Digital Audio Effects – Assignment 2: Original Audio Effect

A Sample-based Karplus-Strong (String) Synthesis Algorithm

Max Graf - 190120296

Abstract

We present a real time synthesis algorithm based on the Karplus-Strong string synthesis method. Instead of a short burst of noise used to initially excite the string as specified in the original algorithm, we define a windowed section of a pre-existing audio recording as source material for excitation. This method offers a wide range of possible spectral combinations which allows for accurate control over the timbral characteristics of the synthesised sound. We evaluate and discuss the sonic qualities of the results achieved through variation of the source sample, window position and length.

Introduction

The original Karplus-Strong string synthesis algorithm [1] describes a method to emulate the sound of a plucked string and variations thereof without the use of any external sound source. The fundamental principle is to send a short burst of noise through a feedback loop and subsequently attenuate this signal in order to simulate the excitation and decay of a flexible string. Several extensions have been proposed to improve the realism of the generated sound, including fine-tuning of the strings, decay time alteration, modelling of dynamics and sympathetic string simulation [2].

Välimäki et al. presented an efficient approach for real time synthesis of different string timbres such as the banjo, mandolin and kantele [3]. Välimäki et al. and Smith et al. described approaches to physical modelling of string instruments using digital waveguides [4] [5]. An overview of the early approaches to improving the synthesised sound was given by Karjalainen, M., & Smith [6]. Further research saw the introduction of real time body morphing techniques [7] and the extension of the principles to other instrument groups, such as a hammered piano [8]. Cook and Scavone presented an implementation of a real time (string) synthesis library that allows for defining a custom waveform for the excitation [9].

While these techniques provide models that produce more realistic string sounds, they are predominantly concerned with manipulating the *development* of the generated sound. The source material used for the initial excitation is still assumed to be a burst of noise, or a filtered variation thereof.

The method proposed here allows for dynamic, real time variation of the sound *source* used for the excitation of the string. The motivation behind this research is mainly artistic in nature. The key idea is to provide means of exploration of the different timbres that can be synthesised from pre-existing audio data using the aforementioned techniques. The proposed method does not aim to provide a significant improvement of the realism of the generated sound, but rather contribute to the pool of techniques concerned with dynamically synthesising sound in real time.

Implementation

The system was implemented as a Virtual Studio Technology audio plugin in the C++ programming language and is based on the JUCE framework. It follows the design principles of software synthesisers, in that it offers several voices for playback as well as controls to manipulate the pitch and envelope of the produced sound. A high level description of the steps necessary to synthesise sounds is as follows:

1. Load a pre-existing audio sample into the plugin. This can be a vocal recording, a drum sound or a full-length song.
2. Define the window bounds (position in source sample and length) used to extract the excitation source.
3. Trigger a MIDI event.

Once an audio file is loaded and a window is defined, incoming MIDI note numbers are used to calculate fundamental frequencies and subsequently the required lengths of delay lines. First order lowpass filters with a cutoff frequency of half the system's sample rate are used to attenuate delayed samples in the feedback loop. This allows for a high degree of control over the strings' spectral contents by means of filter operations that are applied globally at later stages in the signal chain.

The system provides two modes of operation. In the default mode, strings follow the traditional process of excitation and decay. Here, decay times can be controlled by adjusting the amplitude of the signal passing through the feedback loop. Due to the use of a lowpass filter for attenuation, high notes naturally decay faster than low notes. To correct for this imbalance, an option for adaptive decay times was added. If enabled, this will analyse the incoming fundamental frequency to obtain a modified feedback value

$$s = f - \frac{ln(1000)}{f_0} \qquad \textit{Eq. 1}$$

where f is the current feedback value bounded by $[0 \dots 1]$ and f_0 is the fundamental frequency in Hz.

The second mode introduces an envelope used to modify the amplitude of the generated sound based on the conventional attack, decay, sustain and release (ADSR) paradigm. This allows for more nuanced control over the synthesised sound, and adds the possibility of extending the duration of a triggered note.

Due to the dynamic nature of the window's position in the main audio file, the resulting sounds vary greatly in terms of spectral content, yielding a multitude of possible timbral characteristics from a single source audio file.

