MUSIC AND AUDIO PROGRAMMING

REAL-TIME PHYSICAL MODELLING MIDI CONTROLLED SYNTHESIZER

Clifford Moses Manasseh
ec19461@qmul.ac.uk

Friday 28th May, 2021

1 Overview of the task

The Aim of this project is to create a real-time playable physical modelling synthesizer with two models (i) Extended Karplus-Strong model (ii) Digital waveguide jet-woodwind model playable through a midi controller. A video containing the demonstration of this project is available at `https://youtu.be/LWot4nR_Q_o`

Abstract

Physical Modelling synthesis for musical instruments models the physics of the instrument through mathematical models to simulate the sound of the actual instrument. It can be thought of as virtually building the instrument inside the computer. This project implements the Extended Karplus-Strong algorithm [3] and a digital wave guide jet - wood wind model [1] with up to a seven note polyphony on the bela micro-controller. They Physical models can be excited using a midi controller.

2 Introduction

Physical Modelling synthesis is a more recent synthesis method which is compute intensive unlike other synthesis methods like wavetable synthesis, subtractive or additive synthesis. An accurate physical modelling synth has to sound exactly like the physical instrument after which it is modelled and this would require replicating all the physics behind every component that contributes towards the sound of that instrument. This allows for the physical model to be expressive like the real instrument for example, when attempting to model a Grand Piano, the force exerted by the hammer on the strings, the length of the strings, dispersion of vibrations in the string, sound radiation in the soundboard, force exerted by the dampers, resonance of the piano body etc., should be considered to create a physical model that sounds exactly like a piano and each of these calculations involve intense mathematics. Early computers were not able to compute these calculations in real time to become a playable instrument. But as computational power has grown over the years, It is now possible to compute very complex physical models in real-time. Physical modelling can be generalized into two categories (i) Lumped models that use masses, springs, dampeners and other non-linear elements (ii) Distributed models that use Delay lines and other non-linear elements [9]. This project implements two different types of physical models. An Extended karplus strong algorithm (EKS) and a digital waveguide jet woodwind model. The EKS algorithm is a combination of both physical and spectral modelling techniques. It uses a digital waveguide of varying buffer size to model vibrations in a physical medium. Like

the name suggests the EKS technique for modelling plucked strings is based on previous work [5] by kevin karplus and alex strong. the second implementation, the jet - woodwind model uses bi-directional waveguides to simulate the jet reed and the acoustic tube of a jet-woodwind instrument. a bi-directional waveguide uses two delay lines instead of one to model vibrations happening inside a medium, in this case of a jet-woodwind model the waveguides are used to model an acoustic tube with air vibrations that produce sound which is identical to how a flute works. Expressiveness of physically modelled instruments also makes it stand out from the other synthesis methods. Using physical modelling it is possible to create meta instruments that contain components from different musical instruments like for example, the bow of a violin playing a guitar string or a violin string played by a piano hammer.

3 Background

The earliest work in physical modelling dates back to the 1962 speech synthesis work by J. L. Kelly and C. C. Lochbaum [7] which was an attempt to model the human vocal tract. the first physical model of a musical instrument was the vibrational string model done by Ruiz in 1969 [8]. In 1983 karplus and strong discovered digital waveguide synthesis accidentally, they discovered that averaging random noise in a 8 bit-shift register created a sound similar to a guitar. this algorithm was named the digitar. The EKS model proposed by jaffe and smith in the same year with additions to the karplus strong model [3]. They important feature was to be able to tune the digitar string to western musical notes. This extension also included methods to change the pick position, the pick direction and the decay and brightness of the plucked string sound making it a closer representation of an actual an electric guitar.The second implementation, The woodwind model is based on the schematic diagram of work by perry cook [1] digital wave guides as a solution for an one dimensional wave equation. This model is represented by two parallel delay lines propagating in opposite directions.These delay lines are referred to as Digital wave guides in physical modelling synthesis. The purpose of a delay line is to induce a time delay between the input and output samples. In general, for creating software synthesizers or digital audio effects a block of memory is filled with samples and then it is processed using different digital signal processing techniques. In digital waveguide synthesis delay lines act as the block of memory. they are often implemented as circular buffers. A circular buffer can be thought of an endless loop where the end wraps back to the beginning. So, when writing samples into a circular buffer, when the end of the circular buffer is reached the next sample is written back to the first position. This way the circular buffer holds a history of previously chunk of samples. The circular buffers often have an associated pointer variables that keep track of where in the circular buffer the sample is being written into to ensure that samples are written back to the start when the end of the circular buffer is reached. Julius. O. Smith the founder of digital waveguide synthesis defines the digital waveguide synthesis to be made up of delay lines, digital filters and other non linear elements. the digital waveguide represents a sampled acoustic travelling wave and the digital filters and nonlinear elements account for the physical properties of the acoustic system [9]. These components account for modelling features like air pressure inside a flute or the striking force of a piano hammer. Physical modelling has seen rapid growth in the recent years with access to ample computing power.

