DIGITAL SOUND TEXTURE SYNTHESIS: FINAL REPORT

Jordie Shier
University of Victoria

ABSTRACT

A new audio synthesizer plugin inspired by the data-driven synthesis paradigm introduced by TAPESTREA has been developed. The unique compositional opportunities that TAPESTREA provides, coupled with the potential to improve upon the component algorithms, and the benefits of the VST/AU plugin framework were all motivators for a new project design. A reduced-scope version of the initial proposed design has been implemented and an overview of the development completed is provided. The resulting product is a sinusoidal plus stochastic modelling synthesizer plugin with real-time playback capability in a host digital audio workstation.

1. INTRODUCTION

A new technique for composing environmental sounds was developed by Ananya Misra, Perry R. Cook and Ge Wang of Princeton University in 2006 [17, 18]. This technique brought together three methods of sound analysis and synthesis into one package called TAPESTREA, or Techniques and Paradigms for Expressive Synthesis, Transformation and Rendering of Environmental Audio. TAPESTREA allows users to transform and re-synthesize existing sounds to realize new compositions. Composers of Musique Concrète and Soundscapes [33] were indicated as being potential adopters of this tool given the ability to re-compose sounds on a spectrum from "found" (unchanged) to completely abstract by altering the amount of transformation applied. The film, theatre, and game industries are also indicated as being potential adopters of this technology.

While TAPESTREA is a sophisticated framework for synthesizing musical compositions from existing sound, Misra et al. acknowledge room for further development, specifically in the analysis of input audio. Additionally, the application is now several years old (latest version released in 2009) and the provided binary and build process no longer work on the tested operating system [1] . With this in mind, a project is proposed to revive and build upon the TAPESTREA framework in order to continue exploring the synthesis and compositional opportunities that the techniques provide. An overview of TAPESTREA is provided, including a break down of the analysis and synthesis phases of the application, and areas of potential improvement based on relevant research is suggested. A new project design is then introduced which has the intention of implementing and building upon components of the TAPESTREA framework in the context of a VST [2] /AU [3] plugin. At the time of writing, this project is at the end of the development timeline. A progress report written at the midway point included a revised set of objectives for the second half of the available development time. A final report reviews the completed work in relation to these revised objectives and provides suggestions for future work.

[1] Mac OS 10.11

2. OVERVIEW OF TAPESTREA

A type of audio signal modelling describes audio as being comprised of sines, transients, and noise, and previous work has been done to split audio into these components for the purpose of compression and transformation [13,35]. TAPESTREA uses a similar model in which input audio is split into (1) deterministic sound; the sinusoidal components which generally correspond to pitched/harmonic material, (2) transients; brief noisy events such as footsteps, and (3) stochastic sound; background audio such as ocean sounds. As audio is analyzed it is split into events and classified into one of the three categories outlined above. Each event, which can be saved as a template, can then be transformed, re-synthesized, and stored for later use. Functionality for looping and sequencing templates over various temporal ranges allows for granular synthesis type effects [24] and for various templates to be arranged into a larger environmental composition.

3. ANALYSIS PHASE

The analysis phase of the TAPESTREA system receives input audio and segments it into deterministic, transient, and stochastic components. It does this by first creating a sinusoidal model of the deterministic portion of the signal which is then subtracted from the original signal. Transients are detected and sliced out. The remaining signal is the stochastic component which is used as the input to a modified wavelet tree algorithm [5]. This wavelet algorithm can temporally extend the stochastic component and "fill-in" the gaps created from slicing out the transient audio. Misra et al. acknowledge that TAPESTREA struggles when events overlap, such as a simultaneous deterministic and transient event. In this case the transient detection algorithm would cause all the signal within a short segment to be classified as a transient, even if it contained deterministic, transient, and stochastic information. Recent work in

[2] https://www.steinberg.net/en/products/vst.html

[3] https://en.wikipedia.org/wiki/Audio_Units

the area of harmonic plus percussive signal classification has the potential to improve the segmentation performed.

Initial work by Ono et al. on the classification and remixing of percussive/harmonic elements [19] was completed in 2008. The fact that in a spectrogram representation of audio, harmonic material is generally arranged horizontally while percussive material is arranged vertically is leveraged by this research. Since then work has been done to simplify the algorithm using median filters [6]. In another extension of this work Driedger introduced the concept of separation factor and residual signal which made for stricter classification of the harmonic and transient material [4]. In a further refinement, tonal sounds that have some vertical frequency structures (in a spectrogram representation), such as frequency modulated tones that should be considered harmonic but were being classified as either percussive or residual, were addressed by Füg [8].

This work could be used a pre-processing step in the TAPESTREA analysis stage before sinusoidal and stochastic modelling occurs.

3.1 Deterministic

The deterministic portion of the signal is analyzed and re-synthesized using a sinusoid modelling technique initially introduced by McAulay et al. for the purpose of speech analysis and re-synthesis [16]. The spectral modelling synthesis (SMS) technique, which is an extension of the sinusoidal modelling approach, is leveraged to extract the stochastic signal component from the sinusoidal model [30].

Levine et al. suggest an extension of the SMS technique using multi-resolution windowing in order to more accurately calculate the sinusoidal parameters [14]. This technique uses an alias-free multi-rate filterbank [7] in order to segment the input into octave wide frequency ranges before the sinusoidal peak picking and continuation process of the SMS technique. It is suggested that this technique has improved results for polyphonic material.

3.2 Transient

TAPESTREA uses a non-linear one-pole envelope follower with a quick attack and longer decay time in order to track transient sections in the input signal. This envelope follower is used to detect sudden increases in the signal energy to be marked as transients. A user defined period of time containing that transient signal is then extracted from the signal and becomes a new transient event. These brief events are stored as raw audio signals.

3.3 Stochastic

Once the deterministic and transient components of the sound source have been removed, what remains is considered the stochastic or residual component. Usually this is background noise such as traffic, crowd noise, ocean waves, etc. TAPESTREA implements a modified version of a wavelet tree learning algorithm [5] which allows for a short segment of stochastic audio material to be re-synthesized with the same characteristics as the source over an indefinite period of time.