System description

The transfer function describing the relationship between input and output is given by

$$H(z, N) = \frac{1}{1 - H_a(z) * H_d(z, N) * H_c(z) * H_m(z)} \qquad \textit{Eq. 2}$$

where

$$H_a(z) = \frac{F_c z^{-1} + F_c}{(F_c + 1)z^{-1} + (F_c - 1)} \qquad \textit{Eq. 3}$$

is the first order lowpass filter with filter coefficients based on F_c used for attenuation of the signal in the delay line,

$$H_d(z, N) = z^{-N} \qquad \textit{Eq. 4}$$

is the delay of N samples,

$$H_c(z) = \frac{C + z^{-1}}{1 + Cz^{-1}} \qquad \textit{Eq. 5}$$

is the allpass filter used to fine-tune the string with the tuning coefficient C and

$$H_m(z) = \frac{b_0 + b_1 z^{-1} + b_2 z^{-2}}{a_0 + a_1 z^{-1} + a_2 z^{-2}} \qquad \textit{Eq. 6}$$

is the biquadratic lowpass filter applied to the final output in order to allow for trimming high frequency content.

The original Karplus-Strong algorithm uses a two-point average to model the decay of the signal. However it was found that this categorically excludes some possible sounds from the synthesis process. In the current implementation, the filter coefficient F_c for H_a in *Eq. 3* is set to a constant value, so that the attenuation of high frequency content in the feedback loop is minimal. To counterbalance this, the filter H_m was introduced. Its coefficients are dynamically generated and can be configured freely to attenuate all, some or no high frequency content. As described by Jaffe and Smith [2], the fine-tuning allpass filter H_c is necessary to correct for the inaccuracy coming from the quantisation of frequencies due to the use of a delay line of integer length N.

The proposed system is linear, time-invariant, causal and memory dependent (Eq. 3). Given that $f < 1$, where f is the gain value applied to samples passing through the feedback loop, the system is also bounded-input bounded-output stable.

Block diagram

Figure 1 shows the block diagram describing the system's operation. It has four main inputs. The windowed sample data is determined by the window position and length as described above. The delay line length d is calculated from the fundamental frequency determined by incoming MIDI notes. Controls to set the ADSR switch as well as the cutoff frequency of the main lowpass filter H_m are accessible through the user interface. The filters H_a, H_d and H_c are represented in the second row of the diagram. Incoming samples x_n are sent through the delay line, attenuated by the lowpass filter, fine-tuned and further attenuated by the feedback value f. If the ADSR mode is enabled, a volume envelope is applied to each sample coming from the delay line. Finally, the biquad lowpass filter H_m and the main output gain g are applied.

