Technische Universität Berlin

Masterarbeit

Implementation of an additive sound synthesis for an electronic music instrument

Verfasser:
Benjamin Wiemann
[redacted]
[redacted]

Erstgutachter:
Prof. Dr. Stefan Weinzierl

Zweitgutachter
Henrik von Coler

für die Prüfung zum Master of Science

im Studiengang

Audiokommunikation und -technologie

Institut für Sprache und Kommunikation
Fachgebiet Audiokommunikation

Eingereicht am 30. Mai 2017

Eidesstattliche Erklärung

Titel der schriftlichen Arbeit: Implementation of an additive sound synthesis for an electronic music instrument
Verfasser: Benjamin Wiemann
Erstgutachter: Prof. Dr. Stefan Weinzierl
Zweitgutachter: Henrik von Coler

Datum, Unterschrift:

Technische Universität Berlin

Abstract

Fakultät I - Geistes- und Bildungswissenschaften
Institut für Sprache und Kommunikation
Fachgebiet Audiokommunikation

Master of Science

Implementation of an additive sound synthesis for an electronic music instrument

by Benjamin Wiemann

An additive synthesis system based on real instrument recordings is developed. A set of 83 semitones is recorded in four different dynamic levels, for which an analysis algorithm is used to extract amplitude and frequency data in the form of matrices. This data is given to a synthesis application, which can be controlled by the input parameters pitch and velocity. The system uses this information to pick the corresponding data matrices and interpolates between them to enable amplitude and frequency modulation in real time. The analysis data is also transformed into time-independent cumulative distribution functions. This data is used to generate amplitude and frequency from random numbers.

Technische Universität Berlin

Zusammenfassung

Fakultät I - Geistes- und Bildungswissenschaften
Institut für Sprache und Kommunikation
Fachgebiet Audiokommunikation

Master of Science

Implementation of an additive sound synthesis for an electronic music instrument

von Benjamin Wiemann

Es wurde ein additives Synthesesystem entwickelt, welches auf den Aufnahmen eines realen Musikinstruments basiert. Ein Satz von 83 Halbtönen wurde in vier verschiedenen Dynamikstufen aufgenommen. Auf diesen wurde ein Analyse-Algorithmus angewendet, um die Amplituden und Frequenzen zu extrahieren und in der Form von Matrizen zu speichern. Diese Daten werden von einer Syntheseanwendung benutzt, welche mittels der Parameter Anschlagsstärke und Tonhöhe gesteuert werden kann. Die Anwendung nutzt diese Informationen, um die passenden Datenmatrizen herauszusuchen und zwischen ihnen zu interpolieren. Dies ermöglicht die Modulation von Amplitude und Frequenz in Echtzeit. Die Analysedaten werden außerdem in zeitunabhängige kumulative Verteilungsfunktionen umgewandelt. Diese ermöglichen es, Amplitude und Frequenz aus Zufallszahlen zu generieren.

Contents

List of Figures

1 Introduction

In the last century, electronic signal processing enabled engineers to synthesize sounds which could not be found in the natural environment. The first synthesizers based on electronic circuits, and their sound was successfully used for the creation of new electronic music genres. However, attempts to simulate an acoustic instrument with a synthesizer produced interesting results, but their timbre quality could not keep up with their natural ideals. An infinite amount of new possibilities was introduced by computers and the science of digital signal processing. They allowed the music production and performance with applications, which were much closer to natural sounds. Digital controllers could be developed independently from the actual sound synthesis software.
Digital synthesis techniques can be organized into four categories: processed recording, spectral model, physical model and abstract algorithms (Julius O. Smith 1999). Processed recording techniques are based on previously recorded audio material. Physical modeling creates the sound by simulating the physical features of a real instrument. Spectral model techniques do not need knowledge about these features, since the synthesis is directly derived from spectral features of the sound which arrives at the human ear. Using abstract algorithms to simulate a real existing instrument or sound is rather difficult, but they offer many possibilities to create sounds with no connection to real world ideals. In this thesis, a spectral model approach is selected to create an application based on prerecorded and processed audio samples. We used additive synthesis, which first has been extensively described in Moorer 1977. The developed synthesis application is mainly refers to the model proposed by Serra 1997. Additive synthesis is known as one of the most flexible and powerful synthesis methods, which can be used for a proper reconstruction of the timbre of real instruments. It uses the concept of Fourier analysis, which is based on the fact that every signal can be modeled as sum of elemental sinusoid signals. In additive synthesis, these are generated individually. Since it gives control of the time development of every single partial and optionally noise components of instrumental sounds, this type of sound synthesis is more suitable for imitating real acoustic instruments than many other approaches are. However, this is accompanied by a big need for computational resources. For this reason, a lot of effort has been done to develop strategies to save computational performance, also to enable the application of this method for real-time use and live performances. This included e.g. effective oscillator design or frequency-domain synthesis. In this thesis we conceptualize and implement an additive synthesis system, which uses the already well documented approach. The synthesis is based on prerecorded instrumental sound recordings. Our approach consists of two stages: the analysis stage and the synthesis stage. The analysis stage involves all steps which were taken to receive the actual synthesis parameters of the recorded data. The synthesis stage describes the functionality of the system, where we discuss different possible designs. But since the implementation of the analysis stage is not part of this thesis, we go more into detail when we deal with the synthesis part. The classical additive synthesis uses the partial parameters received in the analysis stage to re-create the signal. Our application also supports this classical version. In addition to this, another mode is introduced and implemented, were the

synthesis parameters are created by the use of probability distributions. Furthermore, we discuss possibilities to control the parameters of the system.
In chapter 2, we describe the two stages and the mathematical backgrounds of our system. In chapter 3, we write about details of the implementation of the system. Finally, in chapter 4 we analyze the resulting sound and the performance of the system.

2 Fundamentals

2.1 Basics of Additive Synthesis

A single sound can be modeled as a deterministic part and a stochastic residual. The deterministic part is the tonal part of the sound which can be described as a sum of multiple sinusoids, also called partials. The stochastic residual is simply another word for noise, to which no specific frequency can be assigned, but which can be described by filter parameters. Equations 2.1, 2.2 and 2.3 follow the definitions of Serra 1997.
Every signal kann be modeled as a sum of sinusoids plus a residual noise component:

$$s(t) = \sum_{i=1}^{N} a_i(t) \sin(\phi_i(t)) + r(t). \tag{2.1}$$

The function $a(t)$ represents the instantaneous amplitude, $\phi_i(t)$ the instantaneous phase and $r(t)$ the residual noise component. The instantaneous phase is determined by the integral of the instantaneous frequency $f_i(t)$:

$$\phi_i(t) = \int_{\tau=0}^{t} f_i(\tau)\mathrm{d}\tau. \tag{2.2}$$

The noise signal is defined as follows:

$$r(t) = \int_{\tau=0}^{t} h(t,\tau)u(\tau)\mathrm{d}\tau, \tag{2.3}$$

where $u(\tau)$ is white noise and $h(t,\tau)$ is a time-varying filter impulse response at time t. That is, the noise component is modeled by subtractive synthesis. According to the Fourier theorem, every signal, which also includes noise, can be modeled by a sum of sinusoids. But, as Serra 1997 points out, that approach would need a much bigger number of computationally intensive sinusoids.
However, for the timbre of a violin, as well as for the timbre of many other instruments, noise is not the dominating component in comparison to the deterministic part. The stochastic component is especially audible, when the violin is played at a low dynamic level. Of course, there are also instruments like cymbals, snares and some types of wind instruments (e.g. the panpipe), whose synthesis highly depends on the correct modeling of the noise component. In the context of this thesis we focus our work on the deterministic part. The analysis and synthesis of the noise component will not be implemented, but it might still be a part of the future development of the application.

2.1.1 Comparison to Sampling Technique

Additive synthesis has the disadvantage, that for each partial a digital sinusoidal oscillator is needed, which is very expensive compared to the classical sampling technique. Sampling offers a high-quality reproduction of the original recorded sound with a very low need of computational resources. However, sampling has some limitations, especially relating to the needs of live performances. Sampling does not offer the same possibilities in changing the timbre of the sound. Filters have to be used to adjust spectral phase and amplitude, which are less precise than the individual control of the amplitude and frequency of every partial. Also, pitch shifting and time stretching are bound together: playing a sound slower results in a (probably unintended) lower pitch and a different timbre. This is especially audible in the characteristics of human speech. Conversely, an intended pitch shift of the fundamental frequency causes the timbre and durance of the sound to change. As a solution to this, time stretching and pitch shifting algorithms can be applied. Nonetheless, these suffer from artifacts, when bigger time stretches and pitch shifts are applied (**zolzer˙timesegment˙2011**).

2.2 Application and Control Concept

The application is able to be played in real time, thus forming the software part of an electronic instrument. The number of controllable parameters is determined by the number of sinusoids N, which may vary in amplitude, frequency and starting phase each. The starting phase is not measured in the analysis stage, since it is not essential for the sound of a violin. Nevertheless there still remain $2 \cdot N$ controllable parameters, which need to be controlled by a digital music controller. Most commercially available controllers have in common, that the number of input parameters is below the number of control parameters needed for additive synthesis. To solve this mapping problem, a low-to-many mapping strategy (Hunt, Wanderley, and Kirk 2000) needs to be created. It is not part of this thesis to develop a final mapping strategy for a controller. Instead a more general approach is discussed how to deal with a class of controllers, and which parameters we actually want to be controlled by the user.
A later goal is to run the application together with the digital controller introduced by Treindl 2016 and von Coler, Treindl, et al. 2017, which is under further development. This controller enables the musician to address all semitones of a full octave by pushing combinations of four pressure-sensitive buttons. Additional buttons exist for the octave selection, and a so-called *excitation pad* enables the control of three degrees of freedom. This controller is representative for a class of controllers, which are primarily designed to modulate pitch as discrete values in the form of semitones. This includes ordinary piano-style keyboard controllers and pad controllers, which were used to test the synthesis system. However, these controllers differ in their ability of continuous pitch modulation (e.g. a pitch modulation wheel, which are part of most of the keyboard controllers), and continuous velocity modulation (e.g. so-called aftertouch of pad controllers and some keyboards). Since the controller described by von Coler, Treindl, et al. 2017 has as least four continuous degrees of freedom (the variable pressure on the buttons plus the ones of the excitation pad), these controller features are considered to be available. Another point of view is to ask which parameters are needed to control the application in a way to achieve results which are similar to the simulated instrument. The recordings have been made with a violin, which is mainly played by varying pitch in a discrete (*which* string to touch/play) and continuous way (*where* to press the string). Velocity can be varied on a continuous scale by controlling speed and pressure of the bow. Since in terms of electronic music instruments velocity as a control parameter is often distinguished from aftertouch, we want to clarify that our definition of velocity includes both parameters in one term. Velocity should be understood as a parameter which primarily influences the amplitude of the resulting sound signal. It should be noted that the synthesis system can also imitate another acoustic instrument

which is primarily played by the control of pitch and velocity. This would simply be done by using a recorded set of samples of this instrument. As a conclusion to the pictured findings, the application supports three main control parameters:

1. *note selection* per discrete indices, where a limited number of notes is supported. These indices correspond to the semitones on a equally-tempered scale. The input of a note index results in the creation of a voice. The number of playable notes is determined by the number of recorded notes in the data set (see section **??**. The application is designed to be polyphonic, thus supports the existence of multiple voices at the same time.

2. *velocity* as a continuous input parameter, which can be varied over the whole time in which a note is played. If the controller supports it, every note can be varied independently from the others.

3. *pitch* as a continuous input parameter, which is also variable over the playing time. This variation can also be done per note, although most of the polyphonic controllers support only a global pitch variation, which influences all played voices simultaneously.