4 Design

This section contains information about the design process for the real-time implementation of the physical modelling instruments in bela. After some literature survey the initial test prototype

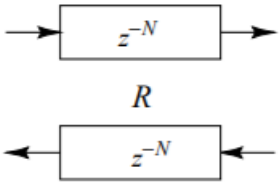

Figure 1: lossless digital wave guide [9]

was a gui based karplus strong model that had slider control for pitch and a button for striking the string. It was based on the gui to bela example in the bela IDE. The initial implementation only contained a variable delay line with an averaging filter and a noise impulse input. The three important core features of the design are the digital filters, the noise input and the digital waveguides.

digital waveguides: These are delay lines with a variable buffer length that store samples that can be accessed using a dedicated pointer variable.

noise input: A white noise impulse is used as an input to trigger the plucked string model and a continuous stream of white noise is used to simulate blowing air into the flute physical model.

digital filters: different types of filters explained in **??** are used to implement components like pick position, pick direction, dynamic level, decay, reflection, string stiffness, tuning of the physical models, the digital filters also include methods to calculate the phase delay to contribute to the fractional delay line for accurate tuning.

4.1 Extended Karplus-Strong algorithm

The extended karplus-strong algorithm defined in [3] uses one digital wave guide z^{-N} to represent a vibrating string. The primary filter is a two point averaging filter $H_a(z)$ that loops back into the delay line. the input noise impulse if filtered through a comb filter to define pick position $H_\beta(z)$, A pick-direction low pass filter $H_p(z)$, a dynamic level low pass filter $H_L(z)$- A second loop filter section that contains a set of 4 biquads to simulate string stiffness$H_s(z)$, a lagrange string tuning filter $h_\Delta(n)$, and a one-zero string dampening filter $H_d(z)$. This algorithm uses an noise impulse response as input to the digital waveguide. The modifications made in the design that varies from the original paper are the lagrange interpolation filter for the fractional delay line and the usage of biquad all pass filters to simulate string stiffness. the overall transfer function of this system can be given by

$$H(z) \triangleq \frac{1}{1 - H_a(z)H_\beta(z)H_p(z)H_L(z)H_s(z)h_\Delta(n)H_d(z)}$$

4.2 jet-woodwind model

The jet-woodwind model **??** is based off Perry cook's slide flute model [1] with an additional sine wave in the input to add a vibrato. This model uses a sigmoid function $x - x^3$ to model the non-linearity of the acoustic tube. There are two delay lines. the delay line $dl2$ models the jet reed as implemented by [4] and the other delay line $dl1$ models the acoustic tube. The one pole

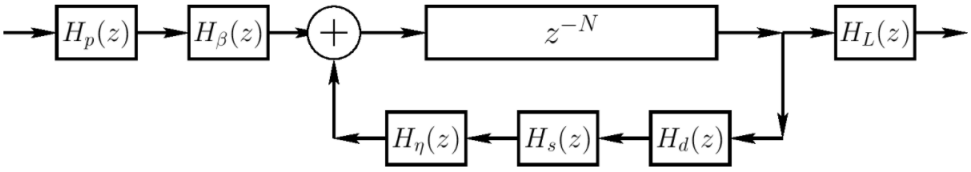

Figure 2: Extended Karplus Strong Algorithm [9]