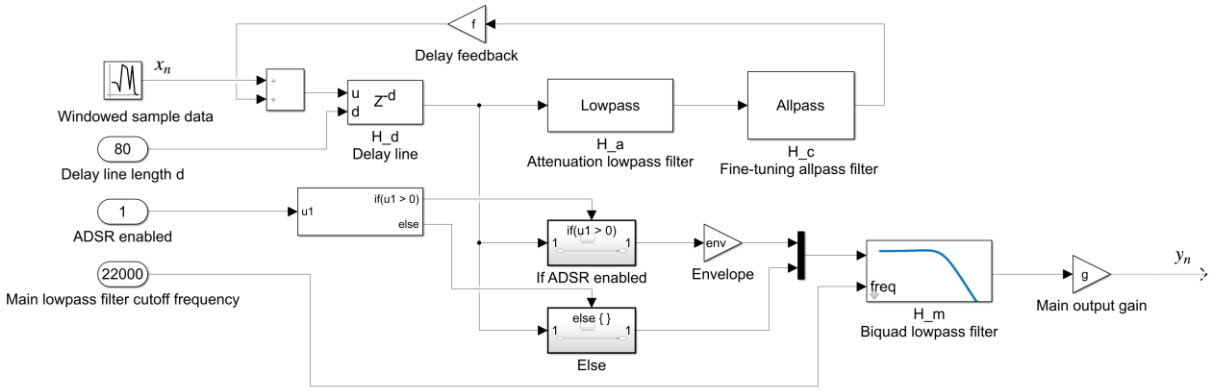

Figure 1: Block diagram of the system's operation

Parameters

Figure 2 shows the user interface and the exposed parameters of the system. The controls are structured into three main sections. The *Global controls* section contains elements to change the fundamental composition of the synthesised sound. The position from which the excitation window is extracted can be configured by changing the *window position* parameter, or by selecting a position in the rendered waveform display. It's current value is indicated by a blue vertical line, with the two lighter lines defining the current window bounds. Its numerical value is mapped to the range [0...1], where 0 corresponds to the beginning of the loaded sample and 1 corresponds to the end. The *window length* parameter defines a range around the window position that is used to extract the source data. Its value is divided by two to obtain the window in the range $[p_w - \frac{w_l}{2}, \ldots, p_w + \frac{w_l}{2}]$, where p_w is the window position in samples and w_l is the window length in samples.

The delay feedback parameter controls the delay feedback value f, which is used to attenuate samples passing through the delay line. As described above, its value can be freely configured in the standard mode, and is automatically set to 1 if the ADSR envelope is enabled.

The *ADSR* section provides parameters to configure the volume envelope. Its values globally control the parameters of the envelopes used in individual voices. The four parameters can be configured to shape the sound in numerous ways. To emphasise a plucked string, a low attack time in combination with a long release time can be used. However, depending on the source sample, the envelope can also be configured to achieve padded sounds that imitate brass and woodwind instrument families, as well as percussive sounds.

The parameters in the *Additional controls* section provide options to further shape the sound without altering its timbral characteristics. If the *dynamic velocity* option is selected, additional gain dependent on incoming MIDI velocity values is applied to the output buffer. As described above, the *Adaptive decay* option will dynamically calculate a stretch factor to apply to the feedback values of individual notes based on their fundamental frequency to equalise decay times. It cannot be enabled in ADSR mode, as the development of the sound is controlled by the volume envelope in that case. The *Pitch bend range* parameter specifies the maximum range by which sounds can be pitch bent using the MIDI pitch wheel control. If the pitch wheel value is changed, the fundamental frequencies of notes are be dynamically adjusted, which

leads to changes in the lengths of the delay lines. This can be used to create slurs, glissandi and vibrato effects. The *Filter cutoff* and *Filter Q* parameters adjust the filter parameters of the biquadratic lowpass filter H_m. The filter's current magnitude response is plotted in the diagram below the control elements. The diagram component is interactive and can also be used to change the two parameters. Finally, the *Main output gain* parameter controls the value of the gain that is applied to the final output. Depending on the volume of the source sample, this parameter can be configured to avoid clipping of the output signal.