The remaining part of musical control is the control of timbre. In Houtsma 1997, the timbre of a sound is described as a multidimensional attribute of a sound, which not only includes the spectral profile, but also the temporal envelope of the sound. Relating to our additive synthesis model, the timbre of the synthesized sound is mainly described by the partial tone trajectories of amplitude and frequency of the imitated instrument. It is possible to play the synthesizer without further modifications of these partial tones, which is also not done in this first version of the synthesis system. Changing the pitch control parameter will result in a consistent exponential change of every partial tone frequency, and changing the amplitude control parameter will result in a consistent exponential change of every partial tone amplitude. The timbre is then selected by interpolating between the timbres of the recordings which are the closest in pitch and amplitude (see section 2.6). Nevertheless, the system uses internal control messages (see section 3.1.2), where additional control parameters can be added to enable timbre modifications in future versions. In comparison to pitch and velocity, finding a convincing mapping concept for timbre control will be more challenging.

2.3 Analysis Data

2.3.1 Recording

Since the timbre of instrument, in this case a violin, changes with the playing velocity and pitch, we tried to gain data of the whole velocity and pitch range of the violin. A set of recordings was played by a professional violinist in a non-reverberant room. A number of 83 semitones or 7 octaves were recorded in the four dynamic levels piano, pianissimo, forte and fortissimo. The chosen tuning frequency was 443 Hz. The violinist was instructed to play each tone on a constant amplitude, a constant pitch and a fixed length. Multiple takes were done to enable a later selection of the best take. The recording was done with two microphones simultaneously. The first one was a DPA 4099 cardiod clip microphone and the second one was a Brüel & Kjär 4006 omnidirectional small diaphragm microphone with free-field equalization. All material was captured in a resolution of 96 kHz and 24 Bit word length.

2.3.2 Segmentation and Labeling

The synthesis application supports two different synthesis modes, the deterministic mode and the stochastic mode. Note that deterministic and stochastic in this context does not mean the separation of sinusoidal and noise components as used in section 2.1, but instead indicates how the synthesis parameters amplitude and frequency are generated. A more detailed description will be given in 2.6. This differentiation is important at this point, because the preparation of the audio data for the stochastic mode requires additional steps.
From every recording, a preferred take was selected by listening to its sound and examining the waveform. In the deterministic mode, the application directly takes the frequency and amplitude data from a text file and plays it in the exact order given in this file. A simple comma separated text file format is used, where every row consists of the amplitude respectively frequency values of all partials of one analysis window. For this purpose, it is sufficient to mark the beginning and the end of the preferred take. The marked sound material will serve as the input for the parameter analysis described in section 2.4. For the data generation in stochastic mode, a further segmentation of the marked recording is necessary. Hence, every chosen recording was segmented into three parts: the attack, the sustain and the release part.
For the procedure of marking the beginning and the end of a take, as well as for the marking of the segments, the software Sonic Visualiser (Cannam, Landone, and Sandler 2010) was used, which enables marking time instants in an audio file. We followed the criteria for music segmentation described by von Coler and Lerch 2014.

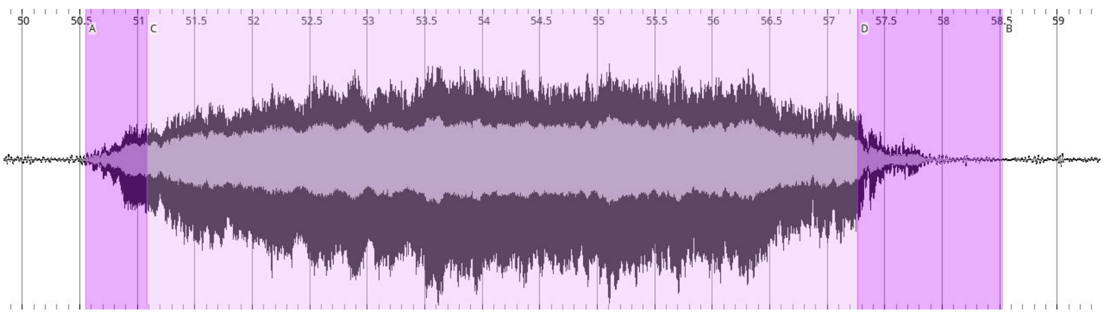

FIGURE 2.1: Segmentation of a recording in attack, sustain and release part. The note was played in pianissimo. The durance of the attack segment is 600 ms.

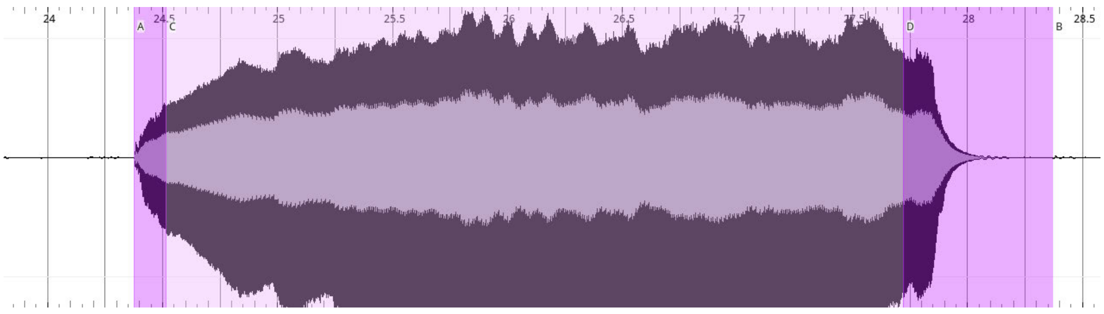

FIGURE 2.2: Another segmentation of a note, this time played in fortissimo. The attack segment is only about 150 ms long.

For each file, the beginning was marked at the last instant where the signal was still inaudible. The end was marked at the first instant, where the signal became inaudible again. The cut between the attack and the sustain part was marked at the instant where the onset transient is finished and the partials reached their target frequencies. These instants were located by using the peak frequency spectrogramm of Sonic Visualiser. The time span of the resulting attack parts reaches from approximately 50 ms till 700 ms, mostly depending on the dynamic level of the examined audio file (see figures 2.1 and 2.2). The beginning of the release part has been marked at that instant where it was assumed that the musician lifts the bow and the contact of the bow and the string is interrupted. This was done by careful listening and also by examining the peak frequency spectrogram. Typically, the release of the bow is indicated by decreasing of the amplitude of the higher partials. Generally, the segmentation markers were more easy to set for the recordings of higher dynamic level. The recordings played at lower levels, which are piano and particularly pianissimo, often contain sections where the bow looses contact to the string and then touches it again, making it more difficult to find the exact instant.

2.4 Analysis Stage

In preliminary work to this thesis, the amplitudes and frequencies, which are needed for the re-synthesis of the sound, were extracted and written into text files. To find the frequencies and amplitudes of every partial trajectory, the analysis procedure of Serra 1997 was used. Given a recording, which was trimmed at its start and ending points, a spectrogram is computed from the whole audio file. Every spectrum of the spectrogram is computed from a windowed part of the signal using the Fast Fourier Transformation (FFT). The analysis window is of length M. Then, a peak detection algorithm is applied to each spectrum of the spectrogram. Subsequently, the algorithm writes rows of N floating point values into the according text file, where N is the number of analysed sinusoids. Opposing to Serra 1997, no peak continuation detection was included into the algorithm. In every spectrum, the algorithm expects a fixed number of N partials. Hence, the first N peaks of a spectrum are analyzed, beginning with the fundamental tone. The analysis was applied on every recording in the data set. In the following, this will be called the deterministic analysis data. In the deterministic mode of the synthesis application, this data is used directly as input for the sinusoid generators. The input parameters note, pitch and velocity are used to determine the most suitable files, from which the rows of the current window index are read and interpolated (see section 2.6). These parameters are used to synthesize a number of M samples. The resulting signal is of the same length as the original recording, and sounds very close to the original.
This approach is the easiest to synthesize a sound signal with the given analysis data. But it has the constraint, that a musician or producer will only be able to play tones of a maximum duration equal to the duration of the currently selected source data. A longer playing is not possible, and if the musician stops the tone earlier, it results in an abrupt, unnatural sounding cut of the signal. Thus, it is advisable to parametrize the available data into another format to make it time-independent.

2.4.1 Generating Statistical Data

To understand how the parametrization of frequency and amplitude could be done, we refer to a model applied by von Coler and Röbel 2011. In connection to a vibrato detection algorithm, the authors suggest that the variation of the fundamental frequency

f_0 of a tonal sound can be modeled as

$$f_0 = f_{\text{step}} + f_{\text{cor}} + f_{\text{mod}}. \tag{2.4}$$

The first summand f_{step} is the fundamental tone of the played note, f_{cor} stands for intended frequency correction moves and glissandi, and f_{mod} models a periodic frequency modulation. We use this model as an initial point to add some features. In the author's opinion, this model can be applied to every partial of the tone, when f_{step} is adjusted to the according partial frequency. In our recordings, the musician was instructed to play a single tone of a given note without any modulation. Hence, f_{cor} and f_{mod} should be zero, and all three summands should remain constant over time. Still, in the sustain part of the signal a very small frequency variation is present. These variations are a result of unintended movements of the musician, but also of the physical properties of the instrument. They cannot be avoided and thus represent a characteristic part of the timbre. For the context of additive synthesis, we propose a new model of the frequency variation of a partial i :

$$f_i = f_{\text{step},i} + f_{\text{cor}} + f_{\text{mod}} + f_{\text{stoch},i}, \tag{2.5}$$

where $f_{\text{stoch},i}$refers to the stochastic timbre element of the frequency variation. Note that this timbre variation is specific for a partial i . Analogously to this, we propose to model the amplitude as

$$a_i = a_{\text{step},i} + a_{\text{cor}} + a_{\text{mod}} + a_{\text{stoch},i}, \tag{2.6}$$

where $a_{\text{step},i}$ declares the amplitude of a dynamic level, a_{cor} models the amplitude correction moves, a_{mod} is the intended amplitude modulation and $a_{\text{stoch},i}$is the stochastic part of the amplitude, which we use to model timbre-relevant random amplitude variations.

To parametrize $f_{\text{stoch},i}$and $a_{\text{stoch},i}$, patterns of the movements of both variables must be explored. In our approach, we simply chose to transform the partial tone trajectories of the sustain part into time-independent statistical data. We only choose the sustain part, since the behavior of amplitude and frequency of the attack and the release part highly depends on time and is thus unsuitable to be modeled by a CDF, which stays constant over time.

For each file containing data of partial trajectories (which is associated to a specific note and dynamic level) and for each partial tone, a cumulative distribution function (CDF) was generated. The algorithm works as follows: First, the boundaries, in which the amplitude of a partial moves in between during the sustain part, are detected. This interval gets subdivided into an equally spaced partition, which contains a fixed number of subintervals. A statistical frequency analysis is done by counting the number of amplitude values in each subinterval. The resulting probability density function (PDF) then gets transformed into a CDF, which in opposite to a PDF can be inverted. The CDF is then written into a text file. The inverted CDFs (ICDF) are used to synthesize the values of $f_{\text{stoch},i}$and $a_{\text{stoch},i}$in the stochastic mode of the application (see section 2.6.2).

2.4.2 Data Denotation

In the following sections, we will refer to the analysis data in many different contexts. Hence, we define some symbols to describe certain elements of the data. Depending on if we are dealing with stochastic or static analysis data, addressing the needed variables follows different principles. Both have in common, that single data structures for a given note index n and a given velocity index v have been generated. These data structures are of different shape.

Denotation of Static Data

In case of static data, a frequency value a or an amplitude value f are located in precalculated matrices. Every recording can be identified by its note index and its velocity index, and so can its data matrices $F_{n,v}$ and $A_{n,v}$. Every value in theses matrices is addressed dependent of two different parameters. These are time, which is subscripted by the analysis window index k , and the partial tone, which is subscripted by the partial index i . Within this matrix, a single value will be written as $a_{i,k}$ or $f_{i,k}$. A whole vector of partial frequencies or amplitudes of a given window index k is denoted as $\vec{f}_k$ respectively $\vec{a}_k$. A whole amplitude or frequency trajectory for a given partial tone i is denoted as a vector $\vec{a}_i$ or $\vec{f}_i$. Generally, to denote a single frequency or amplitude value in the whole set of matrices, the expressions $a_{n,v,i,k}$ and $f_{n,v,i,k}$ can be used. However, this amount of subscripts makes a term unnecessarily complicated. So, depending on the context, the use of certain subscripts will be avoided, if they do not support the understanding of a term.