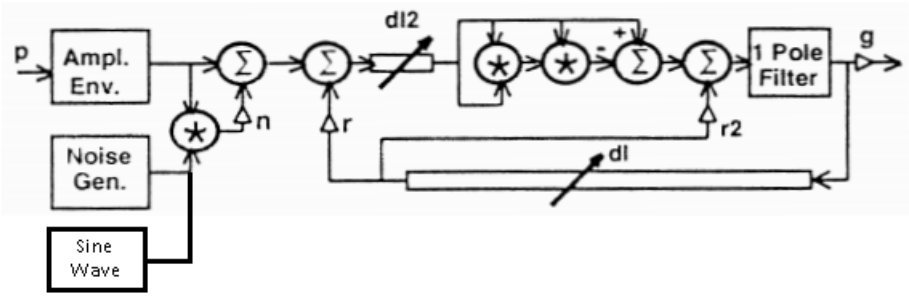

Figure 3: jet-woodwind model [1]

filter at the end models the low pass reflection at the end of the acoustic tube. The reflection coefficients r and $r2$ model reflection gains across the acoustic tube. r is the reflection gain at the end of the mouth piece and $r2$ is the reflection gain at the end of the acoustic tube. g is the over all gain output gain. A Dynamic-level low pass filter is also attached to the input values to be able to control the flute using the midi controller's note velocity.

4.3 Filters

This implementation uses up to 5 different types of filter to achieve the desired sound. The different filters used in the implementation are discussed below. Detailed descriptions of Pick position Comb filter, pick direction low pass filter, one zero string damping filter, first order string tuning all pass filter and the dynamic level low pass filter can be found in [3] the design is based on descriptions by J.O. Smith in his book [9]. A loop gain g is multiplied with the two point averaging filter and in the pick position comb filter in order to keep the loop gain below 1 so that no feedback occurs and the model is stable.

Two Point Averaging Filter The Two point averaging filter is a low pass filter that averages the current and the previous sample. the transfer function is given by

$$H_a(z) = \frac{z + z^{-1}}{2}$$

. The phase delay caused by the two point averaging filter is 0.5 samples.

Pick Position Comb Filter This filter used in the EKS model is used to pick where along the string length between the nut and the bridge the excitation occurs. This comb filter $H_\beta(z)$ has a transfer function

$$H_\beta(z) = 1 - z^{-[\beta N + 1/2]}$$

. Here $\beta \in (0,1)$ defines the pick position between the nut and the bridge of the virtual string. N is the total length of the delay line. The phase delay caused by the pick position comb filter need

not be considered because it does not cause any phase delay after the noise signal enters into the digital wave guide.

Pick Direction Low Pass Filter The pick direction low pass filter is used to define the direction in which the pick strikes the string. that is if the pick is an upstroke or a down stroke. this filter is defined by the transfer function

$$\frac{1-p}{1-pz^{-1}}$$

. the value of p defines if it is a downwards pick or an upwards pick. p takes values between 0 and 1 and a lower value represents a down stroke and an upper value represents an up stroke. Upstroke values result in a more open sound and down stroke values result in a muted sound. The phase delay caused by the pick direction filter also can be safely ignored.

One Zero String Damping Filter The one zero string damping filter is a weighted two point average filter given by the transfer function

$$H_d(z) = \rho((1-S) + Sz^{-1})$$

where ρ is the loss factor and is defined by

$$\rho = (0.001)^{PT/t_{60}}$$

where t_{60} is the decay time in seconds. the phase delay P_a caused by the damping filter should be calculated in order to fine tune the string, this is calculated using the following equation mentioned in [3]

$$P_a(f,S) = -1\frac{1}{\omega T_s}(\frac{-S\sin\omega T_s}{(1-S)+S\cos\omega T_s})$$

here S is the stretching factor T_s is the decay time. this filter in addition to affecting the decay time of the string also affects the brightness of the string timbre.