Schwarz describes sound texture synthesis algorithms as falling into two distinct categories: rule-based or data-driven [28]. Because of the general nature of sounds that TAPESTREA expects to analyze and re-synthesize, data-driven models of sound texture synthesis are more suited for this application. These approaches require input audio which is analyzed and then re-synthesized using techniques such as granular synthesis [24], wavelet synthesis [5], or concatenative synthesis [27]. A state of the art wavelet method [10, 20] and granular synthesis method [29] are suggested by Schwarz in his 2011 State of the Art Sound Texture Synthesis article [28]. One benefit of the wavelet technique recommended by Schwarz is that it can handle stereo audio in a meaningful way.

In 2016, two more data-driven sound texture synthesis methods were proposed and reported promising results. A concatenative technique proposed by Bernardes et al. segments input audio into temporal sections with clear spectral differences and then applies a clustering algorithm to group the sections [2]. A concatenative cost algorithm determines the best location to stitch the segments back together in order to create a perceptually convincing result.

Another concatenative technique, proposed by O'Leary and Röbel, splits input audio into equally sized "atoms" which are then arranged with other "atoms" that have been deemed suitable candidates to create segments, which are then overlapped and added together to create the final synthesized texture [21].

4. SYNTHESIS PHASE

Once the analysis of an input audio source has been completed, deterministic and transient events can be transformed and re-synthesized with a generated stochastic background. All of these can also be stored as templates for recall and recombination at a later time.

4.1 Deterministic

Because of the nature of deterministic events, pitch and temporal transformations can be performed with high quality results by scaling the frequency and time values of the sinusoid parameters prior to synthesis.

Shortly after the initial release of TAPESTREA, further work was done by Lieber, Misra, and Cook to improve the usability of the deterministic synthesis engine [15]. Previously, transformations made to the deterministic templates were applied to the entire template, or all sinusoidal tracks within that template. This work enabled the user to select and transform individual sinusoidal tracks within a deterministic template.

4.2 Transient

Transient events in TAPESTREA are stored as raw audio files so they are simply played back during synthesis. Limited pitch and temporal transformations are achieved through the use of a phase vocoder [3].

4.3 Stochastic

Synthesis of the stochastic background is implemented with a modified version of the wavelet tree learning algorithm [5] implemented by Dubnov et al. The modifications include an option to apply randomness at the first step of the learning algorithm and the context can depend on the tree depth. Also, a performance optimization was made that allows the learning to occur in real-time. This optimization stops learning from running at the highest level which takes the longest time and has the least perceptual effect since it correlates with the highest frequency content.

5. EXTENDING TAPESTREA

A project to implement the ideas presented by the TAPESTREA framework and explore some of the potential improvements mentioned above is outlined below.

5.1 Objectives

5.1.1 TAPESTREA Framework Audio Plugin

The unique audio synthesis paradigm developed by Misra et al. will be revisited and components of the analysis/-transformation/synthesis approach will be implemented as a VST/AU software synthesizer so that it can be used in already widely adopted digital audio workstations (DAWs) such as Pro Tools [4] or Ableton Live [5] . Instead of templates being placed on a timeline inside of the TAPESTREA application, the plugin will receive MIDI events from the host application and in turn synthesize the desired template. Audio analysis will be performed in a similar manner and users will be able to load any external audio files into the application for analysis, transformation, and synthesis.

5.1.2 User Interface

A user interface based on the original application will be implemented. The user will be able to load in new audio for analysis, control various parameters related to analysis and synthesis, save extracted events and stochastic backgrounds as templates, apply transformations to selected templates, and layer templates for synthesis in response to a MIDI event.

5.1.3 Improvements

Once baseline algorithms for analysis and synthesis from the original TAPESTREA application are implemented, consideration of potential improvements from the recent research outlined above will be made. Of particular interest for the analysis phase is the the work on harmonic/percussive/residual segmentation [4, 8].

For the calculation of the sinusoidal model during the deterministic analysis phase, the work done by Levine et al. [14] for improved polyphonic audio response is of interest.

[4] `http://www.avid.com/pro-tools`
[5] `www.ableton.com`

The algorithm used for analysis and re-synthesis of the stochastic background will also be considered. The concatenative method developed by Bernardes et al. [2] in 2016 looks like a promising option, as well as the wavelet method developed by Kokram and O'Regam [10, 20] or the granular synthesis method developed by Schwarz et al. [29]. Because of the real-time synthesis requirements, complexity of the algorithm will be of primary concern when reviewing these options for potential implementation.

On the transform/synthesis side, the improvements made by Lieber et al. [15] are a desirable implementation. These modifications give the user finer grain control over the sinusoidal synthesis of deterministic events by enabling manipulations to individual sinusoidal tracks.

Given the available time for this project, implementing many of these potential improvements will likely be beyond the scope of the project, however they can serve as directives for future work.

5.2 Functional Requirements

- Run as VST/AU in host DAW
- Accept MIDI input from host
- Output 2 or more channels of audio in real-time, in response to MIDI event
- Load audio from file for analysis
- User control parameters for analysis [1]
- Apply pitch modifications to deterministic templates in response to MIDI pitch
- Apply pitch and temporal modifications to stochastic and transient templates
- Save extracted deterministic, transient and stochastic elements as separate templates
- Recall saved templates
- Synthesize templates in response to MIDI event
- Layer templates for creation of new sounds
- Save instances of the layered templates as presets for later recall

6. IMPLEMENTATION

6.1 Resources

This project will be implemented as a VST/AU plugin. The C++ programming language will be used with the JUCE Audio Framework [32] which is a library that enables rapid development of audio plugins. JUCE takes care of many of the integration details in plugin development and allows for developers to immediately start writing DSP code for processing and synthesizing audio.

A number of other freely available audio frameworks may also be useful in this project in addition to the JUCE

framework. A comparison of some of these is provided by Robinson and Bullock [26]. UGen++ [25] and Gamma [23] both performed well in the mixed sinusoid performance test which would make them useful for the deterministic synthesis phase.

In addition to the TAPESTREA documentation, source code and the articles referenced so far, some available textbooks have been found which will be useful resources when implementing particular portions of the application. *Applications of Digital Signal Processing to Audio and Acoustics* by Kahrs and Brandenburg [9] has a chapter on sinusoidal modelling with detailed explanations of the mathematics and implementation, including the deterministic plus stochastic model which is used in TAPESTREA . *Digital Audio Signal Processing* by Zölzer [37] has a chapter on implementing a multi-complementary filter bank which will be useful if the modifications on the deterministic analysis algorithm suggested by Levine [14] are implemented. *Numerical Recipes in C, Volume 2* by Press, Teukolsky, Vetterling, and Flannery [22] provides an implementation of the discrete wavelet transform using DAUB4 filter coefficients, developed by Daubechies, which is used in the current stochastic analysis/synthesis algorithm in TAPESTREA.