Denotation of Stochastic Data

The inverse cumulative density functions, which are a result of further processing of the data matrices, will be denoted as $\mathring{f}_{n,v,i}[x]$ and $\mathring{a}_{n,v,i}[x]$, where $x\epsilon[0,1]$ is a uniformly distributed random variable. A discrete denotation is used, since the function is internally saved as discrete pairs of key and values. Again, depending on the context, the use of subscripts may be avoided.

2.5 Synthesis Stage

2.5.1 Frequency Domain Synthesis

Because of the high computational costs of time domain additive synthesis, a frequency domain synthesis approach has been proposed by Rodet and Depalle 1992. For each time window, a vector of amplitudes $\vec{a}$ and frequencies $\vec{f}$ is used to build a frequency spectrum, which is then transformed into the time domain using an IFFT. The resulting time frames are then stringed together via overlap-add. This is done to interpolate amplitude and frequencies. If both frames contain sinusoids of different frequencies, an unintended amplitude distortion occurs. A solution to this has been introduced by Goodwin and Rodet 1994 and Goodwin and Kogon 1995. The IFFT method achieves a better performance than time domain synthesis, which can be divided by a factor of up to 20 (Rodet and Depalle 1992). However, since the spectrum resolution needs to be high enough to represent low frequencies accurately, the IFFT size must not fall below a certain threshold. A higher IFFT size results in a longer time window. This increases the latency, which is actually required to stay low for live performances in connection with a digital controller. An alteration of either pitch or velocity would not be noticeable until the synthesis of the current window is finished. Since the computational capability of personal computers also increased exponentially since the algor ithm was developed, we stayed with the classical time domain approach.

2.5.2 Time Domain Synthesis

Also in the time domain new amplitude and frequency values will only be picked every N samples, where N is the size of the analysis window. In between, the values need to be interpolated. This can be done by using the overlap-add technique. Unfortunately, this introduces the distortion described in 2.5.1. We chose to interpolate amplitude and phase using interpolation polynoms. An approach to do this has been presented by McAulay and Quatieri 1986. In this work the authors suggest to derive a cubic polynom from the given frequency and phase on the left and right boundary of a frame. This requires to save phase information at analysis stage for every analysis window. However, this isn't contained in the given analysis information and is also not intended for this project. Hence, from the four described parameters, in our synthesizer there are only three given for the synthesis of a frame: both of the frequency parameters, but only the starting phase information, which is a result of the previously synthesized frame. Given these constraints, it would still be possible to generate a quadratic polynom for phase interpolation, like it has been done in an interpolation method described in Qian and Ding 1997. Since the frequency function is the derivative of the phase function, this would result in a linear frequency interpolation. The problem of the quadratic frequency interpolation is, that as long as a frequency of a partial stays constant from the left frame boundary to the right one, the resulting interpolation curve is still shaped like a parabola. This avoids the frequency to stay constant, which isn't the case for the linear frequency interpolation. However, both approaches in McAulay and Quatieri 1986 and Qian and Ding 1997 produce non-differentiable points in the frequency interpolation curve at the frame boundaries. So as an alternative, it's possible to use the derivative of the frequency function, the *rate of frequency change*, at the right boundary of the previous frame. But quadratic phase interpolation stays a legitimate option, also since it is the less computationally intensive solution compared to the cubic interpolation. We hence chose to apply the quadratic interpolation for the frequency, and linear interpolation for the amplitude.
Given the three constants phase ϕ_k, frequency $\omega_k = f_k \cdot 2\pi$ of the left boundary and frequency $\hat{\omega}_{k+1} = f_{k+1} \cdot 2\pi$ of the right boundary, the instantaneous phase is defined as

$$\hat{\phi}_k[t] = \hat{\phi}_k + \hat{\omega}_k t + \frac{\hat{\omega}_{k+1} - \hat{\omega}_k}{2T} t^2, \tag{2.7}$$

where T is the frame size in samples. The instantaneous frequency is then

$$\hat{f}_k[t] = f_k + \frac{f_{k+1} - f_k}{T} t. \tag{2.8}$$

Analogously we define the instantaneous amplitude as

$$\hat{a}_k[t] = a_k + \frac{a_{k+1} - a_k}{T} t. \tag{2.9}$$

Figure **??** shows an example the modulation of amplitude and phase on the time scale.

2.6 Amplitude and Frequency Generation

2.6.1 Interpolation and Extrapolation

The synthesized audio signal will vary depending on control velocity and control pitch. The problem here is, that there only exists one recording per semitone and dynamic level. The synthesizer shall be able to modulate its sound by a continuous pitch and

velocity variation. Hence, an interpolation system is needed, for which the application of different types of interpolation, for instance, linear, polynomial or logarithmic interpolation, have to be considered. That is, including the time scale interpolation described in section 2.5.2, the system needs to interpolate on three different scales.
The pitch-velocity interpolation problem can be viewed as a two-dimensional interpolation on a grid, where pitch is represented by one axis and velocity is represented by the other. The pitch axis position is the sum of the note index n and the pitch Δp. The velocity axis position is determined by the input velocity. Internally this parameter is also split into the sum of the velocity index v and the velocity offset Δv. That is, values from four different data sets are taken to calculate a single interpolated value. For example, a frequency value will be calculated by values from the matrices $F_{n,v}$, $F_{n+1,v}$, $F_{n,v+1}$ and $F_{n+1,v+1}$. In case of deterministic synthesis, the needed vectors are directly taken from these matrices. If the stochastic synthesis mode is used, the matrices contain vectors of CDFs, which are used to generate the partial frequency vector. The interpolation procedure stays the same for both modes.
Since we also need to think about an performant implementation of the synthesis system, we are also going to present more efficient, but less accurate alternatives for some cases of interpolation. Note that for this type of interpolation the partial vectors are taken from the partial matrices of different recordings. The recordings are of different duration, thus the partial tone matrices are of different size on the time axis. A strategy is needed to deal with the situation where only one of the two needed partial vectors for a given k exists. This will be the case for the beginning of the attack state and the end of the release state, as well as for the end of the deterministic mode. As a solution we choose to use extrapolation algorithms, which only use frequency and amplitude data of a single recording. Nevertheless, interpolation is generally favored over extrapolation, since the former achieves results closer to the original timbre.

Frequency Interpolation

We define a formula for frequency interpolation. Let $\Delta p \in [0,1)$ be the pitch offset in semitones and $\tilde{f}(v,n)$ be the sought pitch-dependent frequency interpolation function. Since frequency behaves exponentially to pitch, and the notes have been recorded using the equally tempered scale, the following equation applies:

$$\frac{\log_2(\tilde{f}(v,n+\Delta p)) - \log_2(f_{n,v})}{\log_2(f_{n+1,v}) - \log_2(f_{n,v})} = \frac{\frac{1}{12}}{\frac{1}{12}} \frac{(n+\Delta p) - n}{(n+1) - n} = \Delta p. \tag{2.10}$$

Subsequently, we transform the equation into

$$\tilde{f}(v,n+\Delta p) = f_{n,v} \cdot 2^{\Delta p(\log_2(f_{n+1,v}) - \log_2(f_{n,v}))}. \tag{2.11}$$

For the sake of completeness, we define a similar function $\hat{f}(v,n)$ to interpolate on velocity axis:

$$\hat{f}(v+\Delta v,n) = f_{n,v} \cdot 2^{\Delta v(\log_2(f_{n,v+1}) - \log_2(f_{n,v}))}. \tag{2.12}$$

Since frequency is not expected to differ a lot over different velocity levels, a linear interpolation $\hat{f}_{\text{lin}}(v,n)$ might be a computational less expensive alternative:

$$\hat{f}_{\text{lin}}(v+\Delta v,n) = (1-\Delta v) f_{n,v} + \Delta v f_{n,v+1}. \tag{2.13}$$

Even cheaper is simply to use the closest value:

$$\hat{f}_{\mathrm{clo}}(v+\Delta v, n) = \begin{cases} f_{n,v}, & \text{if } \Delta v < 0.5, \\ f_{n,v+1}, & \text{if } \Delta v \geq 0.5 \end{cases} \tag{2.14}$$

It does not matter on which axis the interpolation takes place first. If it is started on the frequency axis, then two interpolations $\tilde{f}(v, n+\Delta p)$ and $\tilde{f}(v+1, n+\Delta p)$ have to be calculated, and afterwards the interpolation $\hat{f}(v+\Delta v, n+\Delta p)$.

Frequency Extrapolation

For the extrapolation for a tone, let $\Delta p \,\epsilon\, (-\infty, \infty)$ be the pitch offset and $\alpha \,\epsilon\, \mathbb{R}$ a scaling factor. The searched extrapolation function $\tilde{\tilde{f}}(v, n)$ on the pitch axis is then

$$\tilde{\tilde{f}}(v, n+\Delta p) = \alpha \cdot f_{n,v} \cdot 2^{\frac{\Delta p}{12}}, \tag{2.15}$$

where the scaling factor is by default $\alpha = 1$. But it may happen, that for a certain point in time the frequency computation function changes from interpolation to extrapolation. This is the case when a deterministic part of fixed length of a voice gets synthesized, and one of the two frequency matrices (of duration $k = K$) used for interpolation ends earlier than the the other one (which is of size $k > K$). At window K, both functions are of equal value:

$$f_{n,v,K} \cdot 2^{\Delta p(\log_2(f_{n+1,v,K}) - \log_2(f_{n,v,K}))} = \alpha \cdot f_{n,v,K} \cdot 2^{\frac{\Delta p}{12}}.$$

It follows that

$$\alpha = 2^{\Delta p(\log_2(f_{n+1,v}) - \log_2(f_{n,v}) - \frac{1}{12})} \tag{2.16}$$

This avoids the change from one frequency function to another to be discontinuous. The factor α can be kept, as long as the extrapolation function is in use. Note that the change can also happen the other way around, from extrapolation to interpolation. Yet, there is another, more simple approach to circumvent the problem of different time axis sizes. For the deterministic mode, this is to fill the missing values with zero vectors. This would as well lead to a fade out of the sound instead of a sudden stop. For the stochastic mode, attack and release part are that short, that we will stay with a constant use of extrapolation.
The described extrapolation function is still very reliable in terms of finding the correct fundamental frequency value to a pitch value, since we know that for the recordings an equally tempered scale has been used. Yet, we do not have a model to predict the timbre change, so a timbre error will be present. We assume that for a pitch offset $|\Delta p| \leq 0.5$ the timbre error will be inaudible. A larger pitch offset $|\Delta p| \leq 0.5$ is not needed in the normal case, since during a pitch modulation, at any position where $\Delta p = 0.5$, the interpolated frequency will change from $\tilde{f}(v, n+\Delta p)$ to $\tilde{f}(v, (n+1)-\Delta p)$. However, exactly this change represents a problem, since the sudden change from one timbre to another is expected to be audible. To avoid this, it is possible to keep n constant and modulate the pitch change only by Δp. If pitch modulations of more than half a semitone shall be possible, the magnitude of course needs to be greater than 0.5. At a certain threshold, the timbre error will definitely become audible. Thus, the use of extrapolation introduces a dilemma, which is the main reason why interpolation is favored over extrapolation.

There is no change of frequency expected on the velocity axis, so we will not define another extrapolation function for this case.