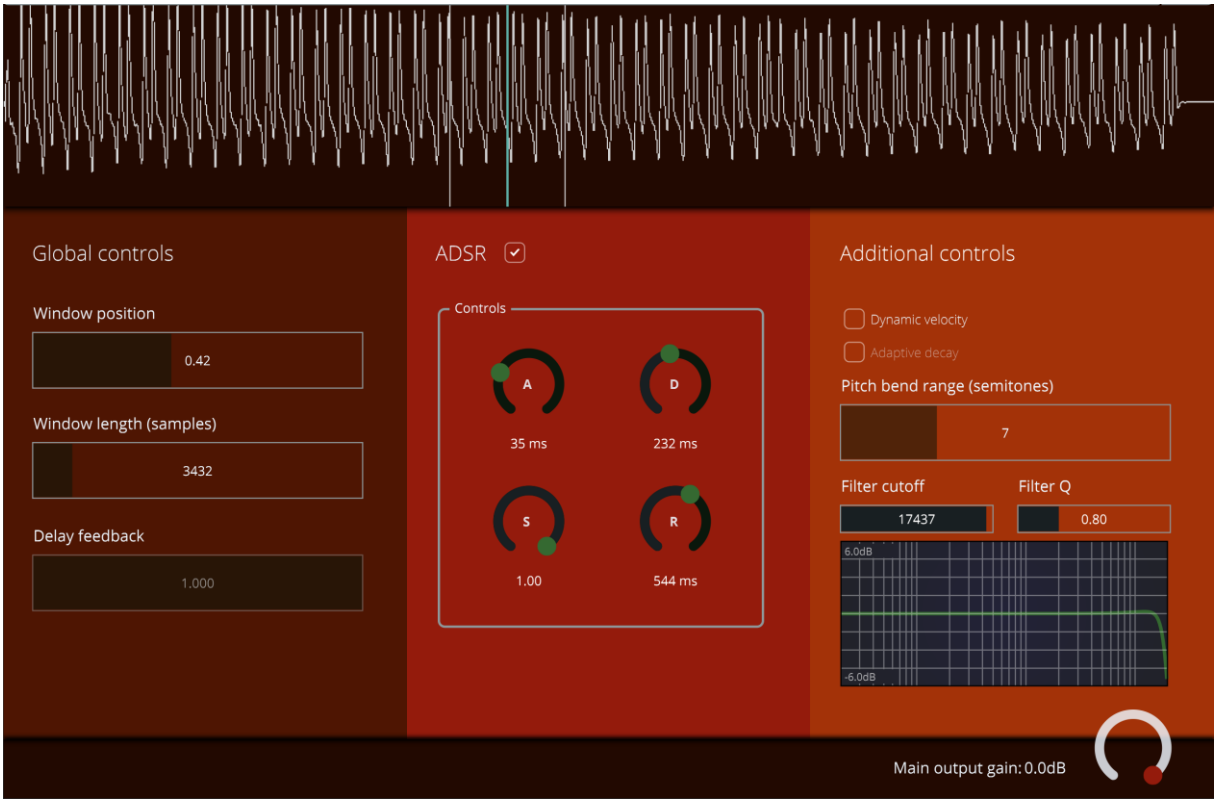

Figure 2: User interface of the system

Results

To demonstrate the system's effect on different kinds of input data, three sound sources were selected. The first item is a recording of a noisy, percussive snare drum. Due to the aperiodic nature of the source signal, the output contains a significant amount of high frequency content above 1kHz, resulting in a bright sound. This is reflected in the magnitude spectrogram of the output, shown in figure 3. The connected dots in white signify the spectral centroid values for each frame. The MIDI note sequence used to generate this output was *G2-11, F2-5, G3-11, F3-5, Eb Major-11 and B Major-5*, with a constant source window length of 27ms.

