Proceedings of the 17th Linux Audio Conference (LAC-19), CCRMA, Stanford University, USA, March 23–26, 2019

A JACK-BASED APPLICATION FOR SPECTRO-SPATIAL ADDITIVE SYNTHESIS

Henrik von Coler

Audio Communication Group
TU Berlin
voncoler@tu-berlin.de

ABSTRACT

This paper presents a real-time additive sound synthesis application with individual outputs for each partial and noise component. The synthesizer is programmed in C++, relying on the Jack API for audio connectivity with an OSC interface for control input. These features allow the individual spatialization of the partials and noise, referred to as spectro-spatial synthesis, in connection with an OSC capable spatial rendering software. Additive synthesis is performed in the time domain, using previously extracted partial trajectories from instrument recordings. Noise is synthesized using bark band energy trajectories. The sinusoidal data set for the synthesis is generated from a custom violin sample library in advance. Spatialization is realized using established rendering software implementations on a dedicated server. Pure Data is used for processing control streams from an expressive musical interface and distributing it to synthesizer and renderer.

1. INTRODUCTION

1.1. Sinusoidal Modeling

Additive synthesis is among the oldest digital sound creation methods and has been the foundation of early experiments by Max Mathews at *Bell Labs*. It allows the generation of sounds rich in timbre, by superimposing single sinusoidal components, referred to as partials, either in the time- or frequency domain. Based on the Fourier Principle, any quasi-periodic signal $y(t)$ can be expressed as a sum of N_{part} sinusoids with varying amplitudes $a_n(t)$ and frequencies $\omega_n(t)$ and an individual phase offset φ_n:

$$y(t) = \sum_{n=1}^{N_{part}} a_n(t)\ sin(\omega_n(t)\ t + \varphi_n) \tag{1}$$

In harmonic cases, which applies to the majority of musical instrument sounds, the partial frequencies can be approximated as integer multiples of f_0:

$$y(t) = \sum_{n=1}^{N_{part}} a_n(t)\ sin(2\ \pi\ n\ f_0(t)\ t + \varphi_n) \tag{2}$$

Although relative phase fluctuations are important for the perception [1], the original phase can be ignored in many cases, which is of benefit for manipulations of the modeled sound:

$$y(t) = \sum_{n=1}^{N_{part}} a_n(t)\ sin(2\ \pi\ n\ f_0(t)\ t) \tag{3}$$

Based on this theory, an algorithm for speech synthesis has been proposed by McAulay *et al.* [2]. For musical sound synthesis the algorithm has been added a noise component [3], resulting in the *sinusoids+noise* model. The signal is then modeled as the sum of the deterministic part x_{det} and the stochastic part x_{stoch}, also referred to as residual:

$$x = x_{det} + x_{stoch} \tag{4}$$

Modeling of residuals can for example be performed by approximating the spectral envelope using linear predictive coding [3] or a filter bank based on Bark frequencies [4]. The phase of the stochastic signal is random, in theory, and thus needs not be modeled. However, residuals usually are not completely random since they still contain information from the removed harmonic content.

In order to fully model the sounds of arbitrary musical instruments, a transient component x_{trans} is included [4] in the full signal model. This component captures plucking sounds and other percussive elements:

$$x = x_{det} + x_{stoch} + x_{trans} \tag{5}$$

Since the work presented in this paper focuses on the violin in legato techniques, the transient component can be neglected without impairing the perceived quality of a re-synthesis.

1.2. Spectral Spatialization

In electronic and electroacoustic music, the term *spectral spatialization* refers to the individual treatment of a sound's frequency components for a distribution on sound reproduction systems [5]. Timbral sound qualities can thusly be linked to the spatial image of the sound, even for pre-existing or fixed sound material. In the case of *spectro-spatial synthesis*, this process is integrated on the synthesis level,for example in additive approaches. This is not yet a common feature in available synthesizers, but several research projects have been investigating the possibilities of such approaches with applications in musical sound processing, sound design, virtual acoustics and psychoacoustics.

