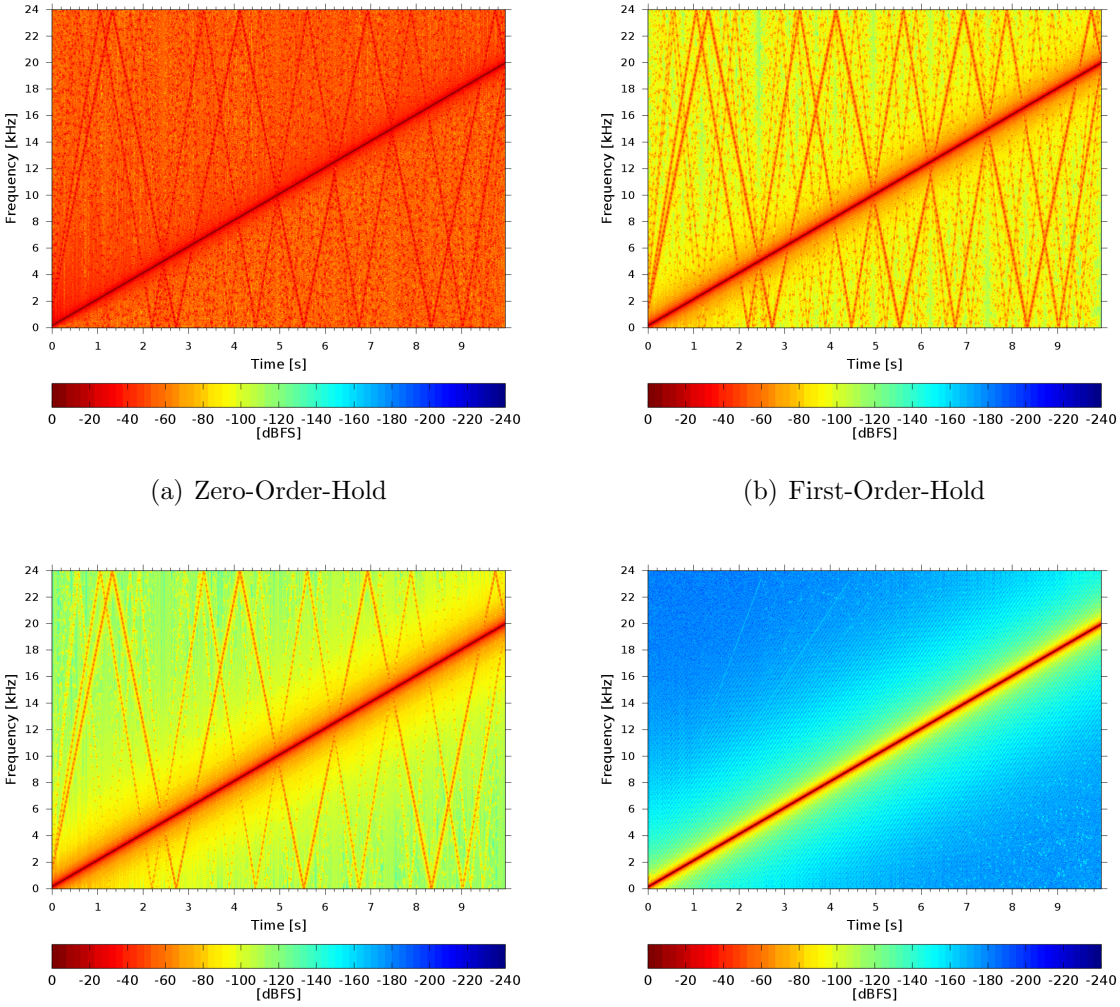
Figure 4.4: Magnitude of the frequency response of the windowed sinc.

4.3.2 Dynamic Ratio

In figure 4.5 we can see the results of the Doppler simulation tests. Apart from the overall background level and the signal line in the spectogram we can see some zigzag lines that are frequencies stronger than the background level. These are caused by aliasing effects and are easily explainable with the resampling behaviour in the DFT domain.