First order string tuning all pass filter An Allpass tuning filter has been implemented but has not been used. The allpass tuning filter can be used to interpolate the delay line to any fractional value Δ using the all pass coefficient η. for any desired fractional delay η can be calculated by

$$\eta = \frac{1-\Delta}{1+\Delta}$$

the transfer function for the all pass delay is given by the equation

$$H_\eta(z) = \frac{\eta + z^{-1}}{1+\eta z^{-1}}$$

the value of η is offset between $\epsilon, 1+\epsilon$ to avoid tuning delays close to zero for the all pass filter.

DC blocking filter A DC blocking filter is added to the end of the woodwind model to problems caused by the DC offset. the transfer function of the DC blocking filter is given by

$$H_{dc}(z) = \frac{1-z^{-1}}{1-Rz^{-1}}$$

and R value of 0.995 is sufficient for a $f_s = 44100$.

Dynamic Level Low-Pass Filter This filter is mapped to the input velocity and is responsible for making the models velocity sensitive. The transfer function is given by

$$H_l(z) = \frac{1-R_l}{1-Rlz^{-1}}$$

.The output of this filter $y(n)$ is then processed along with its input $x(n)$ using the equation

$$L.L_0(L).x(n) + (1 - L).y(n)$$

where $L - 0 = L^{1/3}$

Lagrange Interpolation Filter The Lagranage Interpolation method which is used for tuning the string in this implementation can be thought of as a fourth order linear interpolator. It takes one future sample, the current sample and three previous samples into consideration to interpolate for a fractional value between the current and the previous sample. the lagrange tuning filter $h_\Delta(n)$ is given by

$$h_\Delta(n) = \prod_{k=0}^{N} \frac{\Delta - k}{n - k} \quad k = 0, 1, 2, 3..., N \quad and \quad k \neq n$$

.

String Stiffness Biquad All Pass String stiffness can be simulated by using all pass filter. this is done by varying the phase delay time for different frequencies. In stiff strings, higher frequencies pass much faster than lower frequencies, the biquad all pass filter is of the form

$$H_s(z) = g\frac{1 + b_1 z^{-1} + b_2 z^{-2}}{1 + a_1 z^{-1} + a_2 z^{-2}}$$

, four filters are cascaded at different design frequencies to achieve an higher order all pass filter with different phase delays at different design frequencies. the string stiffness implementation is a port from the stifkarp example module present in STK[1] [2]. the phase delay in samples of a biquad can be calculated using the following equations[2]

$$\tau_p(\omega) = \frac{-atan2(n, m) + atan2(q, p)}{\omega}$$

m = g($b_0 + b_1 cos(\omega) + b_2 \cos(2\omega)$)
n = g($b_1 \sin(-\omega) + b_2 \sin(-2\omega)$)
p = $a_0 + a_1 \cos(\omega) + a_2 \cos(2\omega)$
q = $a_1 \sin(-\omega)) + a_2 \sin(-2\omega)$

4.4 Tuning the Digital Wave guide

One of the prime problems with physical modelling instruments is tuning. Since the delay lines are implemented using integer value sized buffers it is impossible to tune them to fractional values of pitches. for example a C5 note is 521.48Hz but the integer sized delay line buffer can only be of the length 521 or 522. The fractional delay 0.48 has to be interpolated using an interpolation technique to keep the string in tune. In addition to that the phase delay contributed by the filters to the system should also be considered.In the EKS model for a given pitch f and a sample rate f_s the overall phase delay caused by the filters is

$$\Delta = \tau_p(\omega) + P_a(f, S) + 0.5$$

The required integer value length of the delay line is

$$d = f_s/f - \Delta$$

[1] https://ccrma.stanford.edu/software/stk/
[2] https://dsp.stackexchange.com/questions/67504

the fractional delay is given by

$$d_f = f_s/f - \Delta_i - d$$

where Δ_i is the integer part of the phase delays. an offset of 2 is added to the fractional delay value d_f to get better tuning results since the interpolation value should be kept around the range of N/2 where N is the order of the interpolator[9]. In the Woodwind model tuning is achieved by adding two interpolators at the end of the jet delay line and the acoustic tube delay line, however perfect tuning was not achieved especially for higher octaves. tuning for higher pitches in a woodwind model has to be further investigated. The following table depicts pitch in hertz observed across 6 octaves in comparison with the actual midi note frequency for both the implemented models