Topper *et al.* [6] apply additive synthesis of basic waveforms (square wave, sawtooth), physical modeling and sub-band decomposition in a multichannel panning system with real time, prerecorded and graphic control. Their system is implemented in MAX/MSP and RTcmix, running on both Mac and PC/Linux hardware with a total of 8 audio channels.

Verron *et al.* [7] use the *sinusoids + noise* model for spectral spatialization of environmental sounds. Each component can be synthesized with individual position in space on Ambisonics and Binaural systems. Deterministic and stochastic components are composed and added together in the frequency domain and subsequently spatially encoded with a filterbank. Control over the synthesis process is depending on the nature of the environmental sounds [8].

In the context of electroacoustic music, James [9] expands Dennis Smalley's concept of *spectromorphology* to the idea of *spatiomorphology*. *Timbre Spatialization* is achieved using terrain surfaces

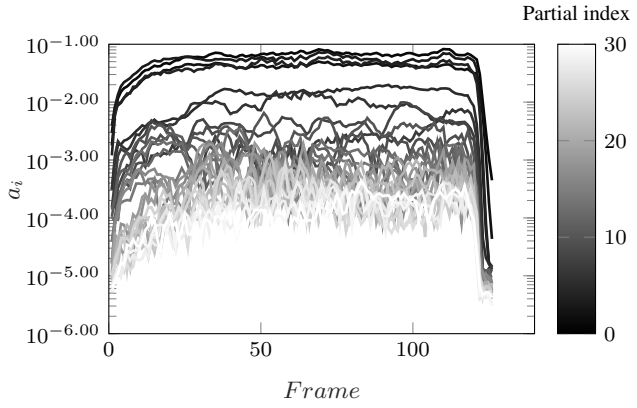

Figure 1: Partial amplitude trajectories of a violin sound

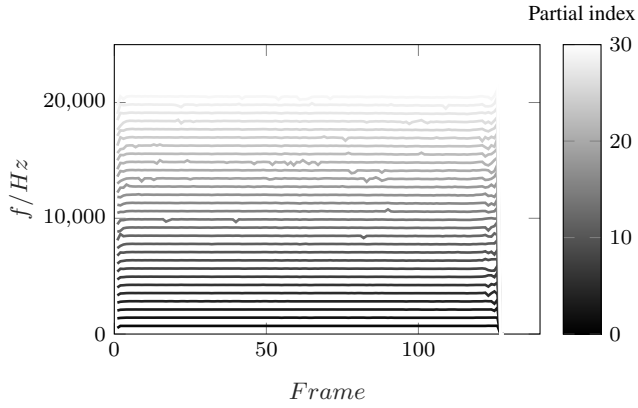

Figure 2: Partial frequency trajectories of a violin sound

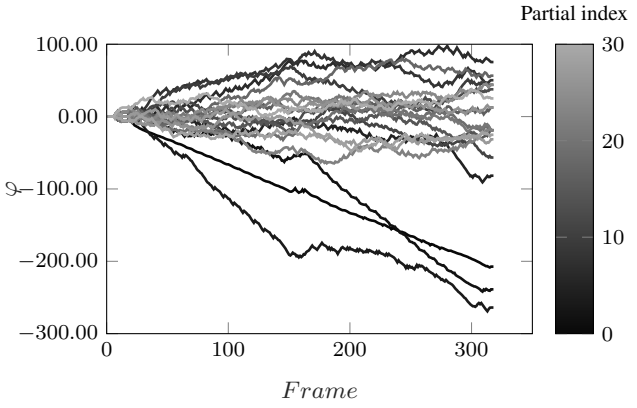

Figure 3: Unwrapped partial phases of a violin sound

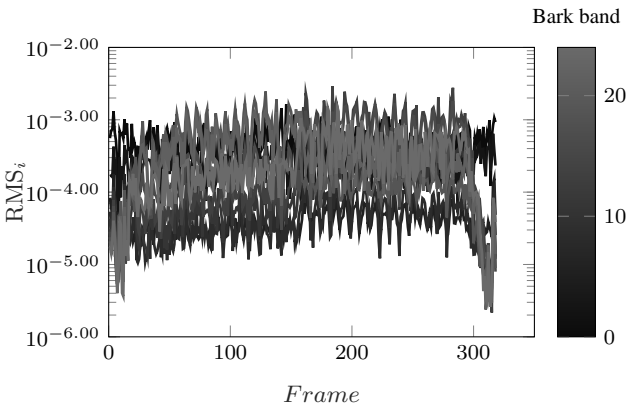

Figure 4: Bark band energy trajectories of a violin sound

and by mapping these to *spacio-spectral* distributions. Max-MSP is used for computing the contribution of spectral content to individual speakers with Distance-based amplitude panning (DBAP) and Ambisonic Equivalent panning (AEP) methods.

Spectral spatialization can also be used to synthesize dynamic directivity patterns of musical instruments in virtual acoustic environments. Since the directivity in combination with movement has a significant influence on an instrument's sound, this can increase the plausibility. Warusfel *et al.* [10] use a tower with three cubes, each containing multiple speakers, to spatialize frequency bands of an input signal for the simulation of radiation patterns.