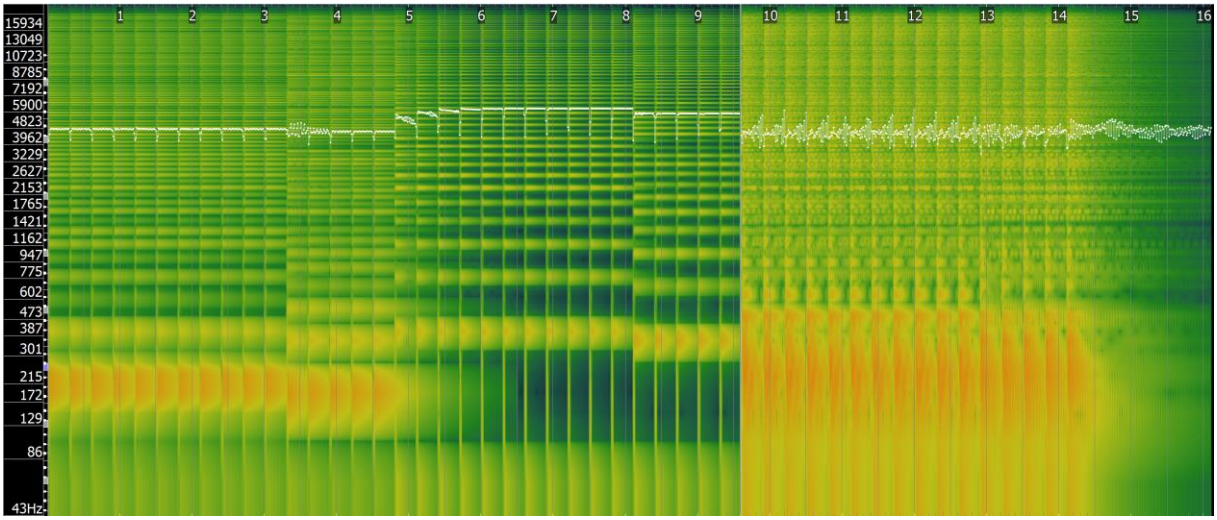

Figure 3: Magnitude spectrogram of synthesised output with percussive input

The second item is a recording of a piano chord. Contrary to the first example, this sound source is highly periodic, which results in a sound with less high frequency content, as shown in the spectrogram in figure 4. The MIDI sequence used to synthesise the sound was the same as for the first item, with a constant source window length of 38ms.

Table 1 shows the mean distribution of spectral centroid values of the three parts (note sequence in octave two, note sequence in octave three, chord sequence in octaves two and three) for the first two recordings. The difference of the mean values is reflected in the distinct timbral qualities of the two outputs.

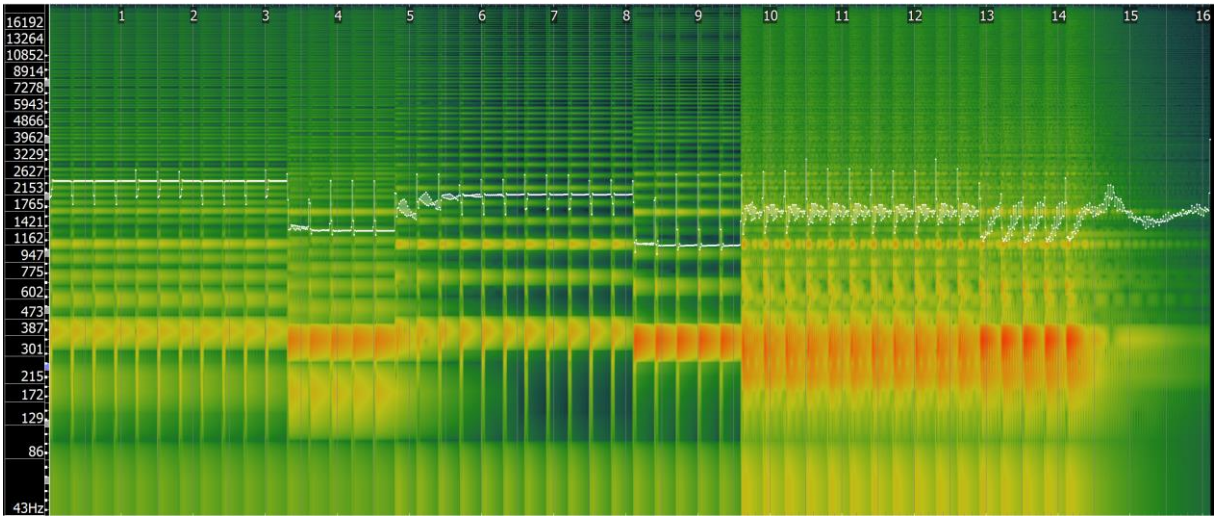

Figure 4: Magnitude spectrogram of synthesised output with piano input

Part	Percussive	Piano
1 (0s – 4.75s)	4285.76Hz	1970.81Hz
2 (4.75s – 9.60s)	4688.06Hz	1649.12Hz
3 (9.60s – 15.00s)	2847.71Hz	1500.87Hz
Overall mean	3940.51Hz	1706.93Hz

Table 1: Comparison of mean spectral centroid values for percussive and piano outputs

Finally, a recording of a pop song was chosen to demonstrate the degree of possible timbral variety within a self-contained source file that can be achieved by adjusting the source position and window length during the synthesis process. Here, the MIDI sequence was changed to *G2-16, G3-16, Eb Major-9.* As indicated in the spectrogram (figure 5), the frequency composites vary significantly during the first part of the output, although the incoming MIDI notes are identical. The same behaviour is observed for the second and third parts of the recording. This was achieved by automatically adjusting the window position as well as the window length after two/four consecutive notes. The resulting output sound contains a variety of different timbres, which were all produced using the same underlying source file.