6.2 Proposed Timeline

Development Timeline and Objectives:

- Week 1 (February 6th - 12th): Implement baseline sinusoid analysis and modelling in VST/AU framework.
- Week 2 (February 13th - 19th): Transient detection and extraction. Begin stochastic modelling using wavelet learning tree.
- Week 3 (February 20th - 26th): Wavelet learning tree.
- Week 4 (February 27th - March 5th): UI. Analysis Parameters.
- Week 5 (March 6th - 12th): Continue UI. Submit progress report, re-evaluate timeline.
- Week 6 (March 13th - 19th): Continue UI. Look into improvements suggested by Lieber [15].
- Week 7 (March 20th - 26th): Percussive / Harmonic Model for Analysis [4]
- Week 8 (March 27th - April 2nd): Polyphonic deterministic analysis using multi-resolution techniques [14]
- Week 9 (April 3rd - 9th): Continue improvements. Look at stochastic model.
- Week 10 (April 10 - 16th): Evaluation and Final Report.

7. PROGRESS REPORT

Progress made up to the sixth week of development is reported here and the development timeline and objectives are reviewed. At this stage the scope of the project has been narrowed down to a sinusoidal plus simplified stochastic modelling synthesizer, with implementations for increased playability as a plugin synth. The synthesizer being implemented has been named Loom [6] because of the heavy influence that is being drawn from the musical tapestry paradigm that TAPESTREA introduced.

7.1 Development Progress

Initial development has been completed using the JUCE audio framework and a functioning software synthesizer with both VST and AU builds work in the testing environment (JUCE Plugin Host) as well as one tested major DAW software: Ableton Live. The plugin receives MIDI from the host application and outputs audio on a stereo channel.

So far, only the sinusoidal modelling analysis/synthesis algorithms have been implemented along with basic UI functionality. The UI currently has one button on it that prompts a user to select an input audio file to be the source for sinusoidal analysis. Once the user has selected an audio file, a sinusoidal model is constructed using an algorithm proposed by Serra and Smith [30]. Once the model has been built, it is automatically loaded into the synthesis side of the application and is rendered in response to MIDI input. Further detail on the implemented sinusoidal analysis and synthesis algorithms are provided. Minimal code is listed, for a full listing of up-to-date code developed for this project, see the GitHub repository [7] .

A new resource that has been exceptionally helpful during development is a free online course on audio signal processing techniques provided by Xavier Serra and Julios O. Smith through Coursera [8] [36].

7.1.1 Sinusoidal Analysis

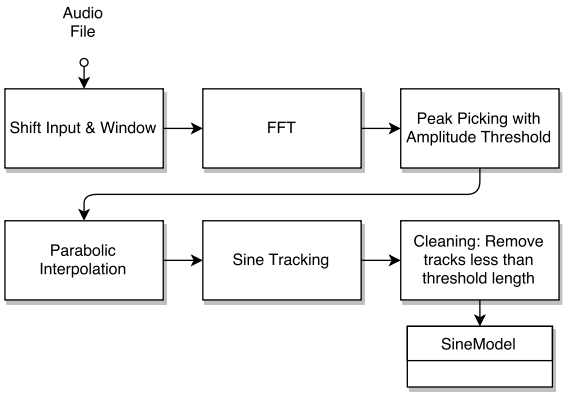


Figure 1: Analysis Phase Signal Flow Diagram

The entry point to the sinusoidal analysis algorithm is an audio file that has been specified by the user. Cur-

[6] See section 9.0.1
[7] https://github.com/jorshi/loom
[8] https://www.coursera.org/

rently only WAV files are supported, but this can easily be extended to include a larger variety of files using built-in functionality provided by JUCE. The audio file is analyzed using shifted frames. A frame size of 2048 samples and a hop size (shift amount) of 128 samples is hard-coded into the system right now, but these could be parameterized and exposed to the user interface. Each frame is windowed using a normalized and zero-phased Hamming window. Code from the Marsyas [34] audio framework is implemented for reading audio from a file, cutting the signal into frames, shifting the signal by the hop-size number of samples, and for calculating the windowing function. Each windowed frame is then run through a Fast Fourier Transform (FFT) to produce the frequency magnitude and phase spectrums for that frame.

Peak picking is applied to the magnitude spectrum at each frame. A peak is defined as a magnitude bin where the preceding and successive bins have a smaller magnitude. Each peak must meet an amplitude threshold requirement in order to be further considered for the model. This threshold is currently set at -80dB and is another value that can be parameterized. All peaks that meet this threshold requirement are then passed on to the parabolic interpolation algorithm.

Parabolic interpolation is used to overcome the fact that the FFT has a limited frequency resolution and the selected peak most likely is a frequency component that falls between magnitude bins. Parabolic interpolation can dramatically increase the accuracy of the estimated frequency. Once the peak location has been determined, approximations for the amplitude and phase are calculated. The frequency, amplitude, and phase for each peak is stored in a SineElement object. A two dimensional vector of SineElement objects represents each of the interpolated peaks for each frame analyzed. This collection of SineElements is stored in a SineModel object which also contains information about the analysis hop-size, frame-size, and sample rate used during the analysis phase. To refine this model, it passes through two more stages: sine-tracking and cleaning. See Figure 2 for code snippet of parabolic interpolation.

Sine-tracking is a process of connecting sinusoids in subsequent frames that have frequency deviations within a certain amount. This extends the model from the previous stage to a set of discrete time-varying sinusoids lasting over a number of frames. A track is represented by an integer that is stored in each SineElement and each SineElement can belong to only one track. The frequency threshold is determined by an offset and slope, which allows for scaling across the frequency spectrum. Once each SineElement in the SineModel object has been marked as belonging to a track then the model is passed into the cleaning algorithm.

The cleaning algorithm simply looks at each calculated track and removes tracks that are less than a particular threshold length. The duration threshold is another parameter that can be exposed on the UI. This cleaned SineModel is then ready to be used by the synthesis algorithm.