1.3. The Presented Application

The presented application incorporates different synthesis modes, of which only the so called *deterministic mode* will be subject of this paper. In this basic mode, precalculated parameter trajectories, as presented in Sec. 2, are used for a manipulable resynthesis of the original instrument sounds.

The software architecture is designed to allow the use of additive synthesis, respectively of sinusoidal modeling, on sound field synthesis systems or other reproduction setups. This is achieved by providing individual outputs for all partials and noise bands in an application implemented as a JACK client, described in Sec. 3. Using JACK allows the connection of all individual synthesizer output channels to a JACK-capable renderer, such as the SoundScape Renderer (SSR) [11], Panoramix [12] or the HOA- Library [13]. Making each partial a single virtual sound source in combination with these rendering softwares, the spatial distribution of the synthesis can be modulated in real-time. Pure Data [14] is used to receive control data from gestural interfaces or to play back predefined trajectories for generating control streams for both the synthesizer and the spatialization renderer. A direct linkage between timbre and spatialization is thus created, which is considered essential for a meaningful *spectro-spatial synthesis*.

2. ANALYSIS

The *TU-Note Violin Sample Library* [15], [16], is used as audio content for generating the sinusoidal model. Designed in the style of classic sample libraries, this data set contains single sounds of a violin in different pitches and intensities, recorded at an audio sampling rate of 96 kHz with 24 Bit resolution.

Analysis and modeling is performed beforehand in Matlab, using monophonic pitch tracking and subsequent extraction of the partial trajectories by peak picking in the spectrogram. YIN [17] and SWIPE [18] are used as monophonic pitch tracking algorithms. Based on the f0-trajectories, partial tracking is performed with STFT, applying a hop-size of 256 samples (2.7ms) and a window size of

