Final Project Report

Ahmet Can Ozbek

Synthesizing Guitar Sounds and Listening Evaluations

Project Overview

The goal of this project is to create realistic guitar sounds. For this purpose, I first researched physical modeling concepts of a single guitar string. After this research I concluded that implementing Digital Waveguide Modeling and Karplus-Strong Algorithm would be good choices for my project. Comparing the results of these two methods' implementation, which is done by Matlab, showed me that the synthesized guitar string sounds which is done by Karplus-Strong algorithm is more satisfactory than the ones with Digital Waveguide Modeling.

The next step of my project was to do listening evaluations regarding how realistic the outputs sound. My listening test material is the synthesized sounds of all of the open six guitar strings, which is created by Karplus-Strong algorithm. I made a listening evaluation test to my test subjects to find out which guitar string sound is more realistic. By doing this, I am also investigating which frequencies of synthesized sounds are more realistic for Karplus-Strong algorithm.

In the last part, which is the fun part of this project is to play a simple musical piece by these synthesized guitar string sounds. For this, I used an image processing system that was implemented by me for another class. This system can read simple sheet music and give the notes of the piece as output. These notes, which are the outputs of the image processing system, are played with the sounds that are generated with the Karplus-Strong algorithm.

Techniques Used For Synthesizing the Guitar String Sounds

1)Digital Waveguide Synthesis

Digital Waveguide Synthesis is a method that aims to simulate wave propogation. For this method physical properties of a guitar string is used. First of all, to physically model a guitar string, we need an equation that governs the wave propagation nature.

$$\boxed{\mu \frac{\partial^2 y}{\partial t^2} = F_x \frac{\partial^2 y}{\partial x^2}}$$

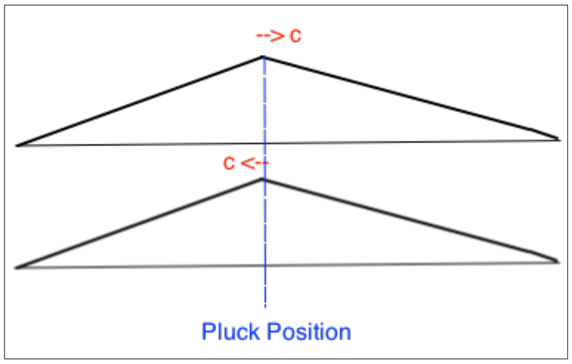
The equation above represents the wave equation for an ideal string. In this equation the variables represent:

μ = the mass per unit length of the string
F_x = string tension

The variable 'y' represents the amplitude of the wave in vertical direction. 'y' is a function of both the location of the x-axis and time. A traveling wave solution for this equation, known as the D'Alembert solution yields:

$$\boxed{y(x,t) = y_+(x - ct) + y_-(x + ct)}$$

where $y_+(x - ct)$ and $y_-(x - ct)$ are traveling waves in opposite directions with velocity c. This solution suggests that the vibration can be modeled as addition of two waves traveling in opposite directions, and the starting point of the waves is the plucking point of the string. The bridge and the nut of the guitar are the two ends of the string and these end points are going to be regarded as reflectors with some damping factor in our synthesizing implementation. The figure below illustrates this concept.



Digital Waveguide Synthesis Implementation

In my project, I used Matlab to implement these approaches. To implement the wave solution, two arrays are used to store the amplitude values of the two waves that travel in the opposite direction. Initially, the waves are considered to be in the shape of a triangle, which starts at the plucking position. We are shifting the values of arrays by the sampling rate to simulate the propagation of the waves. When the wave reaches the ends of the guitar string, it is damped and reflected back with inverted polarity. For this purpose a damping factor of 0.99 is used. In the software, the values that reach to the ends of the arrays are multiplied by 0.99. As the vibration of the guitar string is modeled by these two opposite traveling waves, we should sum the values of these two arrays to get the total wave. If the sound needs to be picked up at a certain location (for example the locations of pickups in an electric guitar), the values at a specific index of the array is considered, because the array index represents the location of the x-axis.