7.1.2 Sinusoidal Synthesis

The goal of the sinusoidal synthesis phase is to reconstruct a time domain signal of a sinusoidal model. The synthesis algorithm receives a SineModel object and synthesizes a corresponding signal in real-time in response to a MIDI event. The synthesis model is implemented in a JUCE Synthesiser class, which holds SynthesiserSound and SynthesiserVoice objects. This collection of classes implements many basic functions such as receiving MIDI events and distributing voices in response to polyphonic events. The sinusoidal synthesis model discussed here implements a FFT based technique and the corresponding algorithms are written into classes that inherit from JUCE's classes. The classes developed are SinusoidalSynthSound, which is a subclass of SynthesiserSound, and a SinusoidalSynthVoice, which is a subclass of SynthesiserVoice.

When a MIDI event is received and passed to the Synthesiser object, a SinusoidalSynthSound object is selected and assigned to a free SinusoidalSynthVoice. At the beginning of a note the first frame from the SineModel is requested and synthesized. Subsequent frames are synthesized until there are no more frames or a note-off MIDI event is received. Each SineElement is modelled in the frequency domain using a shifted Blackman Harris window. Once the spectrum for an entire frame has been modelled that spectrum is run through an inverse-FFT function to get a time-domain representation of the signal. The resulting time-domain signal is then windowed by a window that has been created to undo the effects of the Blackman Harris windowing in the frequency domain. A triangular window is then applied over half of the frame, centered in the middle of the frame, allowing for a Constant-Overlap-Add (COLA) of 25% [31].

7.2 Challenges

Initial development plans projected having baseline implementations for sinusoidal, transient, and stochastic modelling completed by this point in the project. Development of the sinusoidal model inside of a real-time audio plugin was more challenging than expected and a great deal of time was spent tuning the algorithm to achieve acceptable reproduction.

The baseline sinusoidal modelling algorithm was implemented relatively quickly in the first week, however audio production was very distorted. An initial major contributor to this issue was related to the synthesis algorithm and the Blackman Harris window that was being used to generate sinusoids in the frequency domain. The window was created by downsampling a shifted 1000 sample long Blackman Harris window down to 9 samples and then using that to model a sine wave in the frequency domain. A major oversight in this approach is that the creation of a Blackman Harris window depends on the size of the window. So sampling a larger window isn't an accurate way to represent the same window in less samples. This error was fixed using a window sampling method implemented in the sms-

```
/* Peak interpolation using Parbolic Interpolation */

// Find the interpolated location, in terms of bins
// The magnitude of the peak bin is the last bin: pMag
// Preceding bin to pMag: ppMag
// Successive bin to pMag: mag
mrs_real ipLoc = (i-1) + (0.5*(ppMag-mag)/(ppMag - 2*pMag + mag));

// Convert that location to frequency
mrs_real ipFreq = ipLoc * sampleRate / frameSize;

// Interpolate amplitude
mrs_real ipAmp = pMag - 0.25*(ppMag-mag)*(ipLoc-(i-1));

// Phase Interpolation
mrs_natural closestBin = std::round(ipLoc);
mrs_real factor = ipLoc - closestBin;
mrs_natural ipPhaseBin = (factor < 0) ? closestBin - 1 : closestBin + 1;
mrs_real ipPhase;

// Only interpolate if there is not a phase jump between bins
if (std::abs(phases(ipPhaseBin) - phases(closestBin)) < PI)
{
    ipPhase = factor * phases(ipPhaseBin) + (1-factor)*phases(closestBin);
}
else
{
    ipPhase = phases(closestBin);
}

// Save all the detected peaks
frameElements.emplace_back(ipFreq, ipAmp, ipPhase);
```

Figure 2: Parabolic Interpolation in Analysis Phase

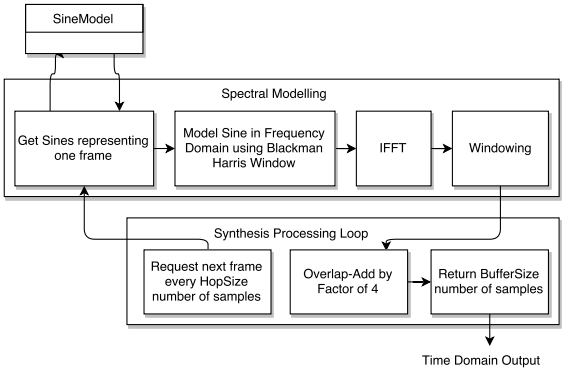


Figure 3: Synthesis Phase Signal Flow Diagram

tools package [9]; this method has a pre-calculated window containing 100 shifted Blackman Harris windows of length 9 that are intended to model the main lobe of a 512 sample window in the frequency domain. The shifted windows allow for approximation of sinusoids falling in between frequency bins.

Another challenge that was encountered was that the output amplitudes were much louder than the input audio signals used for analysis. This was causing the output signal to clip and distort at times. The source of this issue was tracked to the normalization of the windows used during analysis and synthesis. Once both windows were normalized using the same method, the amplitudes matched and the output signal was not distorting.

Once the windowing issues had been detected and fixed, the output signal was accurate for simple test sounds such as sinusoids, but more complex sounds were still distorted.

[9] https://github.com/MTG/sms-tools

```
mrs_realvec window(frameSize);
SynthUtils::windowingFillHamming(window);
window /= window.sum();
```

Figure 4: Analysis Window Normalization

The problem ended up being in how phases were being calculated in the analysis stage. The first method that was implemented simply did linear interpolation between the two bins straddling the detected peak. A different phase interpolation algorithm was used that only interpolated phases between bins if the difference in phase between those two bins was less than π radians. This algorithm is the same that is implemented in the Essentia [10] sinusoidal analysis algorithm.

7.3 Revised Objectives

In light of the challenges encountered, a reduced scope for the remaining development tasks is proposed. Instead of implementing all three of the models used in TAPESTREA, focus will be shifted to enhancing the playability of the implemented sinusoidal analysis and synthesis algorithm, as well as on implementing a simple stochastic modelling system. Creation of a user interface (UI) will also be a large component of this development. Three scenarios are provided with objectives:

7.3.1 Scenario 1: Expected Progress

This scenario represents the baseline progress that is expected to be completed. It is the most probable develop-

[10] http://essentia.upf.edu/

ment outcome.

- Parameterization of analysis variables.
- Ability to drag and drop an audio file into the analysis section.
- Parameterized amplitude envelope on synthesis output.
- Ability to specify the number of allowable voices.
- Synthesis pitch modifications quantized to MIDI notes.
- Time-scale modifications independent of pitch.
- Simplified stochastic model using FFT-based filtering of white noise
- Parameterization of pitch and time-scale modifications.