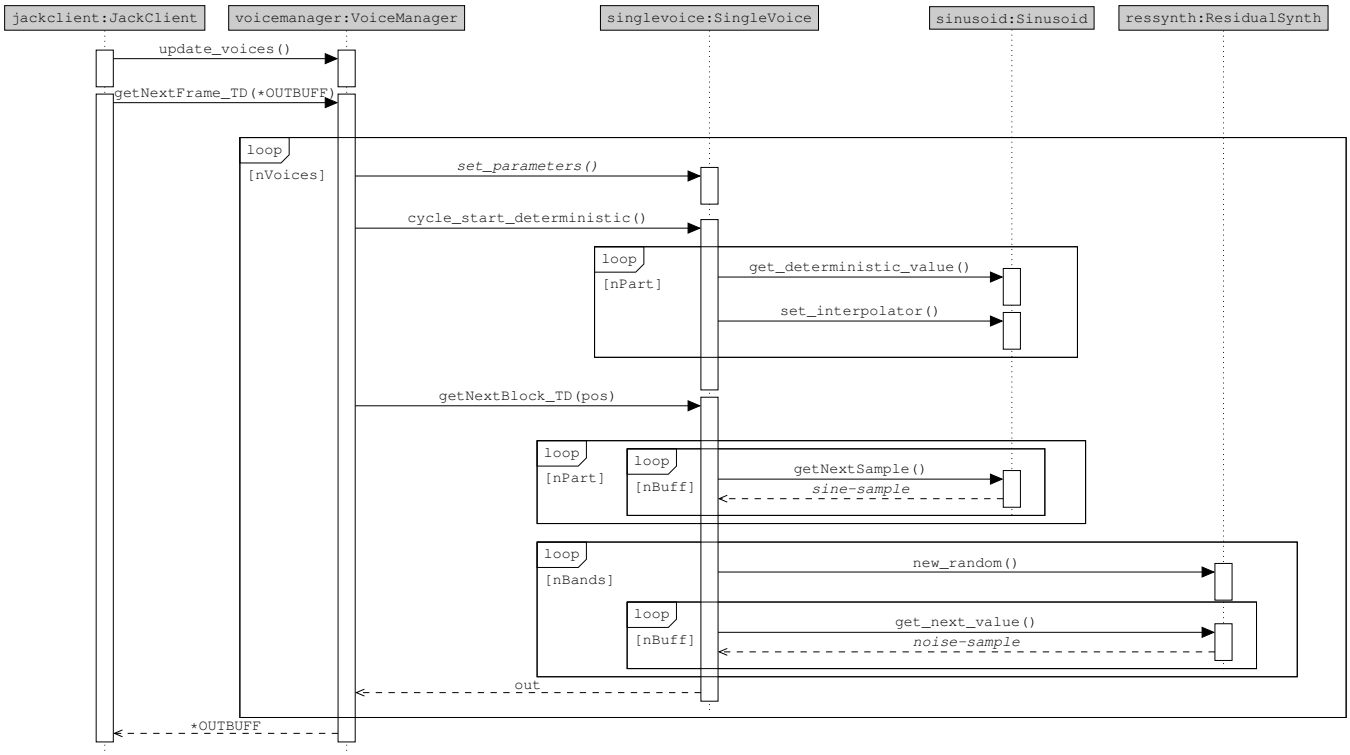

Figure 5: Sequence diagram for the jack callback function

4096 samples, zero-padded to 8192 samples. Quadratic interpolation (QIFFT), as presented by Smith *et al.* [19], is applied for peak parameter estimation of up to 80 partials. Due to the sampling frequency, the full number of partials is only analyzable up to the note D_5 (576.65 Hz)

By subtracting the deterministic part from the complete sound in the time domain, the residual signal is obtained. The residual is then filtered using a Bark scale filterbank with second order Chebyshev bandpasses and the temporal energy trajectories are calculated for the resulting 24 band-limited signals. At this point, a large amount of information is removed from the residual signal. Due to the shortcomings of the time domain subtraction method, the residual still contains information from the deterministic component. By averaging the energy over the Bark bands, this relation is eliminated.

Results of the synthesis stage are trajectories of the partial amplitudes, as shown in Figure 1, the trajectories of partial frequencies and phases, as shown in Figure 2, respectively Figure 3 as well as the trajectories of the Bark-band energies, illustrated in Figure 4. The resulting data is exported to individual YAML files for each sound, which can be read by the synthesis system.