Let's examine, for example, the time in the plot when the signal frequency is 3 kHz: Our original signal is a 5 kHz sine at a sampling ratio of 48 kHz, so to get the 3 kHz at this time, the resampling ratio is $\frac{5}{3}$, which equals upsampling to 80 kHz. Applying the DFT on the input signal we know that the frequency spectrum gets copied to every multiple of the sampling frequency. So the 5 kHz signal produces a DFT spectrum which also contains the frequencies 43, 53, 91, 101 kHz and so on. These frequencies would get filtered out by the ideal lowpass, but as we don't have an ideal lowpass (see figure 4.3), they don't. Now what happens is that these frequencies get mirrored into the frequency response of the resampled signal, for example 43 kHz produces an aliasing image at 37 kHz and 53 kHz at 27 kHz. Considering that we interpret the output signal at the same sampling rate as the input signal, these frequencies have to be multiplied by the inverse resampling ratio $\frac{3}{5}$ turning them into 22.2 kHz and 16.2 kHz, which are exactly the frequencies of these zigzag lines at this particular time.

The higher the mirror frequencies, the better they get filtered, resulting in the order dependant background level.



(a) Zero-Order-Hold

(b) First-Order-Hold

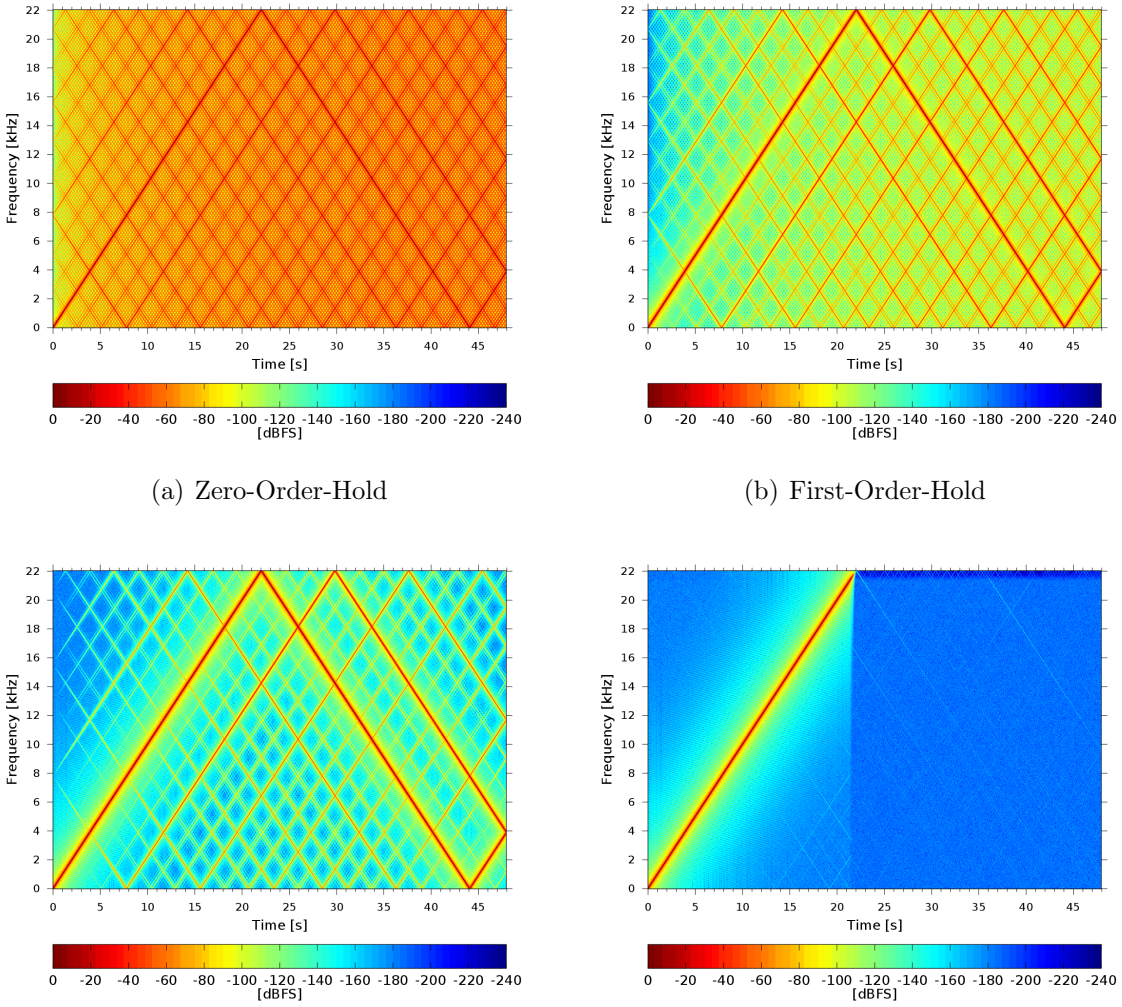
(c) Hermite Spline

(d) Windowed Sinc

Figure 4.5: Dynamic Ratio test result.

4.3.3 Sweep

The sweep test results shown in figure 4.6 look similar to figure 4.5, the Dynamic Ratio results, where the resampling ratio changed to produce a sweep based on a fixed frequency signal. This time however the resampling ratio is fixed, while the signal frequency changes. We can see the same aliasing artifacts with the difference that due to the fixed resampling ratio the artifacts have the same slope as the original signal. Especially the results of the steep fall-off of the high-quality filter can be observed in figure 4.6(d), while due to the gentle fall-off of the others the aliasing artifacts disappear only very slowly with increasing frequency.