Midi Note Number	pitch in Hz	EKS	confidence	Woodwind	confidence
1	65.41	65.305	0.943045	65.275	0.952659
13	130.81	130.767	0.935139	130.179	0.908332
25	261.63	262.357	0.964625	261.129	0.952830
37	523.25	522.972	0.95596	519.108	0.959103
49	1046.50	1039.571	0.957119	1023.059	0.873401
61	2093.00	1974.258	0.898421	1983.949	0.002846

Table 1: comparison between the required note pitch and the pitch of the two models

4.5 Utilities and Midi functionality

parameters from the filters that contribute to the sound of the instrument are exposed and mapped to midi control change messages. The program change parameter has two values 0 and 1 to choose between the EKS sound or the woodwind sound. An additional utility envelope function was added towards the end of the implementation to the woodwind model to shape its transient using an attack and a release curve. The list of controllable parameters for EKS: pick position (midi cc 21), pick direction (midi cc 22), brightness (midi cc 23), decay time (midi cc 24), stiffness (midi cc 25), dynamic level (midi note velocity) and for the woodwind model: reflection coefficient 1 (midi cc 26) and 2 (midi cc 27), over blow (midi cc 44), attack (midi cc 42) and release (midi cc 43), vibrato depth (midi cc 28) and the program change is controlled using midi cc 41.

5 Implementation

All of the above mentioned features is implemented on the Bela platform for real-time performance with an 7 note polyphony. The digital waveguides are implemented using c++ vectors, vectors were chosen over arrays because of their re sizeable and flexible nature. However to prevent edge cases leading to segmentation faults vectors of a predefined size much larger than required by delay line sizes in the playable range are implemented and a variable holding the required buffer size value wraps the write pointer back to the start as soon as the buffer size limit is reached. The model performs well at a sample buffer size of 512 samples at 44100Hz but suffers maximum cpu usage when using full polyphony for the EKS model. the woodwind model performs well at full polyphony. The whole program is implemented as follows. the render module contains the midi functionality and objects to set audio parameters and note

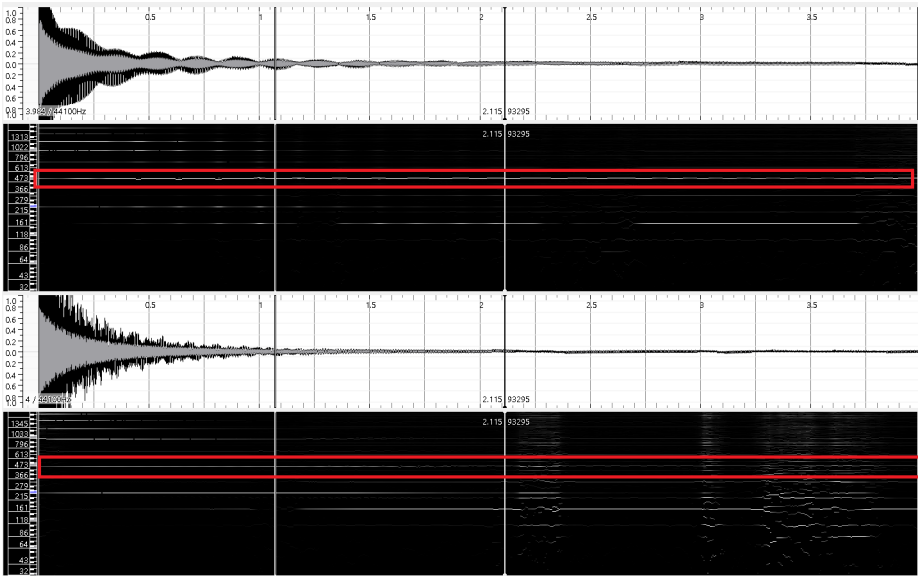

Figure 4: Red box highlights effect of string stiffness on lower harmonics
Top: Maximum string stiffness: Bottom:Minimum string stiffness

on/off messages to the polyphony manager. The polyphony voice manager distributes the input data to the right model based on the chosen program through a number midi cc message and assigns voices to note inputs. A filters module containing all the necessary filters receives the noise buffer from the instruments on note on and processes them based on the block diagram implementation in the respective instruments module. A utility module containing the attack release envelope used to shape transients for the woodwind model.