3. SYNTHESIS SYSTEM

3.1. Libraries

The synthesis application is designed as a standalone Linux command line software. The main functionality of the synthesis system relies on the *JACK*[1] API for audio connectivity and the *liblo*[2], respectively the *liblo C++ wrapper* for receiving control signals. *libyaml-cpp*[3] is used for reading the data of the modeled sounds and the relevant configuration files. *libsndfile*[4] for reading the original sound files, as well as the *libfftw*[5] are included but not relevant for the aspects presented in this paper. Frequency domain synthesis and sample playback are partially implemented but not used at this point.

3.2. Algorithm

Both the sinusoidal and the noise component are synthesized in the time domain, using a non-overlapping method. For the sinusoidal component, the builtin `sin()` function of the `cmath` library and a custom lookup table can be selected. The choice does not affect the overall performance, significantly. The filter bank for the noise synthesis consists of 24 second order Chebyshev bandpass filters with fixed coefficients, calculated before runtime. The amplitude of each frequency band is driven by the previously analyzed energy trajectories.

During synthesis, the algorithm reads a new set of support points from the model data for each audio buffer and increments the position within the played note. Figure 5 shows a sequence diagram for the deterministic synthesis algorithm, starting at the JACK callback function, which is executed for each buffer of the JACK audio server. Since the synth is designed to enable polyphonic play, the voice manager object handles incoming OSC messages in the function `update_voices()` to activate or deactivate single voices.

[1] `http://jackaudio.org/`
[2] `https://github.com/radarsat1/liblo`
[3] `https://github.com/jbeder/yaml-cpp/`
[4] `http://www.mega-nerd.com/libsndfile/`
[5] `http://www.fftw.org/`

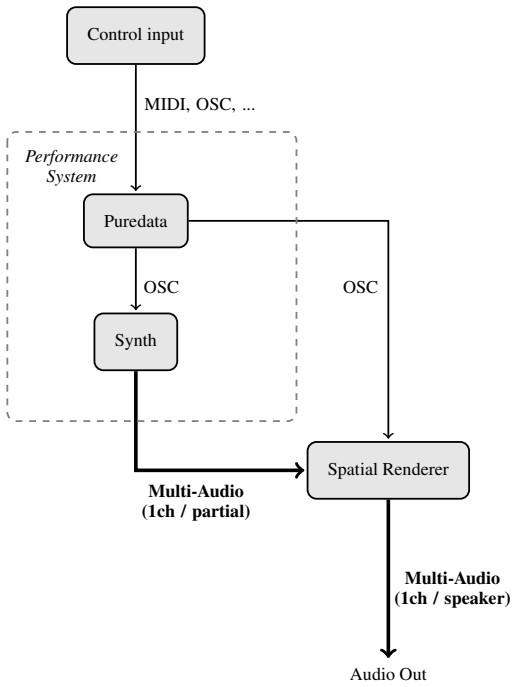

Figure 6: Combination of synthesizer and renderer on separate machines using Pure Data for synth configuration and parameter parsing

For the synthesis of mostly monophonic, excitation continuous instruments like the violin, the polyphony merely handles the overlapping of released notes. Subsequently, the voice manager loops over all active voices in the function `getNextFrame_TD()`, first setting the new control parameters for each voice.

In `cycle_start_deterministic()`, support points for all partial's parameters are picked at the relevant voice's playback position. These support points are then linearly interpolated over the buffer length in `set_interpolator()`.

Finally, in `getNextBlock_TD()`, each single voice generates the output for all sinusoids and all noise bands in two separate vectorizable loops, adding both to the output buffer.

3.3. Runtime Environment and Periphery

The runtime system for the synthesis is starting a JACK server with 48 kHz sampling rate, a buffer size of 128 samples and 2 periods per buffer. This results in $5.3\,\mathrm{ms}$ latency for the audio playback, which is within the limits for this synthesis approach. On an Intel(R) Core(TM) i7-5500U CPU @ 2.40GHz with disabled speed-stepping and a Fireface UFX, the JACK server is showing an average load of approximately $20\,\%$.