(a) Zero-Order-Hold

(b) First-Order-Hold

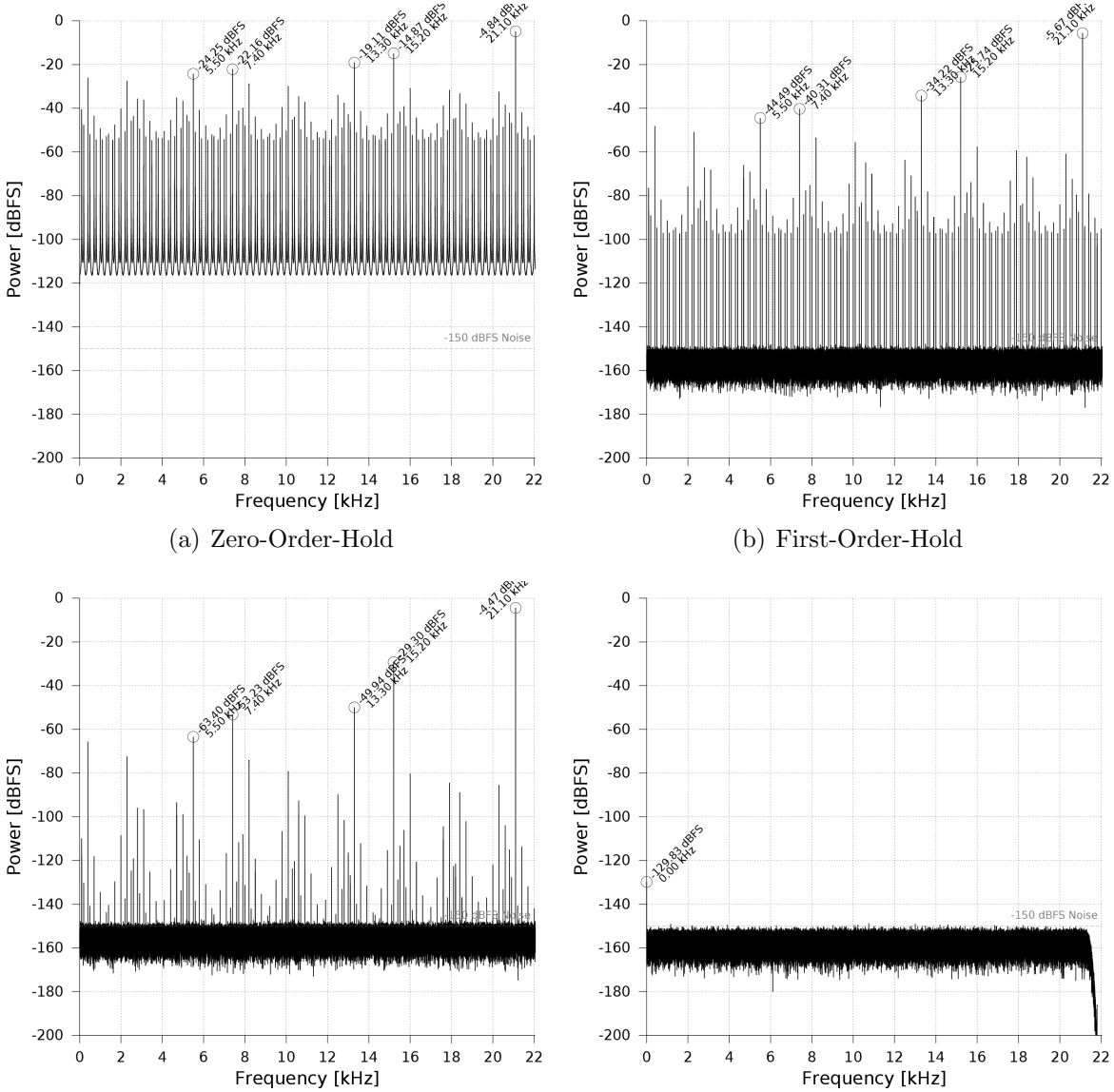
(c) Hermite Spline

(d) Windowed Sinc

Figure 4.6: Sweep test result.

4.3.4 Aliasing

For the aliasing test we expect mirror frequencies of the 23 kHz tone. The first five are 21.1, 15.2, 13.3, 7.4 and 5.5 kHz. These are exactly the highest peaks we can find in figure 4.7. As we expect ZOH in figure 4.7(a) is the worst, where all frequencies are above the −150 dBFS noise level of the original signal. For the first mirror frequency FOH (−5.67 dBFS, figure 4.7(b)) is even better than the spline interpolation (−4.47 