Amplitude Interpolation

Opposed to the frequency interpolation, the functions for the pitch and velocity depending amplitude between matrices of different velocity level are not as well defined. If the four measured sound pressure levels are approximately equally spaced in dB, then the amplitudes will approximatly grow exponential to the velocity index. To verify this assumption, we examined a few samples of matrices and compared the root mean square of the amplitude of the second partial. For a fixed note, the increase of amplitude was indeed approximately doubling on each velocity level. For that reason, the amplitude interpolation and extrapolation functions will be modeled as logarithmic interpolation functions. Furthermore, we also introduce a scaling factor β for both functions. According to this, let Δv be the velocity offset and the amplitude interpolation $\hat{a}(v,n)$ is defined as

$$\hat{a}(v+\Delta v, n) = a_{n,v} \cdot 2^{\Delta v(\log_2(a_{n,v+1})-\log_2(a_{n,v}))}. \tag{2.17}$$

Equivalent to the frequency interpolation functions, the amplitude is neither expected to change much over the pitch axis, nor we know in which functional behaviour it will change. Hence, we also define three different interpolation functions:

$$\tilde{a}(v, n+\Delta p) = a_{n,v} \cdot 2^{\Delta p(\log_2(a_{n+1,v})-\log_2(a_{n,v}))}, \tag{2.18}$$

$$\tilde{a}_{\text{lin}}(v, n+\Delta p) = (1-\Delta p)a_{n,v} + \Delta p a_{n+1,v}, \tag{2.19}$$

$$\tilde{a}_{\text{clo}}(v, n+\Delta p) = \begin{cases} a_{n,v}, & \text{if } \Delta p < 0.5, \\ a_{n,v+1}, & \text{if } \Delta p \geq 0.5 \end{cases} \tag{2.20}$$

Amplitude Extrapolation

According to the frequency extrapolation, we also define an extrapolation function $\hat{a}(v,n)$ for the amplitude on the velocity axis. We know that amplitude increases exponentially to velocity, and thus is approximately described by the function

$$\hat{a}(n, v+\Delta v) = a_{n,v} \cdot 2^{\Delta v \beta} \tag{2.21}$$

Opposing to frequency extrapolation on pitch axis, the exact exponent β of the exponential function is not generally known. We derive β individually using the exponent of the last interpolation function:

$$\beta = \log_2(a_{n,v+1}) - \log_2(a_{n,v}) \tag{2.22}$$

The window index K is again the instant, where the switchover from interpolation to extrapolation, or the other way around, happens. As a simpler, but less accurate alternative, β can also simply be set to 2.

2.6.2 Stochastic Modeling

In the sustain part, the signal remains stationary. That is, the transient effect is finished and the partials reached their desired frequencies. At that point, we can generate the amplitude and frequency values of every partial from a probability density function. The principles of this and similar procedures are described by Devroye 1986.

The generated CDFs are saved as discrete maps with a limited resolution. For every window k and for each partial i , a new random value $x\epsilon[0,1]$ is generated from a uniform distribution number generator and then mapped by the ICDF to a value. Therefore, the result is linearly interpolated from its two closest neighbor values in the ICDF. Another possibility is simply to set it the closest neighbor value.
The random value vector $\vec{x}$ is generated once for a whole amplitude vector and once for a whole frequency vector. This vector is reused for all four ICDF vectors on the pitch-velocity interpolation grid. Every partial has its own random values, from which some may generate negative deviations from the mean value, and some may generate positive ones. In this first version of a stochastic modeling, we assume that $f_{\text{stoch},i}$ and $a_{\text{stoch},i}$ of a partial i do not correlate with the frequencies or amplitudes of other partials. However, this assumption still needs to be proven.

2.7 Sinusoid Generation

Since in this thesis the synthesis is done in time domain, it is important to think about how to implement the oscillators, whose output signals are taken to create the sinusoids. Sine and cosine functions of general purpose math libraries often use approximations like the Taylor approximation. These are unsuitable for real-time applications which use permanent sine function generation, since they are computationally too intensive. Our goal is to find a sine generation method with an acceptable performance, suitable to run on a general-purpose cpu based on the x86 architecture. Acceptable in this context means that the resulting program should be able to run in real-time, with a latency as low as possible. It should be able to generate at least one voice without generating errors or gaps in the resulting signal.

Recursive Oscillators

A comparison of the different sine generation algorithms for additive synthesis was done by Phillips 1996. The author proposes to either choose recursive oscillators or table lookup approaches. The theory behind digital recursive oscillators has extensively been described by Turner 2003. They exist in many different versions, varying on their ability to produce complex output, their ability to keep their amplitude constant and on the easiness of frequency modulation. Furthermore, they differ in the number of arithmetical operations needed for one iteration. Depending on the use context, the best oscillator to use might be different. The oscillator system can be viewed as a feedback loop. Hence, the loop gain needs to be 1. Some recursive oscillators suffer from numerical errors, which are added on every recursion. This error can result in an unstable system. Depending on the number format (floating point or fixed point) and on the word length, this problem is more or less critical.
A popular version used for audio synthesis is the biquad oscillator. It onlys needs one multiplication and one substraction per iteration. It is defined as

$$x[t] = 2\cos(\frac{2\pi f}{f_s})x[t-1] - x[t-2] \tag{2.23}$$

Figure 2.3 shows the block diagram of the biquad oscillator. This type of oscillator was used by Hodes et al. 1999 for an implementation of additive synthesis on a T0 fixed point vector microprocessor. The T0 processor uses a vector arithmetic unit to perform parallel operations on all vector elements at the same time. It also owns a vector load/store unit. Since the frequency only changes when the computationally

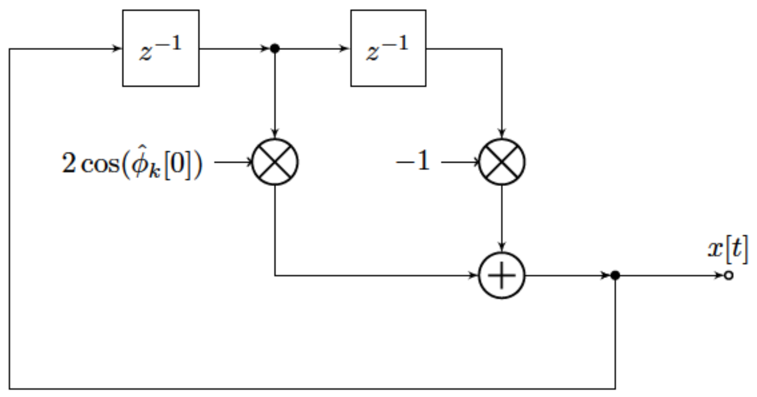

FIGURE 2.3: Block diagram of a recursive biquad oscillator. The first-order coefficient needs to be recomputed for every frame k, taking the instantaneous phase $\hat{\phi}_k[0]$ of the left frame boundary as argument. Therefore a standard implementation of the cosine function can be used.

expensive term $2\cos(\frac{2\pi f}{f_s})$ is recomputed, it is kept the same over the length of one frame. The overlap-add technique with a triangular window function is used to interpolate amplitude and frequency linearly. This works well to interpolate the amplitude between frames and avoids discontinuities. On the other hand different frequencies in both frames cause amplitude distortion. A solution to this has been found for the synthesis in frequency domain (Goodwin and Kogon 1995). However, this problem was not solved by Hodes et al. 1999, but it was simply tolerated. Also, it was not examined if the distortion is audible. Since the frequency might change very quickly due to glissando playings by the musician holding the controller, this is one disadvantage of the biquad oscillator approach. A second reason not to choose the biquad oscillators is the fact, that an overlap-add technique with 50% overlap doubles the number of computed sine values. Another recursive oscillator, the *digital waveguide oscillator*, was proposed by Julius O Smith and Cook 1992. It offers frequency modulation, but the authors also report timbre distortion.

Table Lookup

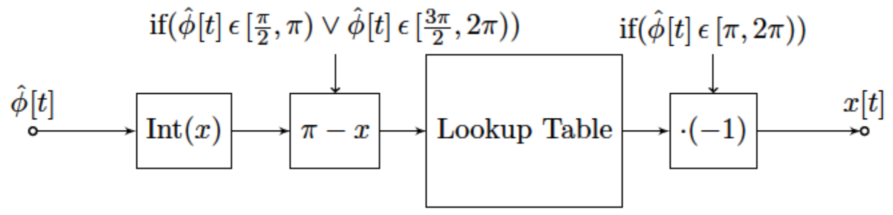
$\text{if}(\hat{\phi}[t] \in [\frac{\pi}{2}, \pi) \vee \hat{\phi}[t] \in [\frac{3\pi}{2}, 2\pi))$
$\text{if}(\hat{\phi}[t] \in [\pi, 2\pi))$
$\hat{\phi}[t]$
$\text{Int}(x)$
$\pi - x$
Lookup Table
$\cdot(-1)$
$x[t]$

FIGURE 2.4: Block diagram of the table-lookup technique. At first, the phase argument is transformed into an integer number (per truncation or rounding) between 0 and the table size 2^n. Depending on the phase value, the table input and/or output gets inverted to find the according sine function value.

The table lookup method uses a table of a precomputed period of a sine wave. The phase argument is used as an index to find the according function value in the table. Since a lookup table has a limited number of entries, most of the times the selected value is not the exact function value for the given index. Hence, error noise is added to every sinusoid signal. There are multiple variations of the table-lookup technique, which differ in their signal-to-noise ratio (SNR) for a given table size, but also need a different number of arithmetical operations per lookup. A comparison was done by Moore 1977, who examind the SNRs of every version. A lookup table is most effectively used when it has a number of 2^n entries. Given the phase as a fixed point argument with a word length of m bits, where $m \geq n$, it can either be truncated or rounded to the correct word length. In a third version, the result is interpolated between the two closest table entries. The table size grows exponentially to the SNR. For the truncation method, the SNR is $6(n-2)$ dB, while for the rounding method, it is $6(n-1)$ dB. The interpolated method has even a SNR of $12(n-1)$ dB. To reach a fixed SNR of, say, approximately 60 dB, the rounding method would need 2^{11} table entries, while the interpolated version only needs 2^6 table entries. A table size reduction of this dimension generally also results in a faster table lookup, since the cpu cache can be used more efficiently. Nonetheless, the interpolated table-lookup algorithm needs two table lookups and additional arithmetical operations per sample, which nullifies these advantages. For this reason, we use a simple table-lookup algorithm with phase rounding. The table stores only a quarter of the sine function, since the other parts can be derived from the symmetry properties of the function. The resulting block diagram is shown in figure 2.4

3 Implementation

3.1 Architecture

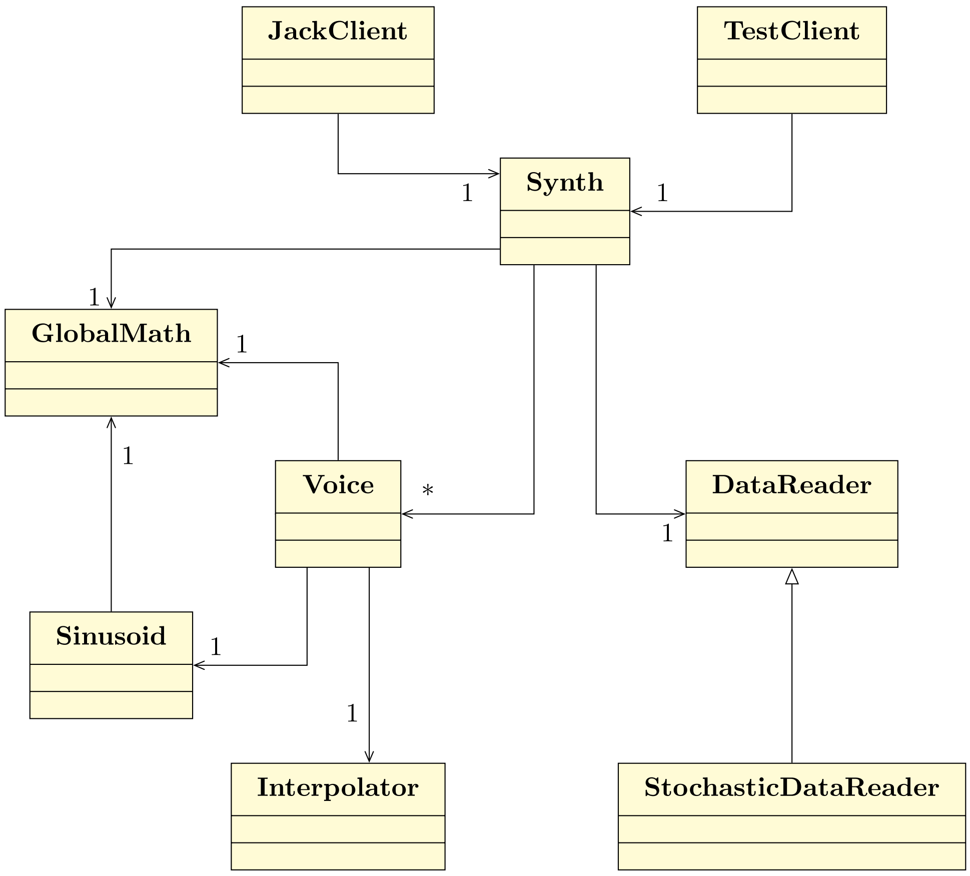

FIGURE 3.1: A simplified class diagram of the application. The `Synth` instance is either embedded into a `JackClient` or a `TestClient` instance. It owns a `DataReader` instance, which, depending on the mode, may also be a `StochasticDataReader` instance.