The interaction of the involved software components is visualized in Figure 6. For reasons of performance and increased flexibility in the studio, two separate machines are used for synthesis and spatialization. Connectivity between the systems is realized with MADI or DANTE, using individual channels for the 80 partials and 24 noise bands.

3.4. Control

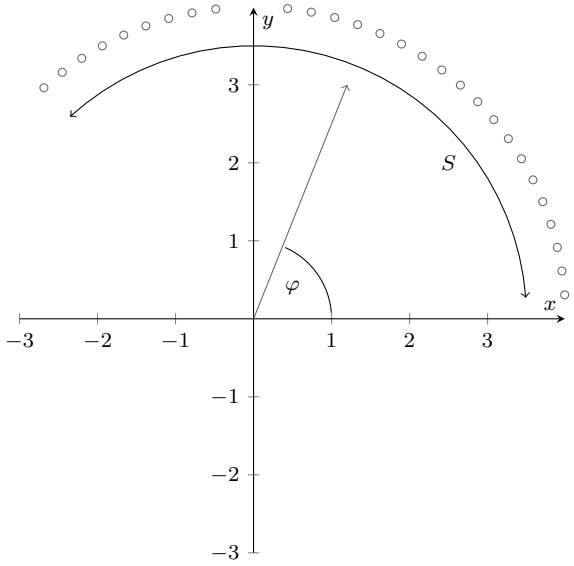

Figure 7: Spatialization scene in a 2D setup with 30 partials and their positions

The control data for the partial positions in the rendering software is not generated in the synthesis system at this point and is managed, externally. This offers more flexibility for testing different mappings at this stage of development. A Pure Data patch is used to receive incoming control messages, either from OSC or MIDI, and distribute them to the synthesizer and the spatialization software. For live performance, the patch receives continuous control streams for *pitch* and *intensity* from an improved version of the interface presented by von Coler *et al.* [20] and visualizes the sensor data. Pitch and intensity are forwarded to the synth, directly. Additionally, data from several Force Sensitive Resistors (FSR) and a 9 degrees of freedom IMU, which can be used for controlling the spatialization, is sent to the patch.

Figure 7 shows an example for a simple spatialization mapping on a 2D system. The absolute orientation of the IMU is used to control the general direction φ of the partial flock. A second parameter S, derived from the intensity and additional sensor data, controls the spread of the partials around this angle, depending on the partial index.

4. CONCLUSION

After significantly improving the performance of the synthesis system, the application can now be used with the full 80 partials and 24 Bark bands as individual outputs. Recent tests in combination with different spatial rendering softwares and different loudspeaker setups show promising results. However, the dynamic spatialization of such number of virtual sound sources and the resulting traffic of OSC messages is demanding for the runtime system. Using separate machines for synthesis and rendering reduces the individual load. The number of rendering inputs can also be reduced without limiting the perceived quality of the spatialization. Multiple partials may share one virtual sound source.

Next steps are now possible, which include the empirical investigation of mappings from controller sensors to both the spectral and spatial sound properties. This includes user experiments to evaluate different mapping and control paradigms, as well as perceptual measurements of the synthesis results.

5. ACKNOWLEDGMENTS

Thanks to Benjamin Wiemann for contributions to the project in it's early stage and to Robin Gareus for the help in restructuring the code and hencewith improving the performance.

6. REFERENCES

[1] T. H. Andersen and K. Jensen, "Importance and Representation of Phase in the Sinusoidal Model", *J. Audio Eng. Soc*, vol. 52, no. 11, pp. 1157–1169, 2004.

[2] R. McAulay and T. Quatieri, "Speech analysis/Synthesis based on a sinusoidal representation", *Acoustics, Speech and Signal Processing, IEEE Transactions on*, vol. 34, no. 4, pp. 744–754, 1986.