dBFS, figure 4.7(c)). This is because the frequency response has a more gentle fall-off at first as seen in figure 4.3(c), but for higher frequencies the spline interpolator apparently damps better. The high-quality windowed sinc shows the best results with none of the mirror frequencies. However there is a peak at 0 Hz which first of all doesn't matter actually as a direct component isn't audible in an audio signal anyway. It is an artifact



(a) Zero-Order-Hold

(b) First-Order-Hold

(c) Hermite Spline

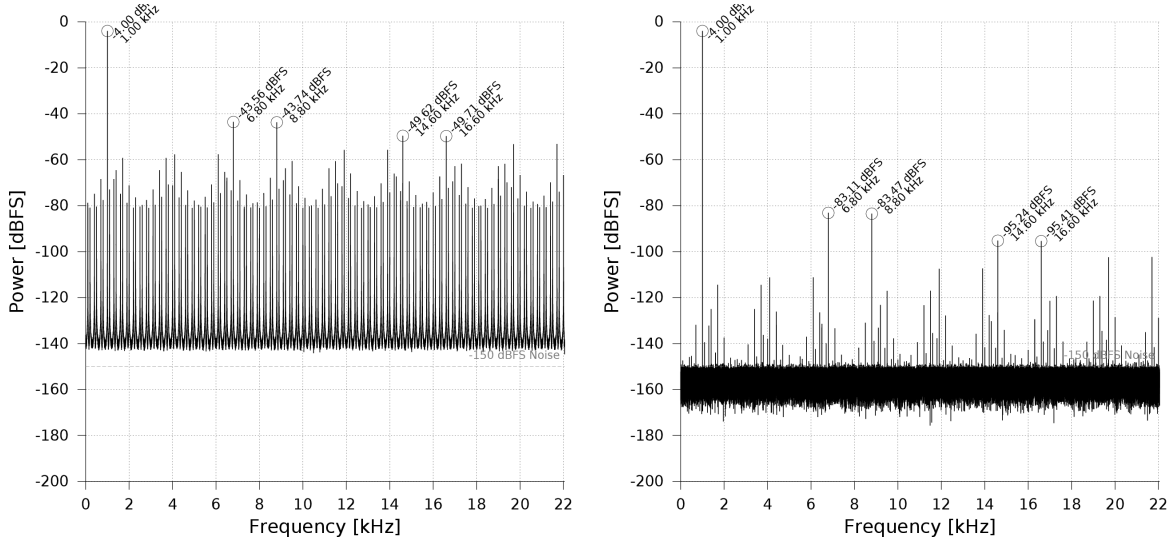
(d) Windowed Sinc

Figure 4.7: Aliasing test result.