The system was implemented in C++ on a 64-Bit Linux system using a CPU of x86-architecture. As a MIDI- and audio interface, the sound server JACK was used (*JACK* 2017, Newmarch 2017). This software guarantees a permanent low latency and offers MIDI and audio routing between different applications. The application takes the role of a client, which offers a callback function to the JACK server. For each audio frame, this callback function is invoked by JACK, where MIDI events are passed to and the resulting audio frame is taken from the client. The class `JackClient` implements this callback function and thus serves as a wrapper, which organizes the communication with the jack server. It transforms MIDI messages coming from the server to an internal control event format. The main class `Synth` does the actual synthesis and is designed independently of the surrounding wrapper. There exists a second wrapper,

called `TestClient`. This class also serves as a wrapper for `Synth` and enables the code to be executed in an offline mode for testing and debugging purposes. It simulates the process of playing a note by sending a note-on event, a rash of modulation events and a final note-off event.
The events are passed to the `Synth` class, the main class of the program, which is responsible for the creation of other program parts. It creates, stores and removes `Voice` instances as a reaction to note events. Since the application is designed as a polyphonic synthesizer, it can store and synthesize multiple voices at the same time. Every voice can be addressed by its note index n . Depending on its mode setting, `Synth` creates an instance of `DataReader` in static mode or an instance of `StochasticDataReader` in stochastic mode. `DataReader` reads the amplitudes and frequencies of fixed partial trajectories into an internal, four dimensional data representation. This representation corresponds to the data matrices $F_{n,v}$ and $A_{n,v}$ introduced in 2.4.2. `StochasticDataReader` is an extension to `DataReader`, which is also able to read ICDFs. A more detailed description of both data formats will be done in section 3.1.3.
The `Voice` class controls the synthesis of a single voice. It uses the current modulation values to select the proper data vectors for frequency and amplitude. Furthermore, it organizes the interpolation on the pitch and the velocity axis. The interpolation functions according to section 2.6 are implemented in the `Interpolator` class. The interpolated values are given to a `Sinusoid` instance. This class executes the actual synthesis for one frame. It also interpolates the phase and amplitude of every partial on the time axis as described in section 2.5.2. The `GlobalMath` class offers data and methods, which are precomputed in the initialization phase of the program. This helps to save performance, since some of the data is reused frequently. For example, it initializes and stores the lookup table, which is needed for the sinusoid generation. The structure of the program is displayed in the UML class diagram in figure 3.1.

3.1.1 Interpolation Scheme

As figure 3.2 shows, the interpolation process is done in four steps, and the interpolation on the time axis appears twice. To understand why this has not been done in one step, it is important to know, that the time axis can be subdivided using different units. The largest unit is the synthesis window, whose size is the analysis hop size, that is, the number of samples between two successive analysis windows. Although it is possible also to choose a different synthesis window size than the analysis window size, it is not recommended, when the synthesized timbre is supposed to be as close as possible to the original timbre. The JACK server itself processes audio signals frame-wise, as it is typical for many digital audio processing systems [1]. These are kept in a limited number in an internal buffer. The frame size is defined by the user and has direct impact on the latency of the system. To keep the latency low, the frames need to be short. This may result in a frame size which is only a fraction of the synthesis window size. For our given data, an analysis window hop size of 256 samples has been used. If the JACK frame size is set to 64 samples, every synthesis window k consists of 4 frames. That is, the vectors $\vec{f}$ and $\vec{a}$ will only be refreshed on every fourth frame. As described in section 2.5.2, we want to interpolate frequency and amplitude linearly from the left boundary to the right. To enable this, the boundary values for $\vec{f}$ and $\vec{a}$ of every frame need to be interpolated according to the frame position in the synthesis window. After

[1]Note that in the JACK documentation, one *frame* is used synonymous to one *sample per channel*. In this thesis, we use the term synonymous as a *sequence of samples*.

the values were interpolated on the velocity and pitch axis, the instantaneous phase and amplitude can finally be computed for each sample.

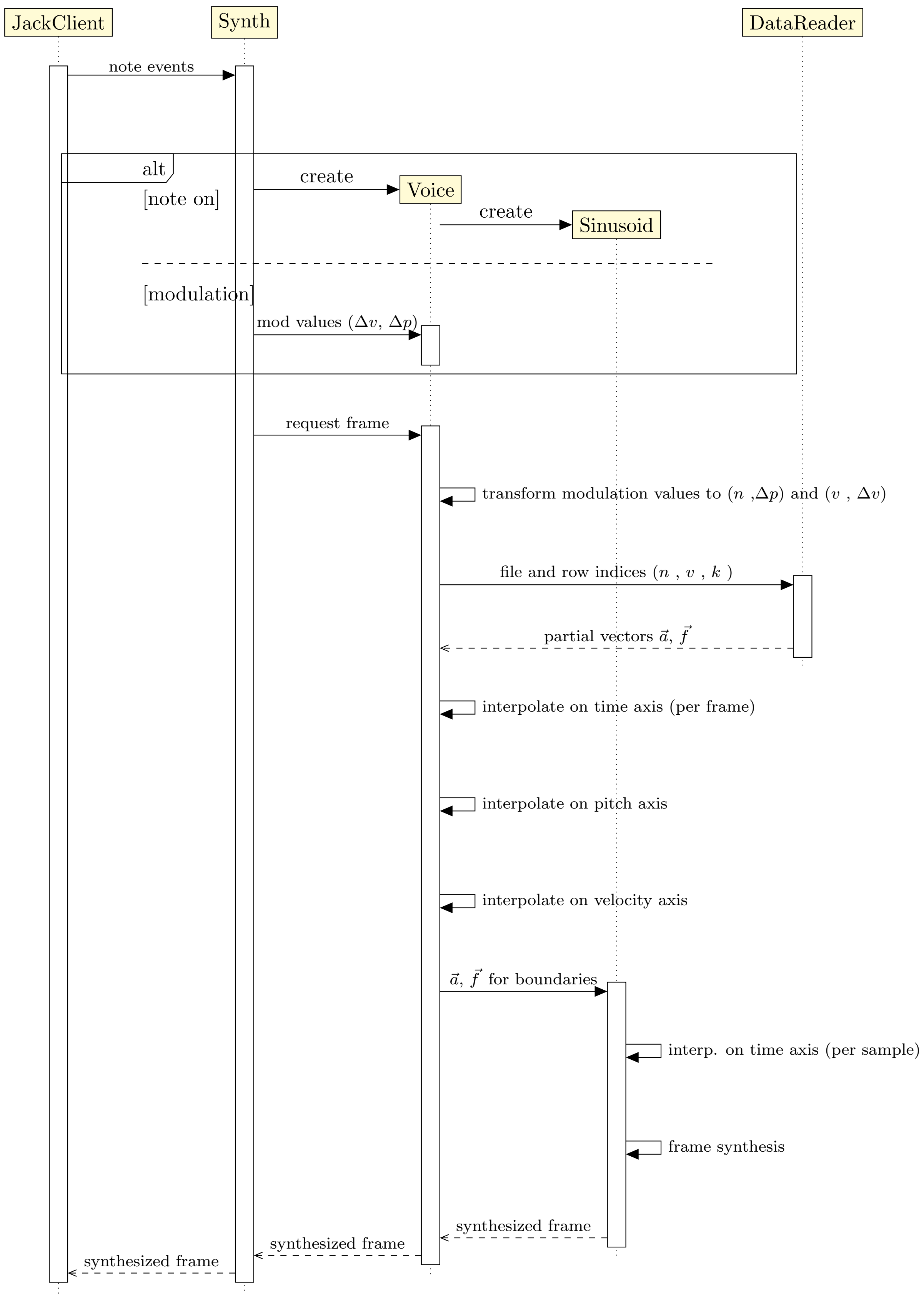
JackClient
Synth
DataReader
note events
alt
[note on]
create
Voice
create
Sinusoid
[modulation]
mod values $(\Delta v, \Delta p)$
request frame
transform modulation values to $(n, \Delta p)$ and $(v, \Delta v)$
file and row indices (n, v, k)
partial vectors $\vec{a}, \vec{f}$
interpolate on time axis (per frame)
interpolate on pitch axis
interpolate on velocity axis
$\vec{a}, \vec{f}$ for boundaries
interp. on time axis (per sample)
frame synthesis
synthesized frame
synthesized frame
synthesized frame

FIGURE 3.2: A simplified sequence diagram which shows the general procedure of the synthesis of a frame. The call is initiated by a `JackClient`, but could also be done by a `TestClient`. The alt fragment shows the creation of a `Voice`, which only happens due to a note-on message, or, alternatively, the modulation of an existing voice.

3.1.2 Control Protocoll

Control Events

The system internally uses a control event format. The `JackClient` instance converts incoming MIDI messages into control events of this type. Hence, the systems works independently from the actual control protocoll. A later extension will implement the conversion from OSC-messages (*OSC* 2017) as well, and this implementation can be done with almost no further modifications of the existing classes.
In the current state, a control event contains the following three parameters:

Control parameter	Description
message type	An enumeration type, which indicates the type of the message. It can assume the values • *note on*: A command to start playing a note. All remaining event values are used. • *note off*: A command to stop playing a note. • *pitch modulation*: A pitch variation. Only the pitch parameter is used. • *velocity modulation*: A velocity variation. Only the velocity parameter is used.
note (n_{in})	An integer number coding a note according to the MIDI specification (Moog 1986). This is the discrete part of the pitch value.
pitch (Δp_{in})	A real number, which is used for the vernier adjustment of the pitch. A value of 0 stands for no pitch offset, a value of -1 shifts the pitch by one semitone lower, and a value of 1 shifts the pitch by one semitone higher. The maximum range can be defined in the configuration file. The controllers parameter range will be mapped to this value. For example, the range of MIDI values generated by a pitch bend wheel, which is $[0, 16383]$, would be centered to zero and then scaled to the defined range.
velocity (Δv_{in})	A real number between 0 and 1. It integrates the velocity value of a MIDI note-on message as well as the value of a polyphonic aftertouch message.