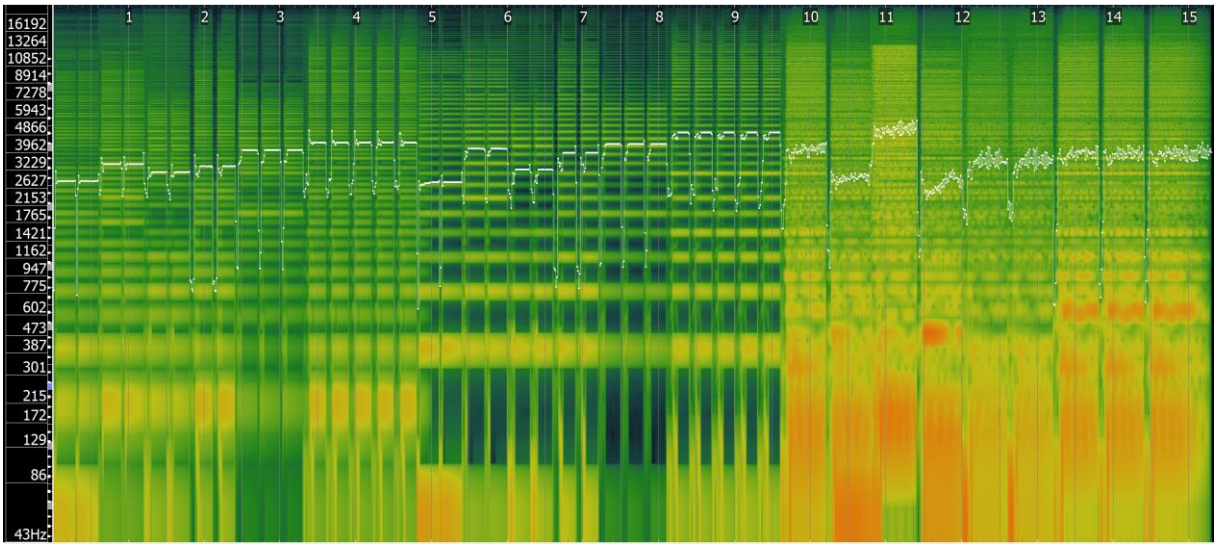

Figure 5: Log-frequency spectrogram of synthesised output with song input

Discussion

It is evident that the number of possible combinations of frequency composites is generally limited by the source material and the window size. However, the use of delay lines of integer lengths poses an additional boundary, due to the rapid decay of frequency components that are not integer multiples of the delay line length. This emulation of perfectly flexible strings, where, after the initial excitation, only harmonic spectral components possess significant amplitude, can be slightly skewed by introducing a coefficient of inharmonicity as described by Jaffe and Smith [2].

The integer delay length is also the main reason why a longer window length, for example a length of several seconds, does not produce results that emulate the timbral characteristics of a string. However, it was found that using a longer window length in the ADSR mode leads to interesting results, especially if incoming MIDI notes share the same key as the source material (in the case that the source material is a tonal sequence, e.g. a recording of a chord progression). This then leads to possibilities of a sort of re-pitching of the source material, which can be exploited for idiosyncratic musical effects.

It is also notable that re-pitching the sound during playback will lead to distinct timbres once the pitch control is returned to its normal state. That is, if the pitch is shifted upwards and back downwards *while a note is played* it will sound differently, due to the output's dependency on both the source material and the data in the delay line.

Future Work

Future work could further investigate the expressive potential of the proposed system. For example, the lowpass filter H_a, used to attenuate the signal in the delay line, could be modified to dynamically adjust its cutoff frequency and quality factor based on a combination of the velocity of incoming MIDI data and the desired fundamental frequency. This would give a performer more direct control over the spectral composition of the plucked strings. Certain audio features, such as the spectral centroid, could be pre-computed for successive windowed sections of the source data in order to visually indicate the spectral characteristics of the sound in the user interface. This would provide a crude, a-priori estimation of the expected output, which could aid in selecting window positions.