at the beginning and end of the audio file caused by the fact that the input signal is not an infinte, but a rect windowed sine. So the size of this artifact actually depends on the segment count of the Welch method and could be prevented by smoothing the amplitude of the input signal at the file margins.

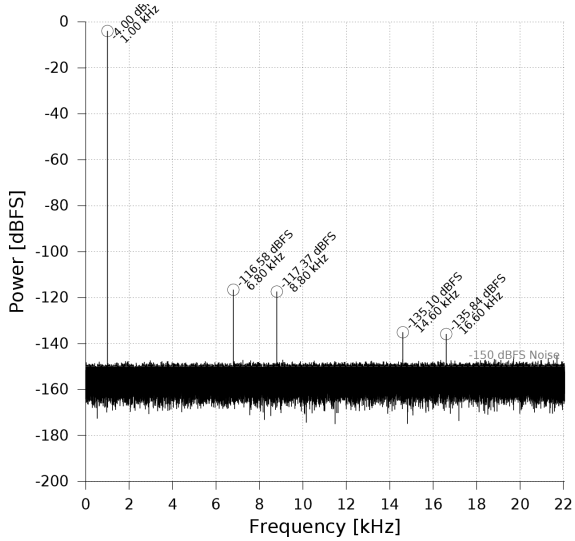
4.3.5 Harmonic Distortion

Looking for harmonics, that are multiples of the 1 kHz signal frequency in figure 4.8, we only discover highest peaks at 6.8, 8.8, 14.6 and 16.6 kHz, which are not harmonics, but peaks caused by aliasing. The high-quality resampler doesn't show any undesired peaks.