Control Parameter Mapping Depending on Mode and State

The control messages are analyzed by the `Synth` class, which, depending on the message type, either starts a new voice, deletes an existing voice, or modulates the parameters pitch or amplitude of an existing voice. When the parameters note, pitch and velocity are passed to a voice, a second transformation to the parameters n , v , Δp and Δv is done. The resulting values are depending on the current state of the voice. If the application runs in stochastic mode, the voice passes the states attack and release, where extrapolation is used. Thus, as long as the `Voice` remains in the current state, amplitude and frequency values are constantly taken from the same matrices $A_{n,v}$ and $F_{n,v}$. That is, when a modulation occurs, n and v remain the same, while Δp and Δv may assume values greater than one. The parameters for the pitch extrapolation are defined as

$$n = n_{\text{init}}, \tag{3.1}$$
$$\Delta p = (n_{\text{in}} - n_{\text{init}}) + \Delta p_{\text{in}}, \tag{3.2}$$

where n_{init} is the input note at the beginning of the state. The velocity parameters are defined as

$$v = v_{\text{init}}, \tag{3.3}$$
$$\Delta v = \Delta v_{\text{in}} \cdot 4 - v_{\text{init}} - 1, \tag{3.4}$$

where v_{init} is the velocity index at the beginning of the state. As noted in section 2.6, this is only one approach to avoid problems with the interpolation between matrices, which are of different size on the time axis. The sustain state uses interpolation, since the problem cannot occur in the stochastic partial generation, which is time-independent.
The deterministic mode solves the problem differently. To use extrapolation only would result in a bigger timbre error, when a modulation over multiple dynamic levels or semitones occurs. Since in the deterministic synthesis may take up to a few seconds (which is the duration of the recording), the probability of such a deviation is much higher than it is in the shorter attack and release parts in the stochastic mode. As soon as one of the matrices ends, zero vectors are used for the amplitude interpolation instead. In case of a frequency interpolation, the interpolation value is simply set to the only existent value. The control parameters are mapped as follows:

$$n = n_{\text{in}} + \text{floor}(\Delta p_{\text{in}}), \tag{3.5}$$
$$\Delta p = \Delta p_{\text{in}} - \text{floor}(\Delta p_{\text{in}}), \tag{3.6}$$

where the function floor(x) rounds its argument down, and

$$v = \text{ceil}(v_{\text{in}} \cdot 4) - 1, \tag{3.7}$$
$$\Delta v = (v_{\text{in}} \cdot 4) - v, \tag{3.8}$$

where the function ceil(x) rounds its argument up.

3.1.3 Data Structure

To be able to use both modes, the analysis data has to be stored in 2 different file formats. Both modes use the same mapping file, which maps a note number and a velocity level to the file containing the according analysis data. In the deterministic mode, a simple csv file is used, where every row contains a partial vector. In the stochastic mode, the YAML data serialization format is used (Ben-Kiki, Evans, and Net 2009). YAML enables complex hierarchical data structures to be serialized in a human-readable text format. Every YAML file contains deterministic data for the attack and release part, as well as the discrete CDF functions. In addition, every YAML file contains meta data about the analysis process. For the future development of the application, an adaptive analysis and synthesis process can be considered: For example, the analysis hop size might be chosen individually for each analysis window.

3.2 Testing

To facilitate the development of the synthesis application, a focus was set on applying software testing on two levels: Unit testing and system testing.

3.2.1 Unit Testing

The unit testing enables to reduce the probability of errors in the code. During the development, new features were encapsulated into own functions and classes (units), which themselves (in most of the cases) are elemental. That is, they do not invoke other units in their code. Before a unit was implemented, a test was written, which were included into a so-called test suite of the Boost unit test framework Rozental and Enficiaud 2016. This framework organizes the automatic execution of the unit tests. Some parts of the code have not been tested with unit tests, since these parts mostly are higher-level functions which pass results from one unit to another. The unit testing was also done to ensure that the future development of the system will not introduce errors into the working units.

3.2.2 System Testing and Evaluation

To make sure that the audio output produced by the system fits to the systems description in this thesis, a visual and auditive evaluation was done. For the visual analysis, we used MATLAB together with MEX. MEX stands for *MATLAB external* and is an API which offers the integration of C and C++ source code into MATLAB. A self-defined MEX function can be used to make C++ functions available in MATLAB, since it then can be used as normal MATLAB function. Functions from the MEX API are used to receive parameters from a MATLAB command, then the code is executed and then the result is again passed by a MEX function to the MATLAB environment. The source code is compiled by a MATLAB command, which uses a suitable C/C++ compiler, which was in our case the G++ compiler. We call this type of MEX function a *MEX wrapper*. We created two wrappers for the classes `Sinusoid` and `TestClient`. The former offers the function `sinusoid_process_mex`, which lets the user pass an amplitude and a frequency vector for both endings of a frame. A `Sinusoid` instance then uses this data to compute the instantaneous amplitude and phase from the first to the last frame index and then synthesizes the actual audio signal. The wrapper returns not only the signal, but also the instantaneous amplitude and phase. These values can be visualized with the MATLAB script `sinusoid_synthesize_test.m`. During the development, the behavior of the sinusoid class was permanently controlled using this visualization. If the interpolation curves did not behave as they were supposed to, or if the waveform itself contained an obvious error, this concept enabled the detection of some errors, for which the unit testing alone was unsuitable.
The other wrapper offers an interface to the class `TestClient` and hence offers access to the whole synthesis system by using the MATLAB command `synth_process_mex`. To the according MATLAB function, an array of structs, similar to the control events described in 3.1.2, the number of frames to be synthesized, the number of partials and the chosen synthesis mode is passed. The function returns a signal which is equal to the processed signal of the online version of the system. It also returns the partial trajectories of amplitude and phase of the whole synthesized signal. The MATLAB function `synth_process_visualize.m` executes an amplitude modulation and visualizes the result. Hence, the playing and modulation of a note can be simulated and examined using the MATLAB plot functions. The function allows to choose between

different visualizations, which include the waveform, amplitude and frequency trajectories and the pitch and velocity modulation curves. This was already done during the development process and gave a good visual feedback of the latest code changes.

4 Evaluation

4.1 Usability Evaluation

Our evaluation includes only a basic usability test of the synthesis system. This included the playing of the system using a pad controller. We used the Livid Base pad controller (*Livid Instruments - Base II* 2017) for this purpose, which features a grid of 32 pressure-sensitive buttons. The buttons support aftertouch control, which enable a continuous velocity modulation. However, it has no additional input for continuous pitch modulation.

4.1.1 Modulation

The continuous variation of the velocity created a seamless interpolation between the data matrices of the four different velocity levels. Problems can occur during a fast modulation at the beginning and especially at the end of the note: If the velocity of the note is raised, it is possible that the sound signal will decrease in amplitude, since the higher velocity data matrix ends earlier than the lower velocity data matrix. This problem only appears in the deterministic mode, because the stochastic mode uses extrapolation in the deterministic attack and release parts. Another advantage of the stochastic mode is, that after a note-off message, the sound fades out in a natural way. In the current version of the deterministic mode, the sound simply stops playing.

4.1.2 Latency

To keep the latency low, the frame size and the number of frames stored in the buffer need to be as small as possible. If too small values are chosen, the JACK server reacts with XRUNS. These messages are logged by JACK if a client is not able to compute an audio frame in the available time. They are also audible as interruptions in the audio signal. The computation time of a frame can be modeled as follows:

$$T_{\text{total}} = T_{\text{const}} + T(x). \tag{4.1}$$

T_{const} is the time which is independent of the actual frame size, and which includes the data selection respectively generation of the partial vectors and the interpolation on the pitch and the velocity axis. However, it is still linear dependent on the number of partials used. In the stochastic mode, the influence of this part is significantly higher than in the deterministic mode, because the partial vector generation by inverse transform sampling is computationally more intensive. $T(x)$ is the processing time which increases linearly with the frame size x. It includes the time scale interpolation and the sinusoid generation. The lower boundary of the frame size is thus determined by T_{const}.
Our test computer used a low-latency 64-Bit Linux on a Intel Core i5-3320M CPU and a M-AUDIO Fast Track audio interface. We chose a sampling rate of 44.1 kHz. The deterministic and the stochastic mode were tested separately, using 30 partials for the deterministic and 15 partials for the stochastic mode. It is necessary to reduce the

number of partials for the stochastic mode, since these highly raise the need of computational resources. A test with 30 partials was even not succesfull at a frame size of 512 samples. Another important parameter which influences the performance is the lookup table size, which was set to 2^{16} values. Logarithmic interpolation respectively extrapolation was used for the frequency depending on the pitch value and for the amplitude depending on the velocity value. On the other axes, linear interpolation was used. As second measure additionally to appearing XRUNS, the JACK dsp load was used. This value displays the current CPU workload in percent. The first XRUNS appear approximately at 50 % dsp load. The measured values can only be an approximate measure for the computationally intensivity of the application, since the DSP load also depends on other processes running on the computer.

Deterministic Mode Using a frame size of 64 samples and three frames per buffer (4.35 ms latency), it was possible to play a single tone at a DSP load of approximately 28 % without causing XRUNs. The first XRUNs occurred already when two voices were played simultaneously. With a frame size of 128 samples and three frames per buffer (8.71 ms latency), it was possible to play a single tone at a DSP load of approximately 20 % without causing XRUNs. In this case, the first XRUNs occurred when 3 voices were played simultaneously. For a very high frame size of 512 samples (34.8 ms latency) and three frames per buffer, a tone could be played at approximately 11 % dsp load. It was possible to play up to 9 tones at the same time.

Stochastic Mode With a frame size of 512 samples and three frames per buffer, a single tone could be played at a DSP load of approximately 28 %. Three tones simultaneously caused XRUNs. Smaller frame sizes were not possible using a number of 15 partials. Note that additional latency can also be caused by zero rows at the beginning of data matrices. This can result in a much bigger latency than the actual latency caused by the chosen frame size.

4.2 Synthesis Evaluation

The auditive evaluation of the synthesized signal was done by a simple listening test by the author of this thesis. The signals were also examined by the visualization tool described in section 3.2.2.

4.2.1 Deterministic Mode

An unmodulated synthesized signal of a fixed note was compared to its original sound file using Superlux HD 681 monitor headphones. The deterministic signal stayed very close to its original. A clearly noticeable difference was the absence of noise, which was present in the original recording. Modulated tones behaved as expected and interpolated seamlessly. However, a natural sounding modulated tone is also determined by the capabilities of the controller. The figures 4.1, 4.2 and 4.3 show an unmodulated, a pitch modulated and a velocity modulated signal. In the author's opinion, the quality of the synthesized signal was satisfying. Of course, a more significant and detailed result can only be achieved with an empiric listening test with a sufficient number of participants.

4.2.2 Stochastic Mode

The stochastic signal was still perceived as related to a violin tone, but in a very bad and distorted quality. The state transitions were clearly audible. As figure 4.4 shows, in the sustain part the amplitude behaves very differently than in the deterministic mode. Deterministic and stochastic signals of the same tone have the same probability distribution in amplitude and frequency, and both parameters move in the same interval. But in a natural signal, the amplitude takes some time to move from one extremum to another. In the stochastic mode, for each frame, the amplitude hops from one value to another, regardless of the amplitude value of the previous frame. Since the amplitude values are generated by a uniform noise generator, this noise can be relocated in the signal and is also clearly audible.
Another problem is, that the original recordings still rise and fall in amplitude during the selected sustain parts. This enlarges the range of the possible output values of the ICDF and results in a louder noise. Yet, to execute the recording and segmentation step more accurately will still not lead to a satisfying solution. It seems, that our model in its current version is not yet suitable to model the parameters $a_{\text{stoch},i}$ and $f_{\text{stoch},i}$ in a satisfying way. In the authors opinion, a modification is required, which also considers the previous values of amplitude and frequency.