[3] X. Serra and J. Smith, "Spectral Modeling Synthesis: A Sound Analysis/Synthesis System Based on a Deterministic Plus Stochastic Decomposition ", *Computer Music Journal*, vol. 14, no. 4, pp. 12–14, 1990.

[4] S. N. Levine and J. O. Smith, "A Sines+Transients+Noise Audio Representation for Data Compression and Time/Pitch Scale Modi cations", in *Proceedings of the 105th Audio Engineering Society Convention*, San Francisco, CA, 1998.

[5] D. Kim-Boyle, "Spectral spatialization - an Overview", in *Proceedings of the International Computer Music Conference*, Belfast, UK, 2008.

[6] D. Topper, M. Burtner, and S. Serafin, "Spatio-operational spectral (sos) synthesis.", in *Proceedings of the International Computer Music Conference (ICMC)*, Singapore, 2003.

[7] C. Verron, M. Aramaki, R. Kronland-Martinet, and G. Pallone, "Spatialized additive synthesis of environmental sounds", in *Audio Engineering Society Convention 125*, Audio Engineering Society, 2008.

[8] C. Verron, G. Pallone, M. Aramaki, and R. Kronland-Martinet, "Controlling a spatialized environmental sound synthesizer", in *2009 IEEE Workshop on Applications of Signal Processing to Audio and Acoustics*, IEEE, 2009, pp. 321–324.

[9] S. James, "Spectromorphology and spatiomorphology of sound shapes: Audio-rate aep and dbap panning of spectra", in *Proceedings of the International Computer Music Conference 2015*, 2015.

[10] O. Warusfel and N. Misdariis, "Directivity synthesis with a 3d array of loudspeakers: Application for stage performance", in *Proceedings of the COST G-6 Conference on Digital Audio Effects (DAFX-01), Limerick, Ireland*, 2001, pp. 1–5.

[11] J. Ahrens, M. Geier, and S. Spors, "The SoundScape Renderer: A unified spatial audio reproduction framework for arbitrary rendering methods", in *Audio Engineering Society Convention 124*, Audio Engineering Society, 2008.

[12] T. Carpentier, "Panoramix: 3d mixing and post-production workstation", in *Proceedings of the International Computer Music Conference (ICMC)*, 2016.

[13] A. Sèdes, P. Guillot, and E. Paris, "The HOA library, review and prospects", in *International Computer Music Conference| Sound and Music Computing*, 2014, pp. 855–860.

[14] M. S. Puckette, "Pure Data", in *Proceedings of the International Computer Music Conference (ICMC)*, Thessaloniki, Greece, 1997.

[15] H. von Coler, J. Margraf, and P. Schuladen, *TU-Note Violin Sample Library*, TU-Berlin, 2018. DOI: 10.14279/depositonce-6747.

[16] H. von Coler, "TU-Note Violin Sample Library – A Database of Violin Sounds with Segmentation Ground Truth", in *Proceedings of the 21st Int. Conference on Digital Audio Effects (DAFx-18)*, Aveiro, Portugal, 2018.

[17] A. de Cheveigné and H. Kawahara, "YIN, a Fundamental Frequency Estimator for Speech and Music", *The Journal of the Acoustical Society of America*, vol. 111, no. 4, pp. 1917–1930, 2002.

[18] A. Camacho, "Swipe: A Sawtooth Waveform Inspired Pitch Estimator for Speech and Music", PhD thesis, Gainesville, FL, USA, 2007.

[19] J. O. Smith and X. Serra, "PARSHL: An Analysis/Synthesis Program for Non-Harmonic Sounds Based on a Sinusoidal Representation", Center for Computer Research in Music and Acoustics (CCRMA), Stanford University, Tech. Rep., 2005.

[20] H. von Coler, G. Treindl, H. Egermann, and S. Weinzierl, "Development and Evaluation of an Interface with Four-Finger Pitch Selection", in *Audio Engineering Society Convention 142*, Audio Engineering Society, 2017.