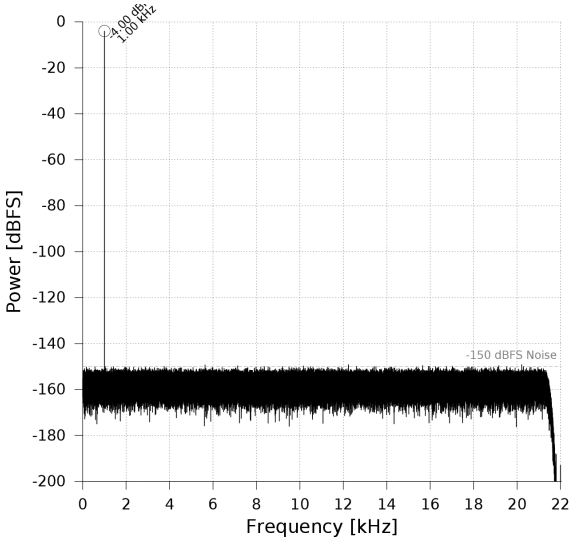


(a) Zero-Order-Hold

(b) First-Order-Hold



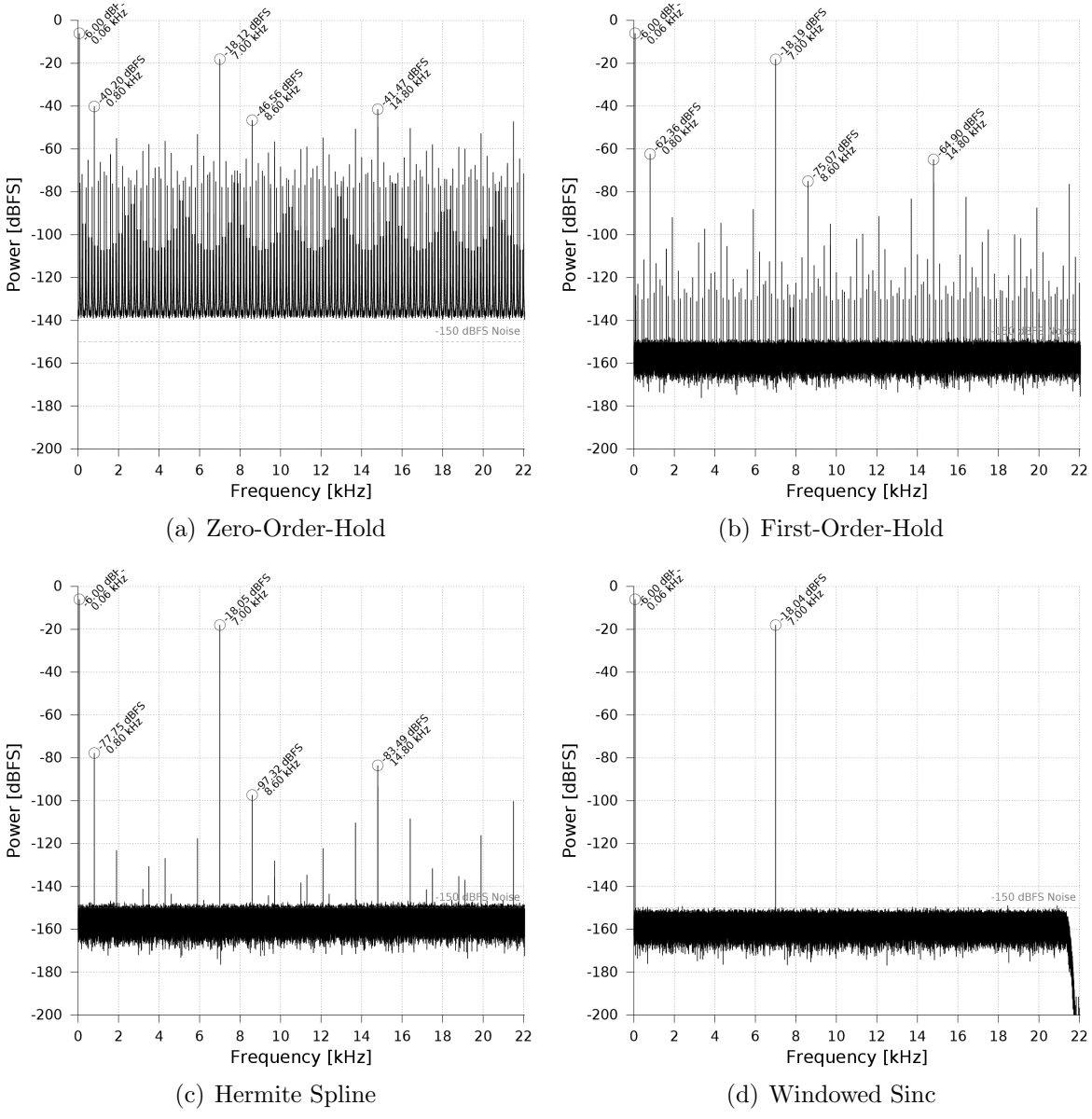
(c) Hermite Spline



(d) Windowed Sinc

Figure 4.8: Harmonic Distortion test result.

It's also remarkable that the spline interpolator in figure 4.8(c) has it's highest aliasing peak below −100 dBFS, which means that for default 16 bit audio the spline interpolator would be as good as the high-quality windowed sinc. Looking at the first harmonic at 2 kHz in figures 4.8(a) and 4.8(b), we can see peaks, but with a very low power, so that we cannot tell whether they are caused by aliasing or harmonic distortion. However we can conclude with these test results, that harmonic distortion is not a problem for the analysed resamplers; the bigger problem is definitely aliasing.



(a) Zero-Order-Hold

(b) First-Order-Hold

(c) Hermite Spline

(d) Windowed Sinc

Figure 4.9: Intermodulation Distortion test result.