7.3.2 Scenario 2: Exceptional Progress

This scenario represents the case that development goes very well and time remains for further development. Some subset of these objectives will be completed.

- All objectives from expected progress are completed.
- Save the plugin state as a preset and recall presets.
- Save and load sineModels as XML.
- Ability to layer multiple sinusoidal models and synthesize simultaneously.
- Independent control over parameters and amplitude envelope of layered sinusoid models.
- Loop playback sections of a sine model.
- Freeze playback of frames from a sine model.
- Show Sinusoid model as spectrogram representation on UI.
- Ability to select a subsection of an audio file for analysis.

7.3.3 Scenario 3: Minimal Progress

This scenario represents the case that development goes poorly over the next several weeks. This is the absolute baseline progress that is hoped to be accomplished.

- Parameterization of analysis variables.
- Ability to drag and drop an audio file into the analysis section.
- Parameterized amplitude envelope on synthesis output.
- Ability to specify the number of allowable voices.

7.4 Revised Timeline

Revised development timeline, based on the expected progress objectives.

- Week 6 (March 13th - 19th): UI analysis parameterization, ability to drag and drop an audio file into the analysis section.
- Week 7 (March 20th - 26th): Pitch and time-scale modifications. Pitch quantized to MIDI notes.
- Week 8 (March 27th - April 2nd): Simplified stochastic model. Amplitude envelope.
- Week 9 (April 3rd - 9th): Ability to specify number of allowable voices, parameterization of time/pitch modifications and amplitude envelope.
- Week 10 (April 10 - 16th): Final development, evaluation, and final report.

8. FINAL REPORT

At the time of writing this final report section, the project is now in the 10th week of the development timeline. Development completed up to this point will be discussed in relation to the goals outlined in the progress report section (7.3).

8.1 Development Completed

After the initial challenges encountered during the early stages of development, the revised set of objectives in section 7.3 provided a much more realistic trajectory for the last several weeks of work. All of the minimal progress and expected progress objectives were accomplished, except for the parameterization of the number of allowable voices. This feature was left out as it was deemed a non-crucial feature and a decision was made to re-direct efforts towards implementing some of the stretch goals outlined in the exceptional progress objectives. Of the exceptional progress objectives, the model layering and spectrogram visualizer features were implemented. Development phases (not in chronological order of implementation) can be loosely grouped into stochastic modelling, pitch quantization, synthesis modulation, parameterization, layering, and user interface. Features related to each of these groups will be outlined.

8.1.1 Stochastic Modelling and Synthesis

Given the amount of time for development, a greatly simplified stochastic modelling algorithm was implemented. The implemented stochastic analysis algorithm follows the sinusoidal modelling algorithm in the analysis chain. See figure 5 for an updated signal flow diagram including stochastic modelling. Using the sinusoidal model a residual signal can be obtained by subtracting a re-synthesized version of that model from the frequency domain version of the original signal and then performing an inverse FFT

and overlap-add operation. This residual signal is transformed back into the frequency domain, then the magnitude spectrum is calculated and stored in a StochasticModel object, which is a 2-D array representing the time evolution of the magnitude spectrum of the residual signal. Each magnitude spectrum is represented with 128 values currently. Future work could perform an approximation of this magnitude spectrum [30]. The Stochastic-

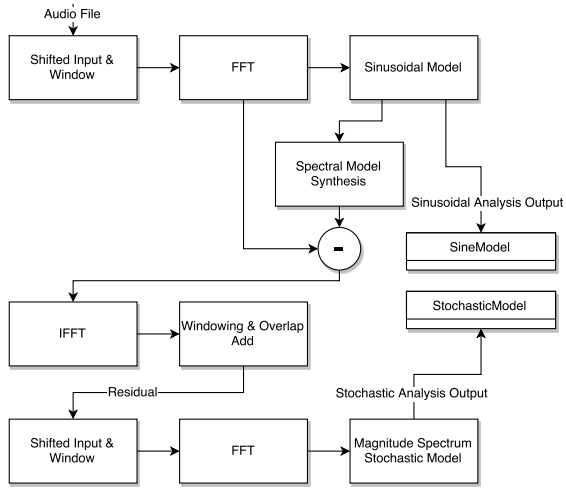


Figure 5: Analysis Phase with Stochastic Model Signal Flow Diagram

Model object is then passed into the synthesis side of the application along with a corresponding SineModel for resynthesis. During synthesis, each frame is resampled using linear interpolation and the phase spectrum is randomized to approximate white noise. This spectrum is added to the frequency domain signal calculated for the sinusoidal model at that frame before converting to a time domain signal.

8.1.2 Pitch Quantization

Pitch quantization is a frequency scaling of the sinusoidal model in a way that corresponds to notes of the equal temperament tuning system. This allows for musical passages to be performed using a MIDI keyboard, or to be programmed using MIDI in host applications. A value for frequency scaling is calculated as a ratio relative to the pitch A4, which is the tuning key for most western music.

$$frequency\_scale = 2^{\frac{note-A4}{12}} \quad (1)$$

8.1.3 Synthesis Modulation: Frequency

The following two sections outline the frequency and time modulations that can be applied to each model during resynthesis. In order to facilitate these modifications the calculation of sinusoidal phase had to be modified. When frequency and/or time-scale modifications are applied the phase for subsequent sinusoids in a track will be incorrect. Only the analysis phase is used when a new sine track begins, and then all sinusoids in subsequent frames for that track are re-calculated based on the previous phase and frequency.

- *Tuning*
  Tuning modifications are multiplications based on discrete tuning ratios at various scales. The largest scale is the octave. Octave tuning is achieved by multiplying or dividing a frequency by a multiple of two. Semitone tuning splits an octave into 12 equal parts. Cent tuning splits a semitone into 100 equals parts.

- *Frequency Scaling*
  Frequency scaling is similar to tuning, except a continuous scaling value is used as a multiplier to achieve pitch bends and other creative effects. A potential future feature could include the ability to attach a low frequency modulation oscillator to this in order to get vibrato type effects.

- *Frequency Shifting*
  Frequency shifting is the addition of a fixed frequency in hertz to each sinusoid frequency. This gives an effect similar to ring modulation.

- *Spectral Morphing*
  Spectral morphing is an interesting frequency domain effect that uses the ratio between a frequency and a reference to exponentially scale that frequency. The implemented algorithm is a modified version of a similar effect used in sms-tools [11] . See algorithm 1 for spectral morphing pseudocode.