6 Evaluation

The tuning of both the models were evaluated using crepe[3] [6] a deep convolutional neural network for monophonic pitch tracking. To carry out this process output of both the models were recorded playing the note C with a uniform velocity across 6 octaves. in EKS only note 61 was recorded multiple time due to its short decay time. Then the audio was prepocessed by trimming silence and the loudness was normalized in a DAW. This processed audio was sent into the crepe model for fundamental pitch $f0$ estimation. The result csv file contained $f0$ values for all the input notes along with confidence values for the detected pitches with the time stamps and a fundamental pitch plot. The values are listed out in table 1. String stiffness filter was evaluated by observing spectrogram outputs for extreme stiffness values. It was clearly seen that for the lower extreme the decay of lower frequencies was much faster than for higher extremes **??**, However the string stiffness filters also contributed a slight detune to the lower harmonics of the plucked string. One main hurdle during the evaluation process was the constant digital noise generated from bela around 172Hz. Since it was prevalent, this was wrongly identified as the fundamental pitch by the neural network sometimes. In addition to that manual analysis was performed using the sonic visualizer to evauate the different audio parameters.

[3] `https://github.com/marl/crepe`

7 Discussion

Common problems faced in this implementation are high cpu usage for the EKS model. Look up tables for filter coefficients and approximations for functions like *atan*2 can make the program more lighter. It can be observed that both models despite of using a higher order lagrange interpolation suffer from detuning at the highest octave. It was also observed that despite calculating phase delays, at very high pitches the decay time and decay factor variables from the string damping filter were capable of detuning the string by contributing more phase delay to the loop if tweaked, however this was not observed for midi notes lesser than 61 also The string stiffness filter tends to slightly detune the lower harmonics of the pitch when changed, these can be investigated further to make a more robust implementation. A better method for loop filter design using sinusoidal model analysis is mentioned by J.O.Smith[4] which can potentially solve the detune caused by the string stiffness filter. One other issue that was not investigated was the instability of the woodwind model during initialization. The model only started sounding like a flute after playing a few notes and sometimes it was observed that on the first run the tuning was random for input notes which stabilized only after playing since notes however the cause of this issue was not further investigated.

8 Conclusion

Hence, robust implementations of two physcial models, the Extended karplus strong algorithm and the Jet-woodwind model controllable via midi were done successfully and the tuning of the instruments were evaluated using crepe.

References

[1] P. Cook. A meta-wind-instrument physical model, and a meta-controller for real-time performance control. In *ICMC*, 1992.

[2] Perry Cook and Gary Scavone. The synthesis toolkit (stk). 08 2000.

[3] D. Jaffe and J. Smith. Extensions of the karplus-strong plucked-string algorithm. *Computer Music Journal*, 7:56, 1983.

[4] Matti Karjalainen, Unto Laine, Timo Laakso, and Vesa Välimäki. Transmission-line modeling and real-time synthesis of string and wind instruments. 10 1991.

[5] K. Karplus and A. Strong. Digital synthesis of plucked-string and drum timbres. *Compute Music Journal*, 7:56, 1983.

[6] Jong Kim, Justin Salamon, Peter Li, and Juan Bello. Crepe: A convolutional representation for pitch estimation. pages 161–165, 04 2018.

[7] C Lochbaum and K Kelly. Speech synthesis. *Proc. Fourth ICA*, 01 1962.

[8] P. Ruiz. A technique for simulating the vibrations of strings with a digital computer. 05 2021.

[9] Julius Smith. *Digital Waveguide Architectures for Virtual Musical Instruments*, pages 399–417. 01 2009.

[4] `https://ccrma.stanford.edu/~jos/pasp/General_Loop_Filter_Design.html`