4.3.6 Intermodulation Distortion

In figure 4.9 we would expect peaks created by intermodulation distortion at, for example, $14 \pm 0.06\,\text{kHz}$. However, again we mainly can only find aliasing caused peaks and those mainly from the 7 kHz signal (0.8, 14.8, 8.6, 21.5, 16.4, 13.7, 19.9, 5.9, 12.1 and 1.9 kHz), especially easy to see in figure 4.9(c). Even the first mirror frequency of the 60 Hz signal – which is 7740 Hz – is only weakly developed if at all, which is due to the fact that the low-order filters have roots near the multiples of the sampling frequency as we could see already in figure 4.3. This results in a good damping of the nearby frequencies, such as the aliasing frequencies of 60 Hz. The filter designed for high-quality

sample rate conversion shines forth as always. Again we conclude that intermodulation distortion doesn't seem to be a problem for the analysed converters.

4.4 Conclusion

The tests showed that there is only one real problem with the dynamic sample rate conversion algorithms discussed: Aliasing distortion. The other two tested types of non-linear distortion, namely harmonic and intermodulation distortion are not present. The three lower-order/quality filters are very fast to calculate and suit perfectly for real time applications like games. But for the high-quality target, Smith's algorithm with the possibility to actually influence the quality with the filter design is perfect. Although it hasn't been tested explicitly, it is also possible to design a low-quality filter which is realtime suitable at a higher quality than spline interpolation. Moreover it's possible to apply other continuous filters with a finite support (or ones which can be windowed) to a discrete signal with this algorithm. For example it's also possible to implement a causal sample rate converter using the Hilbert transform to transform the linear-phase sinc to a minimum-phase filter as described in [3].

The proof of the pudding is the eating.

English proverb

5 Practice

5.1 Setup

As the high-quality filter has been designed and compared against other sample rate conversion methods, we will now do a final test of the Doppler functionality using the final implementation in the open source software Blender.

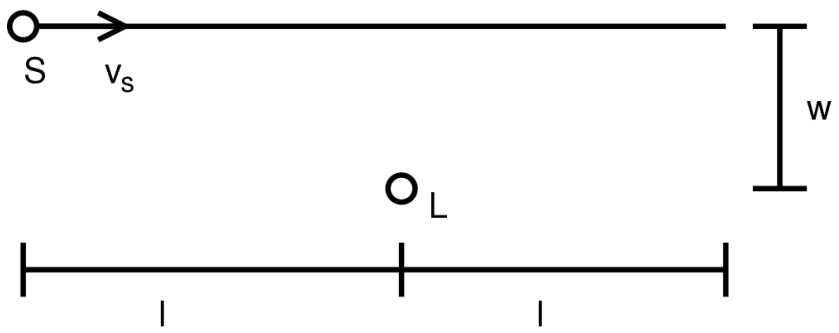


Figure 5.1: Doppler experiment setup.

For the test scenario we go back to our ambulance example and imagine an ambulance car with a siren S producing a pure sine tone at $f_0 = 1\,\mathrm{kHz}$, passing a listener L as shown in figure 5.1. The ambulance car moves at the speed $v_s = 25\,\frac{\mathrm{m}}{\mathrm{s}}$ over two times the distance $l = 50\,\mathrm{m}$ passing the listener at a distance of $w = 3\,\mathrm{m}$, resulting in a duration $T = 4\,\mathrm{s}$ of the experiment.

5.2 Calculation

We use the equations from chapter 1 to calculate the resulting perceived frequency:

$$\vec{SL} = \begin{pmatrix} l - v_s \cdot t \\ -w \end{pmatrix} \tag{5.1}$$

$$\vec{v_l} = 0 \tag{5.2}$$

$$\vec{v_s} = \begin{pmatrix} v_s \\ 0 \end{pmatrix} \tag{5.3}$$

$$v_{ls} = 0 \tag{5.4}$$

$$v_{ss} = v_s \cdot \frac{l - v_s \cdot t}{\sqrt{(v_s \cdot t - l)^2 + w^2}} \tag{5.5}$$

$$\Delta f = \frac{c_{sound}}{c_{sound} - v_s \cdot \frac{l - v_s \cdot t}{\sqrt{(v_s \cdot t - l)^2 + w^2}}} \tag{5.6}$$

$$f(t) = f_0 \cdot \frac{c_{sound}}{c_{sound} - v_s \cdot \frac{l - v_s \cdot t}{\sqrt{(v_s \cdot t - l)^2 + w^2}}} \tag{5.7}$$

5.3 Experiment

The setup has been created in Blender using the same sound file used for the harmonic distortion test, but without the noise. A spectogram of the resulting output with equation 5.7 evaluated and plotted in white above is shown in figure 5.2, showing that the Doppler calculation works as expected.