Algorithm 1 Spectral Morphing

$freq \leftarrow$ frequency of current sinusoida
$center \leftarrow$ center frequency for morphing
$amount \leftarrow$ amount of morphing, 0 to 1
$morphRatio \leftarrow freq/center$
if $morphRatio > 1$ **then**
    $exp \leftarrow (morphRatio * amount) + 1$
else
    $exp \leftarrow 1 - (morphRatio * amount)$
$freq \leftarrow freq^{exp}$

8.1.4 Synthesis Modulation: Time

- *Playback Position*
  Playback position offsets the start position in time of the model. This is represented as a percentage of the total number of frames in the model allowing for synthesis to begin at any frame.

- *Playback Rate*
  Playback rate is a ratio of the original speed of the model. Values for the sampling rate and hop size used during analysis are stored in the model. This allows for playback at the original speed during resynthesis using different sampling rates and hop size choices. Playback rate can go all the way down to zero which will cause one frame to be continually synthesized.

[11] https://github.com/MTG/sms-tools

8.1.5 Synthesis Modulation: Amplitude

The amplitudes of the sinusoidal and stochastic models can be modified independently. In addition to this an envelope is applied to both of these amplitudes that is triggered on every new MIDI note-on event. The envelope is a traditional attack - decay - sustain - release (ADSR) envelope.

8.1.6 Synthesis Modulation: Sinusoidal Model

The number of sinusoids used during re-synthesis can be manipulated in real-time. This is a percentage of the total number of sinusoids at any given time in a frame. During the analysis stage the sinusoids for each frame are sorted by amplitude, so the quietest sinusoids are removed first when this parameter is reduced.

8.1.7 Layering

One of the objectives that was achieved from the exceptional progress section is the ability to layer multiple sinusoidal and stochastic models for simultaneous playback. On the synthesis side this feature is relatively simple. During the main synthesis processing block an array stores the active models that are to be layered. The synthesized models are then summed in the frequency domain. This approach is desirable because it means that no additional FFTs are required for layering, only the computational cost of calculating additional spectrums is incurred.

The challenging part of layering was setting up the parameters and user interface to allow for the dynamic addition of new layers and making sure that parameters for each layer were connected correctly. A known issue that is currently being addressed resulted in parameters for different layers getting mixed up during synthesis resulting in unexpected modifications to other layers.

8.1.8 Parameterization

JUCE has a built-in parameterization system that handles the creation of parameters that can be easily attached to UI components such as buttons and sliders, as well as accessed in processing blocks for modifying DSP algorithms. In addition to this, once a parameter has been added using this system, JUCE will automatically register the parameter with a host application allowing for automation of these parameters in a DAW. The JUCE AudioProcessorValueTreeState class is the interface that is used to access the parameterization system.

As mentioned in the previous section, handling parameters in the context of layering provided some challenges. Each layer represents a sinusoidal and stochastic model. That layer could be in one of three states: (1) the model is empty (no audio file has been loaded in), (2) a sound file has been loaded and is in the analysis stage, or (3) the model is being synthesized. The empty state has no parameters associated with it, but the analysis and synthesis state have unique sets of parameters. Also, each layer must have its own set of these unique parameters for independent control of that layer. Parameters from the AudioProcessorValueTreeState are named and accessed using that name, so each parameter stored must have a unique name. In order to avoid using these unique string names throughout the code and to keep the algorithms generic, interfacing classes were created in order to manage the layers and their parameters. Figure 6 shows the class diagram for this system.

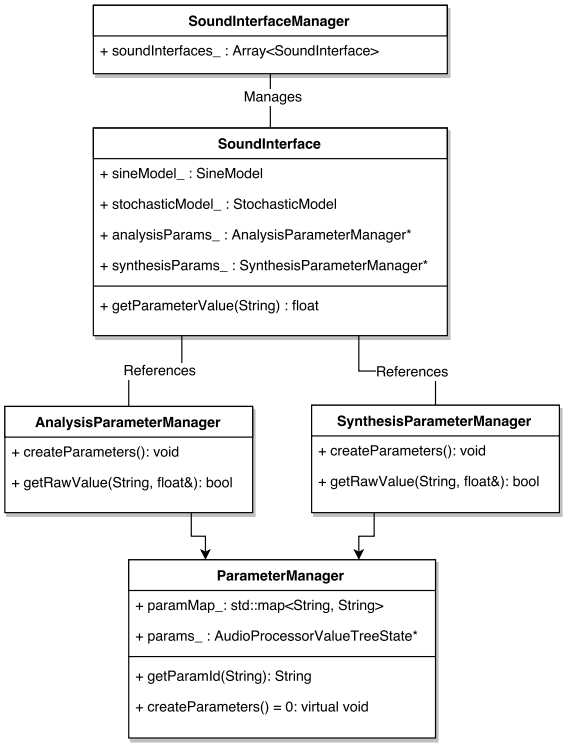
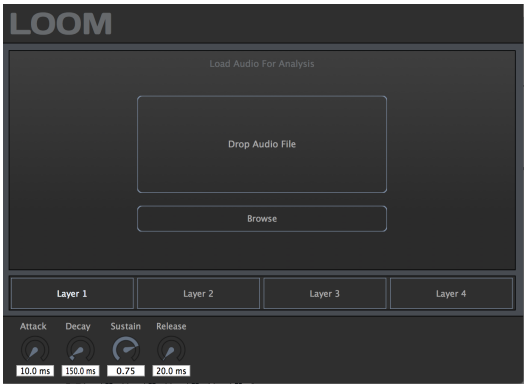


Figure 6: Interfaces for accessing models and parameters for a layer

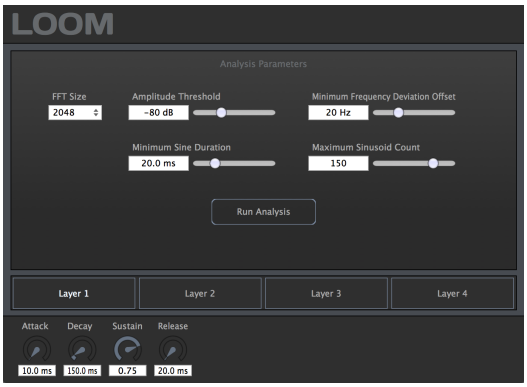
The SoundInterface class holds all the analysis classes, models, and parameters related to one layer. Each SoundInterface holds a reference to unique instances of subclassed ParameterManager objects which allows for generic parameter names to be mapped to the unique parameters stored in the AudioProcessorValueTreeState. Each subclassed ParameterManager is also responsible for the construction of a set of parameters related to its functionality; the AnalysisParameterManager creates a set of parameters related to the analysis state for a particular layer and the SynthesisParameterManager does the same but for parameters related to synthesis. The SoundInterfaceManager stores an array of SoundInterfaces which allows for an arbitrary number of layers to be created; currently a maximum of 4 is set, but the infrastructure is in place to support more layers. The SoundInterfaceManager is a single object that can be passed to the synthesis and user interface components of the application. Models and parameters can then be accessed in a way that is more general and allows for layering to occur dynamically.