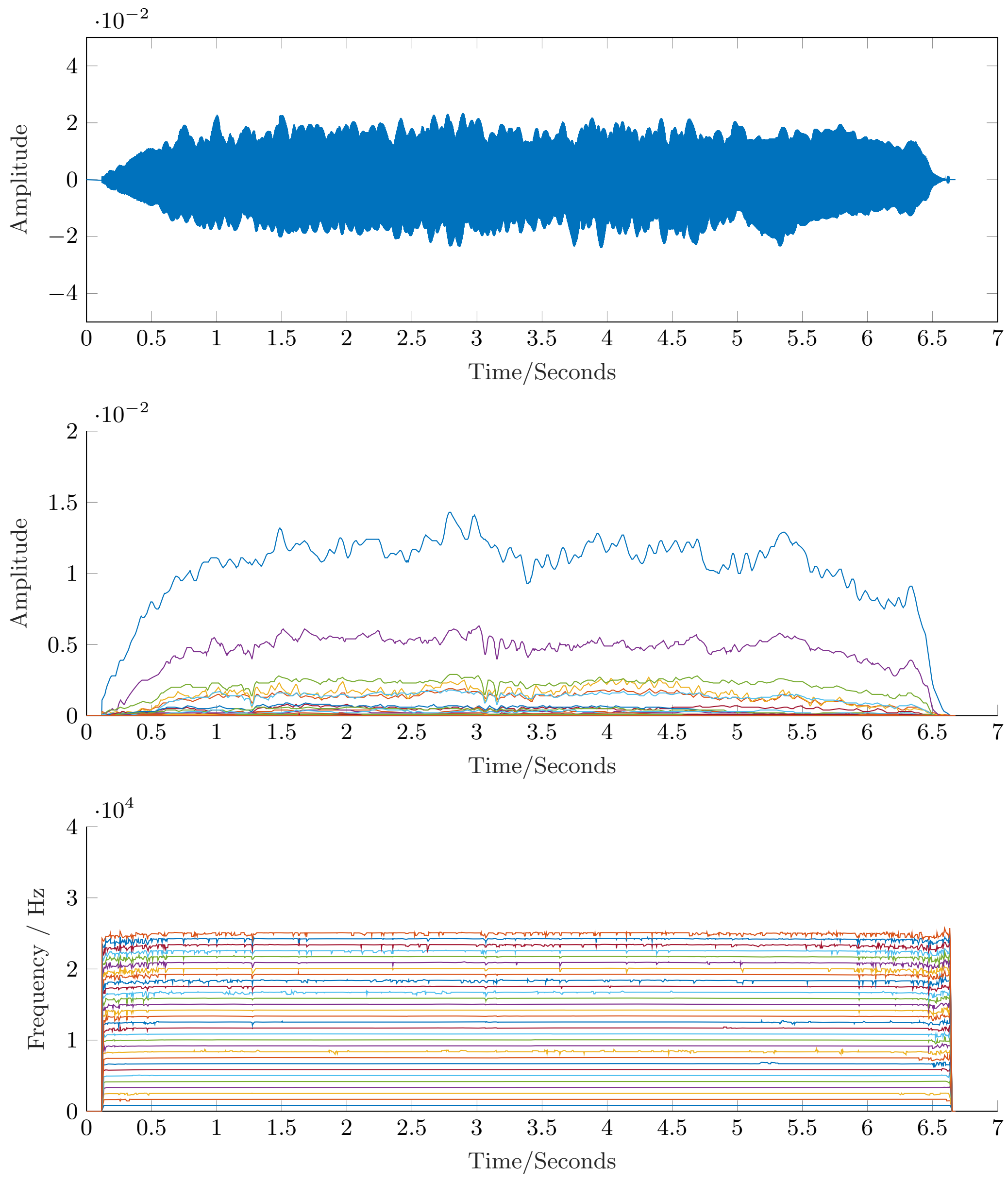

FIGURE 4.1: An unmodulated signal which was synthesized in the deterministic mode.

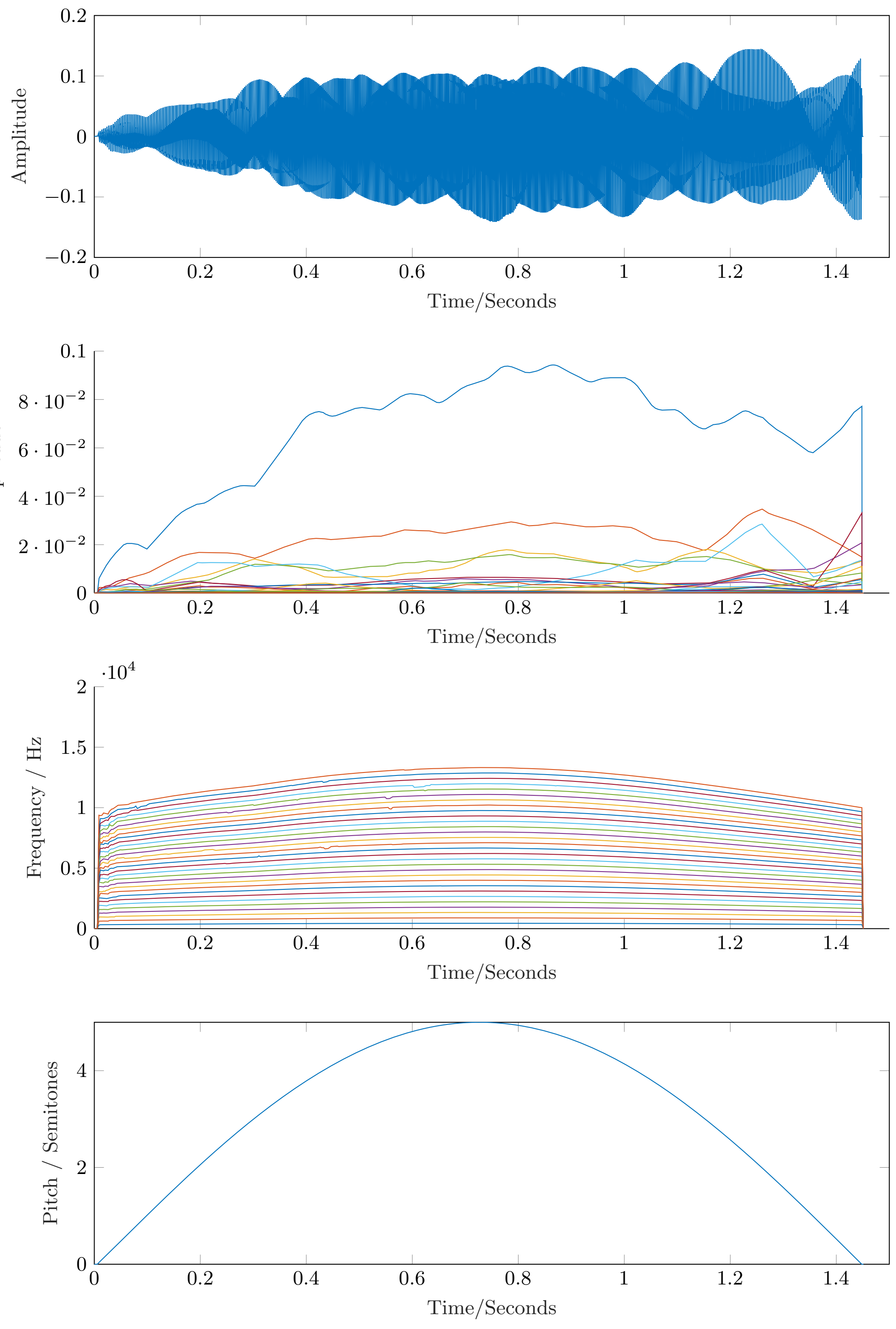

FIGURE 4.2: A deterministic signal, where the pitch is modulated over five semitones.

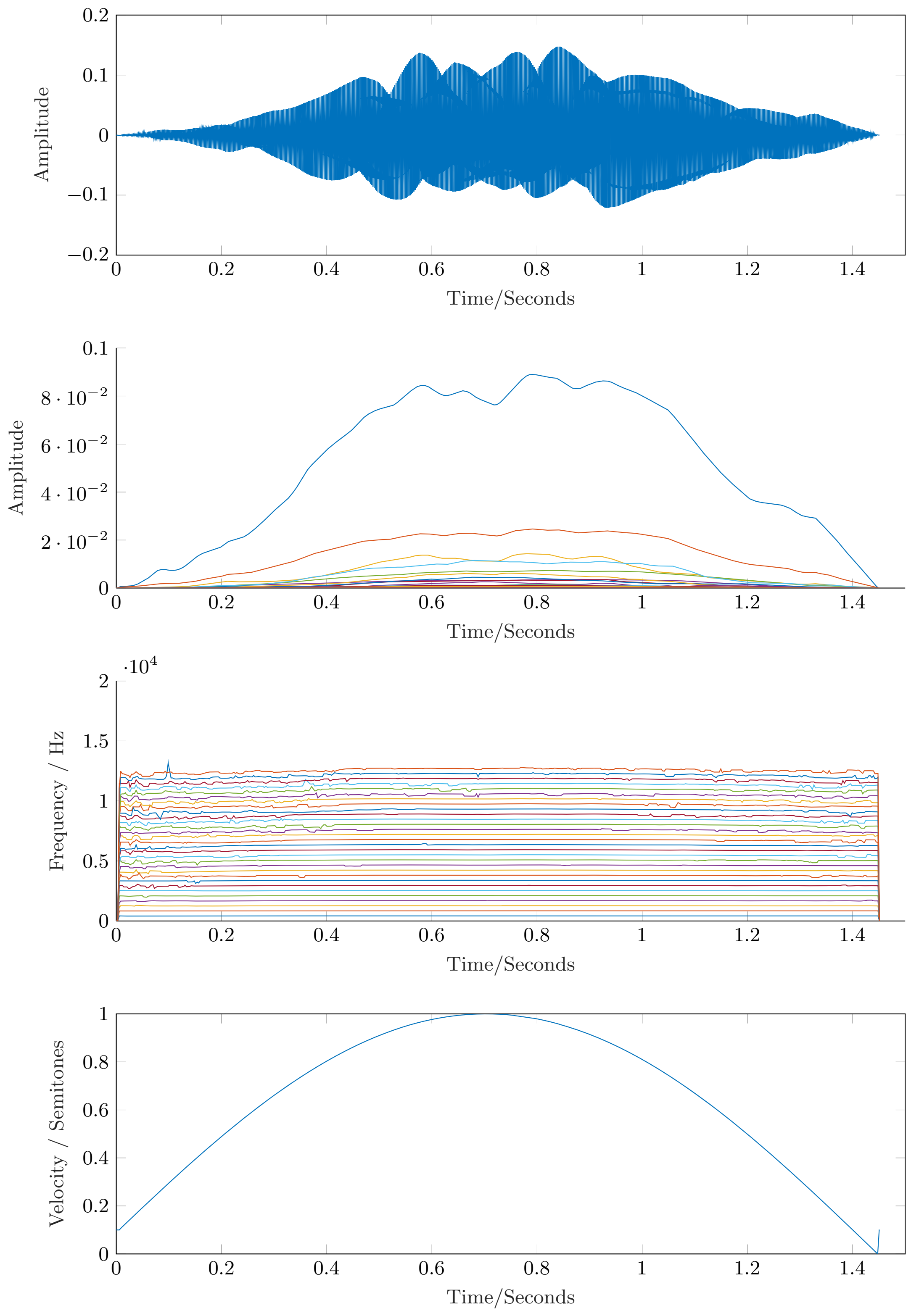

FIGURE 4.3: A determinstic signal, where the velocity is modulated over the whole velocity scale. On the partial frequency graph, the timbre change over the different velocity levels can be observed: the frequency variation becomes less, when the velocity increases.