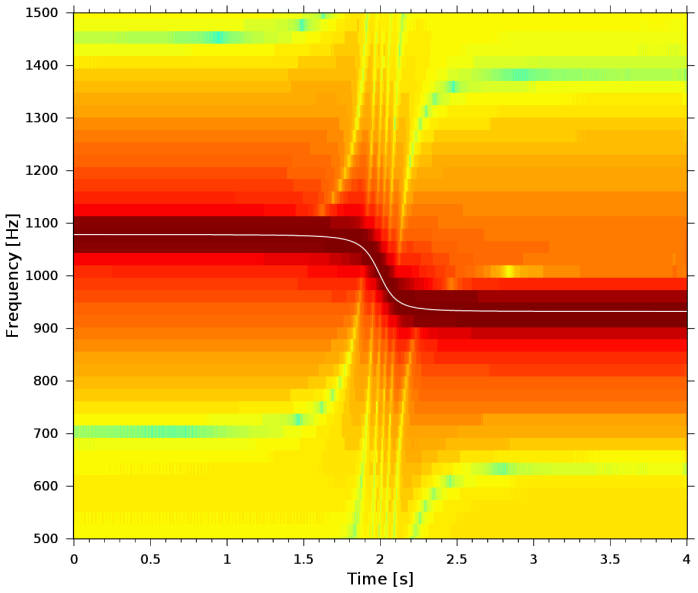


Figure 5.2: Doppler experiment result.

5.4 Conclusion

The Doppler effect has been shown to work quite well with the described methods and is a real enhancement for the 3D audio capabilities of future Blender created movies.

URLs The final results of the project can be found as

- web page on the SPSC Laboratory website (`http://www.spsc.tugraz.at/courses/bachelor-projects/sample-rate-conversion-audio-doppler-effect`), as
- C++ Source code of the implementation of Smith's algorithm (`https://svn.blender.org/svnroot/bf-blender/trunk/blender/intern/audaspace/intern/AUD_JOSResampleReader.cpp`) and as
- Video, showing examples of the Doppler effect in blender, including the above experiment (`http://www.youtube.com/watch?v=yOqpfJTXYYM`).

Bibliography

[1] Guoan Bi and Sanjit K. Mitra. Sampling rate conversion in the frequency domain. *IEEE Signal Processing Magazine*, 28(3):140–144, May 2011.

[2] Creative Labs, Inc. *OpenAL 1.1 Specification and Reference*, 2005.

[3] Niranjan Damera-Venkata, Brian L. Evans, and Shawn R. McCaslin. Design of optimal minimum-phase digital fir filters using discrete hilbert transforms. *IEEE Transactions on Signal Processing*, 48(5):1491–1495, May 2000.

[4] International Organization for Standardization. *ISO C Standard 1999*, 1999. ISO/IEC 9899:1999 draft.

[5] A.V. Oppenheim, R.W. Schafer, and J.R. Buck. *Discrete-Time Signal Processing*. Prentice-Hall, 2nd edition, 1998.

[6] A. Papoulis. *The Fourier integral and its applications*. McGraw-Hill electronic sciences series. McGraw-Hill, 1962.

[7] Paolo Prandoni and Martin Vetterli. From lagrange to shannon ... and back: Another look at sampling. *IEEE Signal Processing Magazine*, 26(5):138–144, September 2009.

[8] Julius O. Smith. Digital audio resampling home page. `http://ccrma.stanford.edu/~jos/resample/`, January 2002.

[9] Julius O. Smith. *Spectral Audio Signal Processing, March 2010 Draft*. March 2010. online book.

List of Figures