In this implementation, it should also be stated that the array length is analogous to string length. As the frequency of a string vibration is determined by string length, our length of the array is going to determine our output frequency of the sound. Therefore we should choose the length of array according to a rule:

In discrete time, the velocity of the wave is going to be F_s (samples/second), where F_s is the sampling rate and it is equal to 44.1kHz in my implementation. The velocity of the wave is equal to wavelength times the frequency:

$$c = \lambda . f$$
$$c = Fs \Rightarrow \lambda = Fs / f$$

For the fundamental frequency vibration of the string length, which represents the array length in discrete time, is equal to half of the wavelength.

$$string\ length = \lambda / 2$$

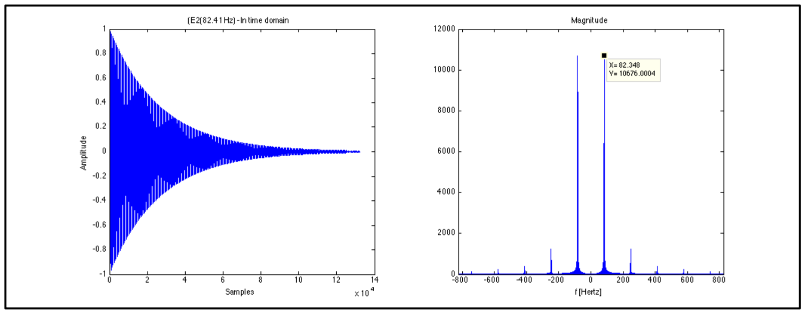
Therefore, the string length is equal to:

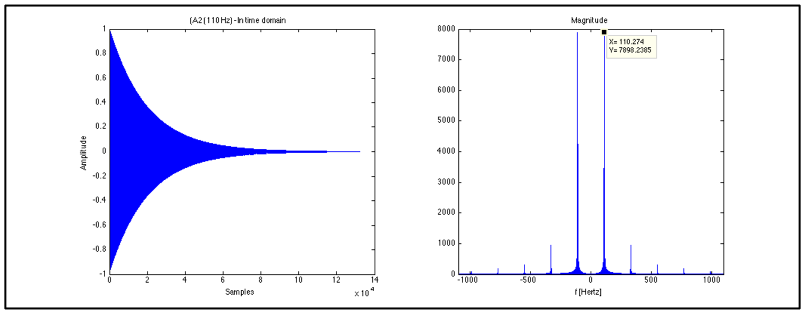
$$string\ length = (Fs / f)/2$$

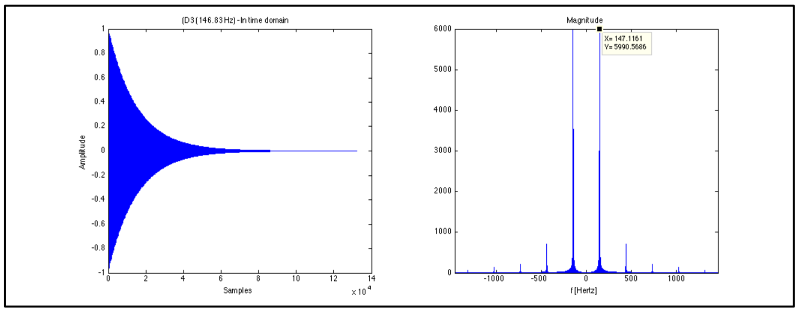
After getting all the parameters I need for this implementation, I created all six strings of a guitar using Matlab, the table below shows the name of the notes and their corresponding frequencies:

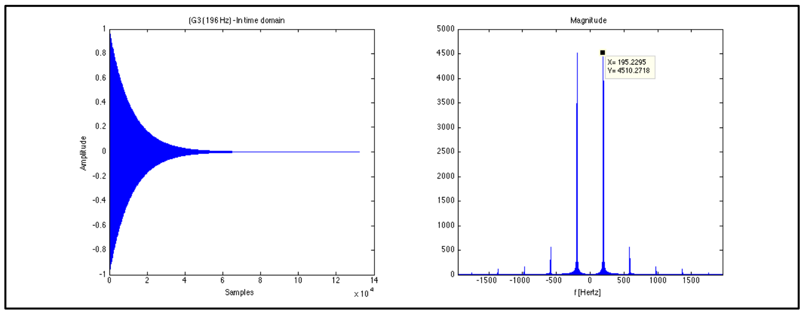
E2	82.41 Hz
A2	110 Hz
D3	146.83 Hz
G3	196 Hz
B3	246.94 Hz
E4	329.63 Hz

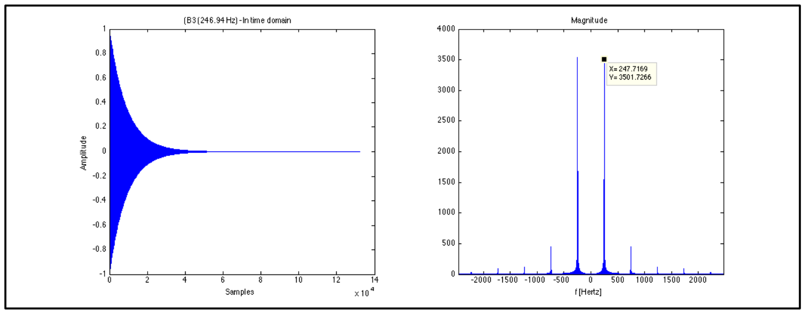
For digital wave guide implementation, these figures show us how synthesized guitar string notes behave in time domain and frequency domain. Here, we have all six guitar strings.

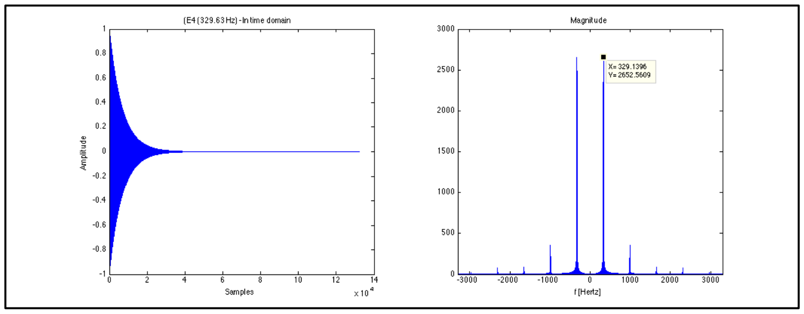










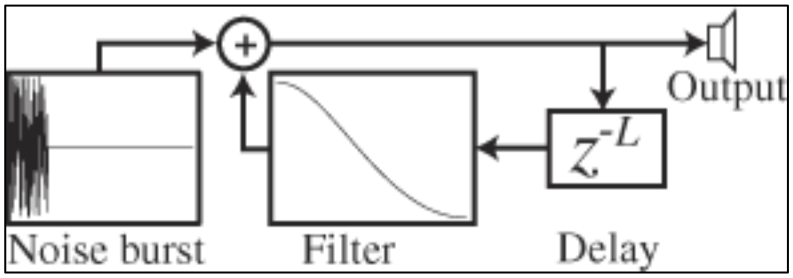


Results of Digital Waveguide Synthesis Implementation

This implementation did not give satisfactory results. The output sounds are not realistic enough to make listening evaluation tests, therefore we switch to a better and a popular method which is Karplus-Strong algorithm.

2)Karplus-Strong Algorithm

The figure below, illustrates the implementation of Karplus-Strong algorithm. The method starts with generating "p" random samples, which is called "noise burst" in the diagram. Then these random samples circulate inside of a feedback loop, associated with a filter. As these "p" samples run through the feedback loop, the output is periodic although the initial samples are random to begin with.