8.1.9 User Interface

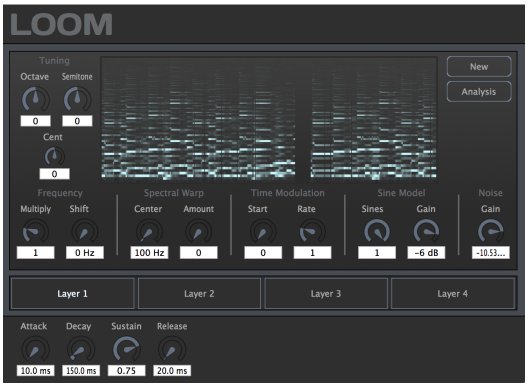
A user interface was designed and implemented to allow user modification of the various parameters related to each layer in various states. Layers can be switched between and a spectrogram visualizer is implemented for each layer.



(a) Load File State



(b) Analysis State



(c) Synthesis State

Figure 7: User Interface States

A section for envelopes is at the bottom of the UI; currently only one envelope has been created and parameterized, however room is left on the UI with the intention of allowing for more envelope and modulator parameters to be visualized here. Design of the UI was done using Sketch [12] . See figure 7 for screen captures of the UI.

The state that a particular layer is currently in is reflected on the UI for that layer. Upon opening the application, the load state is shown for the first layer prompting a user to load an audio file. An audio file can either be dropped onto the UI or a file browser can be opened to allow the user to find a file. A quick test to make sure that a readable audio file has been loaded is performed and then the analysis state is entered. The analysis state occurs before the analysis algorithms are run and allows the user to adjust parameters related to that. When the user presses the "Run Analysis" button the analysis algorithms are set in motion. Once they have completed, the resulting SineModel and StochasticModel objects are moved into a variable within the SoundInterface for that layer, which allows them to be accessed by the synthesis algorithm. This moves the layer into the synthesis state. Parameters related to synthesis are shown and a real-time spectrogram showing what is currently being synthesized by that layer is continually rendered.

[12] `https://www.sketchapp.com/`

9. FUTURE WORK

A working synthesizer plugin has been developed, inspired by the TAPESTREA system. The current implementation can serve as a foundation for further development in this paradigm of sound creation and development towards an application that could potentially be released publicly in some capacity is planned by the author. Refinement of the current algorithms and implementation of features based on the goals of the project and the related work that was referenced in sections 3, 4, and 5 all serve as directives for future work that can be done. Some specific and planned goals for future work are outlined.

9.0.1 Name Change

The current name Loom will need to be changed. An oversight in the initial naming was made as a commercial software synthesizer named Loom [13] already exists. Air Music Technology released this additive synthesizer in 2013. Replacement names have been considered but nothing has stuck so far.

9.0.2 Progress Report: Exceptional Progress Objectives

The objectives outlined in the the exceptional progress report objectives (section 7.3.2) are all relevant for future work. Particularly important features would be the ability to save the plugin state, save and recall presents, and save

[13] `http://www.airmusictech.com/product/loom`

and recall previously created models. These would greatly improve upon the functionality of the plugin in a modern music production work flow allowing the user to save and recall previous work.

9.0.3 Initial Objectives

The initial project objectives outlined in section 5.1 are relevant for future work. Implementing the wavelet based stochastic modelling method used in TAPESTREA would be a good way to improve upon the currently implemented algorithm. The suggested improvements on the TAPESTREA methods can also be explored.

9.0.4 Improved Sinusoidal Model with Source Separation

Work performed by Lagrange et al. on improved sinusoidal tracking as well as subsequent work related to this on source separation could be interesting for future work [11, 12]. The work done on source separation, if implemented could allow for extraction and re-synthesis of certain musical source components within polyphonic material. The original TAPESTREA application had implemented a similar feature using a different method for grouping and separating sinusoidal tracks, although it was indicated that the algorithm would benefit from a more sophisticated approach.

10. CONCLUSION

A model for composing environmental sounds was developed by Misra et al. called TAPESTREA. This model was unique in that it used a revised version of the sines+transient+noise model for enabling musicians and sound designers to explore and re-compose sounds. An overview of the analysis and synthesis phases of TAPESTREA in conjunction with a review of relevant research has identified some areas for potential improvement. Research related to the harmonic+percussive model initially developed by Ono et al. [19] may be useful for the analysis phase of the application. A more advanced sinusoidal modelling algorithm [14] may also be a beneficial addition. A great deal of research has also been conducted in the area of sound texture synthesis in recent years which could be useful for the stochastic modelling portion of the application.

In light of the unique compositional opportunities presented by the TAPESTREA paradigm, recent developments in related research fields, and because of the now legacy status of the original application, a new project design has been proposed and a version of it developed. After challenges in the initial phase of development a reduced-scope application was proposed. The main objectives of this revised project have successfully been completed resulting in a functioning sinusoidal plus stochastic modelling synthesizer plugin. This application provides a basis for future work that may include improving on the implemented algorithms and adding new features that will benefit use in a modern music production environment.

11. ACKNOWEDGEMENTS

Many thanks to George Tzanetakis for supervising this project.

12. REFERENCES

[1] Tapestrea: Analysis parameters. `http://taps.cs.princeton.edu/doc/analysis/params.html`.

[2] Gilberto Bernardes, Luis Aly, and Matthew EP Davies. Seed: Resynthesizing environmental sounds from examples. 2016.

[3] Mark Dolson. The phase vocoder: A tutorial. *Computer Music Journal*, 10(4):14–27, 1986.

[4] Jonathan Driedger, Meinard Müller, and Sascha Disch. Extending harmonic-percussive separation of audio signals. In *ISMIR*, pages 611–616, 2014.