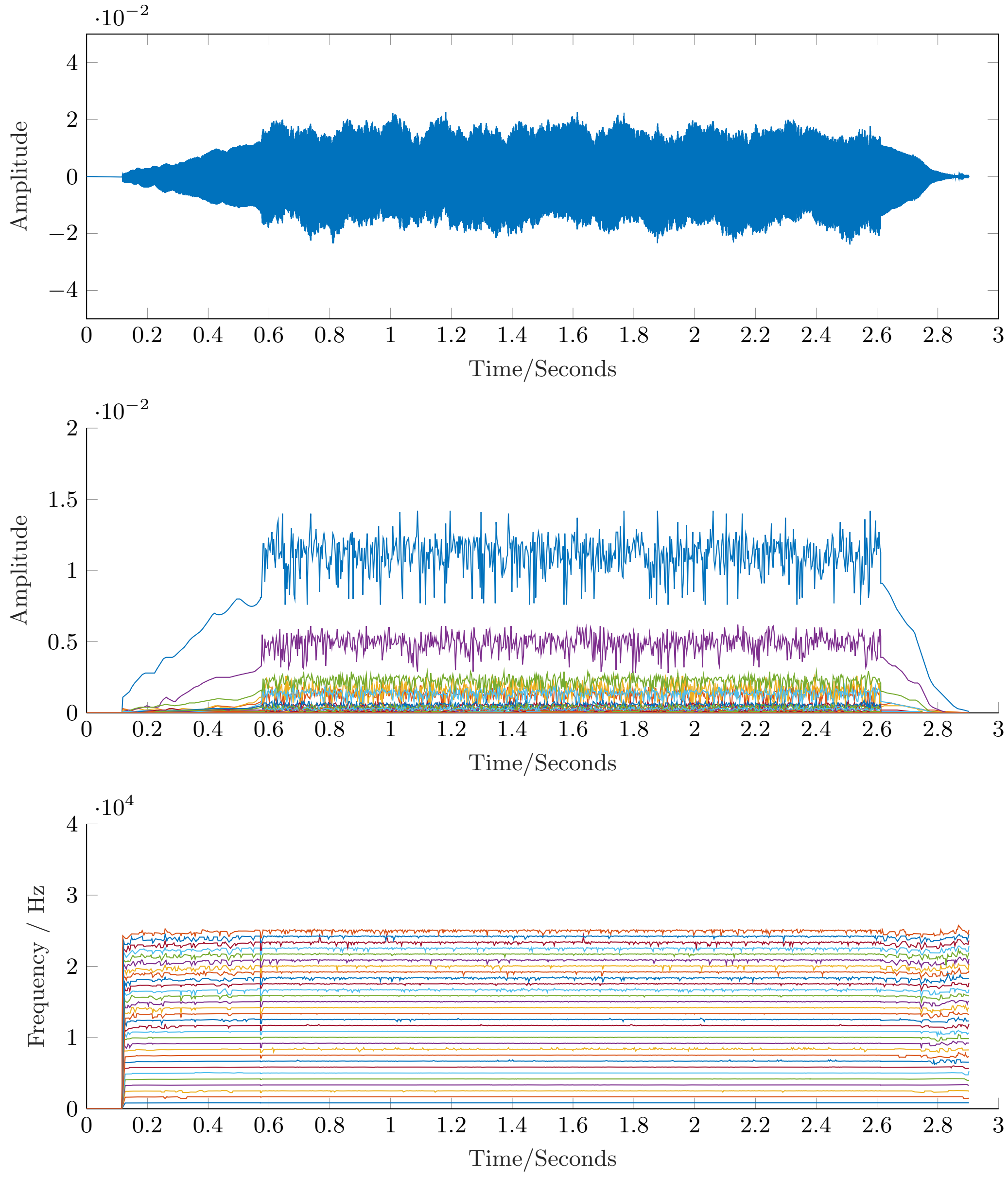

FIGURE 4.4: A stochastic, unmodulated signal. The change to the sustain part is clearly visible on the partial amplitude graph, since the signal becomes very noisy.

5 Conclusion

We successfully implemented an additive sound synthesis system, which can be used in live performances and studio production. An additional component for the noise generation is needed to make the synthesis more powerful. Also, the controllability can yet be extended. The abilities of additive synthesis to control timbre has not yet been exploited. To achieve this, a mapping concept has to be found.
More evaluation needs to be done on the quality of the sound signal in the deterministic mode. Since several parameters influence the need of computational resources on the one hand and the signal quality and usability of the system on the other hand, for each of these parameters a threshold of the just noticeable difference (JND) should be found. Latency is definitely an important aspect of the usability of the system, but it is also a matter of the used controller (Mäki-patola and Hämäläinen 2004). Another interesting parameter which might influence the usability and/or the signal quality is the interpolation type on the pitch-velocity-grid. A linear interpolation might be a computational less intensive alternative for logarithmic interpolation. However, its impact on the usability and the modulation quality has not yet been measured. Two other important parameters which only influence the signal quality are the lookup table depth, which influences the SNR, and the number of partials. Additional listening test can be applied to find a JND for these, although existing research may already contain satisfying results. To achieve the best compromise of quality and performance, these parameters have to be set closely above the JND.
The problem of the interpolation between parameter matrices of different length can be solved by using only extrapolation. Yet, this would nullify the advantage of the dynamic timbre selection during a pitch or velocity modulation. This is the most important reason to find a robust time-independent data generation system. The design of the stochastic mode of our application was a first attempt to achieve this goal. However, the result was not yet satisfying and computationally to intensive. We believe, that in a further development of this technique, a random value should not only be generated by a global ICDF, but by a ICDF dependent on the previously generated value. This would avoid noise-like signals and lead to smoother amplitude and frequency trajectories. If we call the currently used ICDFs an ICDF of first order, it is possible to compute a new ICDF for every possible output value of the ICDF of first order. This would then be an ICDF of second order. However, this raises new problems: It needs even more arithmetical operations for the vector generation, and the need of memory grows exponenentially. The former is a problem, since our evaluation showed, that the current need of resources is already critical. These problems might be solved by further code optimizing. An approach to solve the latter problem might be to find an polynomial approximation for the ICDFs (Olver and Townsend 2013), so that these can be represented with only a small fraction of the memory needed for a discrete ICDF representation. Another focus needs to be set on note transitions: In the current state, timbre changes introduced by the modulation process itself are not considered. Overall, using stochastic techniques for additive synthesis seems to have a high potential, which has not yet been discovered.

Bibliography

Ben-Kiki, Oren, Clark Evans, and Ingy döt Net (2009). *YAML Ain't Markup Language (YAML^{TM}) Version 1.2.* URL: http://www.yaml.org/spec/1.2/spec.html (visited on 05/20/2017).

Cannam, C., C. Landone, and M. Sandler (2010). "Sonic Visualiser: An Open Source Application for Viewing, Analysing, and Annotating Music Audio Files". In: *Proceedings of the ACM Multimedia 2010 International Conference.* Firenze, Italy, pp. 1467–1468.

Devroye, Luc (1986). "General Principles in Random Variate Generation". In: *Non-Uniform Random Variate Generation.* DOI: 10.1007/978-1-4613-8643-8_2. Springer New York, pp. 27–82.

Goodwin, Michael and Alex Kogon (1995). "Overlap-Add Synthesis Of Nonstationary Sinusoids". In: *Proc. Int. Computer Music Conf.*

Goodwin, Michael and Xavier Rodet (1994). "Efficient Fourier synthesis of nonstationary sinusoids". In: *Proceedings of the International Computer Music Conference.* INTERNATIONAL COMPUTER MUSIC ACCOCIATION, pp. 333–333.

Hodes, T. et al. (1999). "A fixed-point recursive digital oscillator for additive synthesis of audio". In: *1999 IEEE International Conference on Acoustics, Speech, and Signal Processing. Proceedings. ICASSP99 (Cat. No.99CH36258).* 1999 IEEE International Conference on Acoustics, Speech, and Signal Processing. Proceedings. ICASSP99 (Cat. No.99CH36258). Vol. 2, 993–996 vol.2.

Houtsma, A.J.M. (1997). "Pitch and timbre: Definition, meaning and use". In: *Journal of New Music Research* 26.2, pp. 104–115.

Hunt, Andy, Marcelo Wanderley, and Ross Kirk (2000). "Towards a model for instrumental mapping in expert musical interaction". In: *Proceedings of the 2000 International Computer Music Conference*, pp. 209–212.

JACK (2017). *JACK Audio Connection Kit.* URL: http://jackaudio.org/ (visited on 05/08/2017).

Livid Instruments - Base II (2017). URL: http://lividinstruments.com/products/base/ (visited on 05/21/2017).

Mäki-patola, Teemu and Perttu Hämäläinen (2004). "Latency Tolerance for Gesture Controlled Continuous Sound Instrument Without Tactile Feedback". In: *Proc. International Computer Music Conference (ICMC*, pp. 1–5.

McAulay, R. and T. Quatieri (1986). "Speech analysis/Synthesis based on a sinusoidal representation". In: *IEEE Transactions on Acoustics, Speech, and Signal Processing* 34.4, pp. 744–754.

Moog, Robert A. (1986). "MIDI: Musical Instrument Digital Interface". In: *Journal of the Audio Engineering Society* 34.5, pp. 394–404.

Moore, F. Richard (1977). "Table Lookup Noise for Sinusoidal Digital Oscillators". In: *Computer Music Journal* 1.2, pp. 26–29.

Moorer, J. A. (1977). "Signal processing aspects of computer music: A survey". In: *Proceedings of the IEEE* 65.8, pp. 1108–1137.

Newmarch, Jan (2017). "Jack". In: *Linux Sound Programming.* DOI: 10.1007/978-1-4842-2496-0_7. Apress, pp. 143–177.

Olver, Sheehan and Alex Townsend (2013). “Fast inverse transform sampling in one and two dimensions”. In: *arXiv:1307.1223 [math, stat]*. arXiv: `1307.1223`.

Phillips, Desmond Keith (1996). “Digital Sine Oscillator Design”. In: *Algorithms and architectures for the multirate additive synthesis of musical tones*. Durham University, pp. 106–125.

Qian, Xiaoshu and Yinong Ding (1997). “A phase interpolation algorithm for sinusoidal model based music synthesis”. In: *1997 IEEE International Conference on Acoustics, Speech, and Signal Processing*. 1997 IEEE International Conference on Acoustics, Speech, and Signal Processing. Vol. 1, 451–454 vol.1.

Rodet, Xavier and P. Depalle (1992). “Spectral Envelopes and Inverse FFT Synthesis”. In: AES Convention 93.

Rozental, Gennadiy and Raffi Enficiaud (2016). *Boost.Test - 1.64.0*. URL: `http://www.boost.org/doc/libs/1_64_0/libs/test/doc/html/index.html` (visited on 05/21/2017).

Serra, X (1997). “Musical Sound Modeling with Sinusoids plus Noise”. In: *Musical Signal Processing*. Studies on New Music Research. Swets & Zeitlinger, pp. 91–122.

Smith, Julius O. (1999). *Viewpoints on the History of Digital Synthesis*.

Smith, Julius O and Perry R Cook (1992). “The second-order digital waveguide oscillator”. In: *Proceedings of the International Computer Music Conference*. Citeseer, pp. 150–150.

OSC (2017). *The Open Sound Control 1.0 Specification*. URL: `http://opensoundcontrol.org/spec-1_0` (visited on 05/29/2017).

Treindl, Gabriel (2016). “Entwicklung und Evaluation eines Controllers zur Tonhöhensteuerung über Vier-Finger-Kombinationen”. Master’s Thesis. Berlin: Technische Universität Berlin.

Turner, C. S. (2003). “Recursive discrete-time sinusoidal oscillators”. In: *IEEE Signal Processing Magazine* 20.3, pp. 103–111.

von Coler, Henrik and Alexander Lerch (2014). “CMMSD: A Data Set for Note-Level Segmentation of Monophonic Music”. In: Audio Engineering Society Conference: 53rd International Conference: Semantic Audio. Audio Engineering Society.

von Coler, Henrik and Axel Röbel (2011). “Vibrato Detection Using Cross Correlation Between Temporal Energy and Fundamental Frequency”. In: Audio Engineering Society Convention 131. Audio Engineering Society.

von Coler, Henrik, Gabriel Treindl, et al. (2017). “Development and Evaluation of an Interface with Four-Finger Pitch Selection”. In: Audio Engineering Society Convention 142. Audio Engineering Society.