Conclusion

We presented a modified version of the Karplus-Strong algorithm for string synthesis that allows for dynamic selection of the source material for string excitation. A classification and description of the system's components were given and the parameters used to control the synthesis process were explained. We evaluated the resulting sounds with regard to the source material and discussed their sonic qualities.

References

[1] Karplus, K., & Strong, A. (1983). Digital Synthesis of Plucked-String and Drum Timbres. Computer Music Journal, 7(2), 43-55. doi:10.2307/3680062

[2] Jaffe, D. A., & Smith, J. O. (1983). Extensions of the Karplus-Strong plucked-string algorithm. Computer Music Journal, 7(2), 56-69.

[3] Välimäki, V., Huopaniemi, J., Karjalainen, M., & Jánosy, Z. (1995, February). Physical modeling of plucked string instruments with application to real-time sound synthesis. In Audio Engineering Society Convention 98. Audio Engineering Society.

[4] Smith, J. O. (1992). Physical modeling using digital waveguides. Computer music journal, 16(4), 74-91.

[5] Karjalainen, M., Välimäki, V., & Tolonen, T. (1998). Plucked-string models: From the Karplus-Strong algorithm to digital waveguides and beyond. Computer Music Journal, 22(3), 17-32.

[6] Karjalainen, M., & Smith, J.O. (1996). Body Modeling Techniques for String Instrument Synthesis. ICMC.

[7] Karjalainen, M., & Smith, J. O. (1996, August). Body Modeling Techniques for String Instrument Synthesis. In ICMC.

[8] Välimäki, V. & Pakarinen, J. & Erkut, C. & Karjalainen, M. (2006). Discrete-time modelling of musical instruments. Rep. Prog. Phys. 69. 1-78. 10.1088/0034-4885/69/1/R01.

[9] Cook, P. & Scavone, G. (2000). The synthesis toolkit (stk). International Computer Music Conference.

Appendix

Submission Notes

Due to the QM+ upload limit, the input file for third item in the Evaluation section (the song) was uploaded to QMUL's filesharing service and is available until 24/05/2020 at the following link:

https://collect.qmul.ac.uk/get/5132P257GNO8ED2FQS/61AIN2V8LAM45T8I29L0JBO/song.wav

The other two discussed source files are located in the *input* folder. The generated output files for all three items are located in the *output* folder.

Both the pre-compiled **.vst3* plugin file and a Windows executable **.exe* file are included in the submission.

Technical details

The system was implemented using the JUCE framework[1] in version 5.4.7. It can be compiled with the Projucer[2], using the `DAFX_Assignment_2.jucer` file.

The system's two main components are the `PluginProcessor`, which is concerned with handling and processing audio data, and the `PluginEditor`, which provides a graphical user interface

[1] https://juce.com/, accessed on 02/05/2020

[2] https://juce.com/discover/projucer, accessed on 02/05/2020

that can be used to control the system's parameters. Voices were abstracted into a separate `Voice` class, which allows for straightforward configuration of the system's level of polyphony. Each voice instance is a self-contained audio subsystem. As such, the `Voice` class contains a significant portion of the synthesis code. It should be noted that the native editor class `PluginEditor` mainly serves as a wrapper for the custom `JUCEEditor` class that was partly generated by the graphical user interface builder that is supplied by JUCE. This was done to systematically structure the user interface.

The `Delay` and `DelayLine` classes contain the functionality for a delay object and its integrated delay line respectively. They were adapted from the standard delay classes described in the JUCE guidelines. The `SamplePanel` class bundles the functionality to load, store and display audio sample data. The `FilterGraph` and `FilterInfo` classes were adapted from Sean Enderby's implementation[3] to allow for interactive filter control. The original code contains the functions used to plot the magnitude response of a digital filter, in this case the biquad lowpass filter. The `Utility.cpp` class contains helper functions. The `Constants.h` class defines system-wide constants.

Version Control

The source code is available publicly at:

https://github.com/maxgraf96/DAFX_Assignment_2

[3] https://sourceforge.net/p/jucefiltergraph/wiki/Home/, accessed on 02/05/2020