[5] Shlomo Dubnov, Ziv Bar-Joseph, Ran El-Yaniv, Dani Lischinski, and Michael Werman. Synthesizing sound textures through wavelet tree learning. *IEEE Computer Graphics and Applications*, 22(4):38–48, 2002.

[6] Derry Fitzgerald. Harmonic/percussive separation using median filtering. 2010.

[7] NJ Fliege and U Zolzer. Multi-complementary filter bank. In *Acoustics, Speech, and Signal Processing, 1993. ICASSP-93., 1993 IEEE International Conference on*, volume 3, pages 193–196. IEEE, 1993.

[8] Richard Füg, Andreas Niedermeier, Jonathan Driedger, Sascha Disch, and Meinard Müller. Harmonic-percussive-residual sound separation using the structure tensor on spectrograms. In *Acoustics, Speech and Signal Processing (ICASSP), 2016 IEEE International Conference on*, pages 445–449. IEEE, 2016.

[9] Mark Kahrs and Karlheinz Brandenburg. Applications of digital signal processing to audio and acoustics. pages 343–416. Kluwer Academic Publishers, 1998.

[10] Anil Kokaram and Deirdre O'Regan. Wavelet based high resolution sound texture synthesis. In *Audio Engineering Society Conference: 31st International Conference: New Directions in High Resolution Audio*. Audio Engineering Society, 2007.

[11] Mathieu Lagrange, Sylvain Marchand, and Jean-Bernard Rault. Enhancing the tracking of partials for the sinusoidal modeling of polyphonic sounds. *IEEE Transactions on Audio, Speech, and Language Processing*, 15(5):1625–1634, 2007.

[12] Mathieu Lagrange, Luis Gustavo Martins, Jennifer Murdoch, and George Tzanetakis. Normalized cuts for predominant melodic source separation. *IEEE Transactions on Audio, Speech, and Language Processing*, 16(2):278–290, 2008.

[13] Scott N Levine and Julius O Smith III. A sines+ transients+ noise audio representation for data compression and time/pitch scale modifications. In *Audio Engineering Society Convention 105*. Audio Engineering Society, 1998.

[14] Scott N Levine, Tony S Verma, and Julius O Smith. Alias-free, multiresolution sinusoidal modeling for polyphonic, wideband audio. In *Applications of Signal Processing to Audio and Acoustics, 1997. 1997 IEEE ASSP Workshop on*, pages 4–pp. IEEE, 1997.

[15] Tom Lieber, Ananya Misra, and Perry R Cook. Freedom in tapestrea! voice-aware track manipulations. In *ICMC*, 2008.

[16] Robert McAulay and Thomas Quatieri. Speech analysis/synthesis based on a sinusoidal representation. *IEEE Transactions on Acoustics, Speech, and Signal Processing*, 34(4):744–754, 1986.

[17] Ananya Misra, Perry R Cook, and Ge Wang. A new paradigm for sound design. In *Proceedings of the International Conference on Digital Audio Effects (DAFx-06)*, pages 319–324. Citeseer, 2006.

[18] Ananya Misra, Ge Wang, and Perry Cook. Musical tapestry: Re-composing natural sounds. *Journal of New Music Research*, 36(4):241–250, 2007.

[19] Nobutaka Ono, Kenichi Miyamoto, Hirokazu Kameoka, and Shigeki Sagayama. A real-time equalizer of harmonic and percussive components in music signals. In *ISMIR*, pages 139–144, 2008.

[20] Deirdre O'Regan and Anil Kokaram. Multi-resolution sound texture synthesis using the dual-tree complex wavelet transform. In *Signal Processing Conference, 2007 15th European*, pages 350–354. IEEE, 2007.

[21] Seán O'Leary and Axel Röbel. A montage approach to sound texture synthesis. *IEEE/ACM Transactions on Audio, Speech, and Language Processing*, 24(6):1094–1105, 2016.

[22] William H Press, Saul A Teukolsky, William T Vetterling, and Brian P Flannery. Numerical recipes in c, second edition. pages 591–605. Cambridge Univ Press, 1982.

[23] Lance Putnam. Gamma: Generic synthesis library, 2006. `http://www.mat.ucsb.edu/gamma/`.

[24] Curtis Roads. *Microsound*. MIT press, 2004.

[25] Martin Robinson. Ugen++—an audio library: Teaching game audio design and programming. In *Audio Engineering Society Conference: 41st International Conference: Audio for Games*. Audio Engineering Society, 2011.

[26] Martin Robinson and Jamie Bullock. A comparison of audio frameworks for teaching, research, and development. In *Audio Engineering Society Convention 132*. Audio Engineering Society, 2012.

[27] Diemo Schwarz. Concatenative sound synthesis: The early years. *Journal of New Music Research*, 35(1):3–22, 2006.

[28] Diemo Schwarz. State of the art in sound texture synthesis. In *Digital Audio Effects (DAFx)*, pages 1–1, 2011.

[29] Diemo Schwarz and Norbert Schnell. Descriptor-based sound texture sampling. In *Sound and Music Computing (SMC)*, pages 510–515, 2010.

[30] Xavier Serra and Julius Smith. Spectral modeling synthesis: A sound analysis/synthesis system based on a deterministic plus stochastic decomposition. *Computer Music Journal*, 14(4):12–24, 1990.

[31] Julius O. Smith. *Spectral Audio Signal Processing*. `http://ccrma.stanford.edu/~jos/sasp/`, accessed March 12, 2017. online book, 2011 edition.

[32] Julian Storer. Juce (jules' utility class extensions), 2012.

[33] Barry Truax. Genres and techniques of soundscape composition as developed at simon fraser university. *Organised sound*, 7(01):5–14, 2002.

[34] George Tzanetakis and Perry Cook. Marsyas: A framework for audio analysis. *Organised sound*, 4(3):169–175, 2000.

[35] Tony S Verma and Teresa HY Meng. An analysis/synthesis tool for transient signals that allows a flexible sines+ transients+ noise model for audio. In *Acoustics, Speech and Signal Processing, 1998. Proceedings of the 1998 IEEE International Conference on*, volume 6, pages 3573–3576. IEEE, 1998.

[36] Julius O. Smith Xavier Serra. Audio signal processing for music applications. Universitat Pompeu Fabra of Barcelona, Stanford University, Coursera `https://www.coursera.org/learn/audio-signal-processing/`, 2017.

[37] Udo Zölzer. Digital audio signal processing. pages 168–180. John Wiley & Sons, 2008.