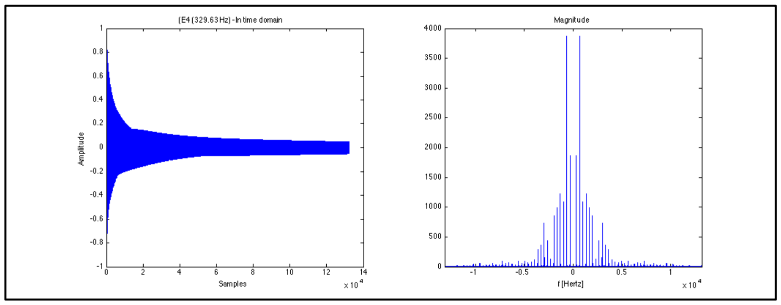
As the period of the output signal is going to be the number of samples in the initial randomly generated signal which is "p", the frequency of the output signal is going to be (F_s / p), where F_s is the sampling frequency and in my implementation F_s is taken 44.1kHz. Therefore, If we want to generate a sound with a frequency "f", p should be ($p = F_s / f$). That is how we decide the length of the random initial samples.

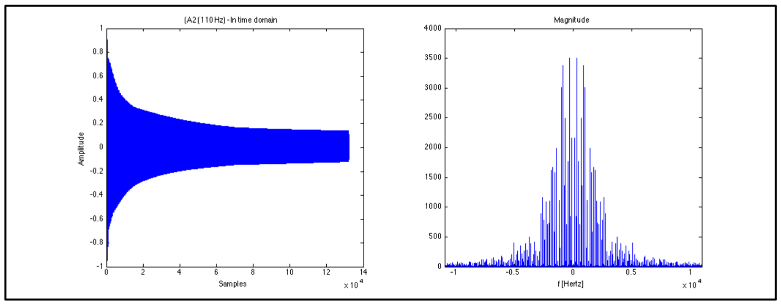
The implementation of Karplus-Algorithm in Matlab is simple. In my implementation, I started with generating "p" random samples. Then, the feedback loop is realized by simple averaging. The two samples at the beginning of the random array are averaged and appended to the end of the random array. This procedure is done repeatedly until we have the desired duration of the note. If we want a duration of a note in "t" seconds, as our sampling rate is F_s , this procedure should repeat until we have ($t * F_s$) samples. The difference equation below, explains this concept.

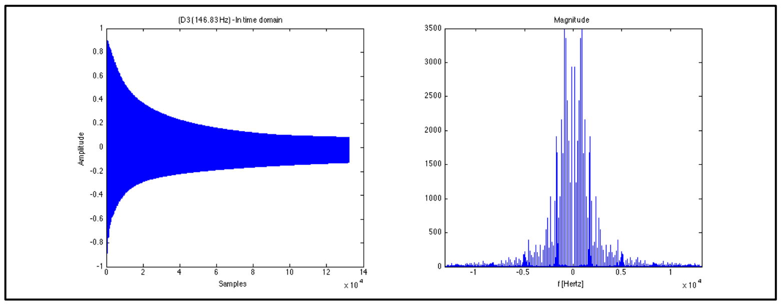
$$\boxed{Y(k) = 0.5*(Y(k-p) + Y(k-p-1)).}$$

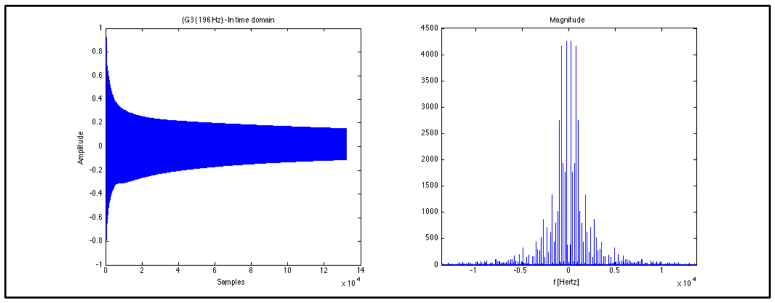
There are some properties that make Karplus-Strong a good algorithm. Firstly, different harmonics have different decay rates, this property helps the naturalness of the sound. In addition to this, as we are starting the note with an array of initial random values, which contain many high harmonics, the signal generated is also going to be abundant in high harmonics and this helps the sound be realistic.

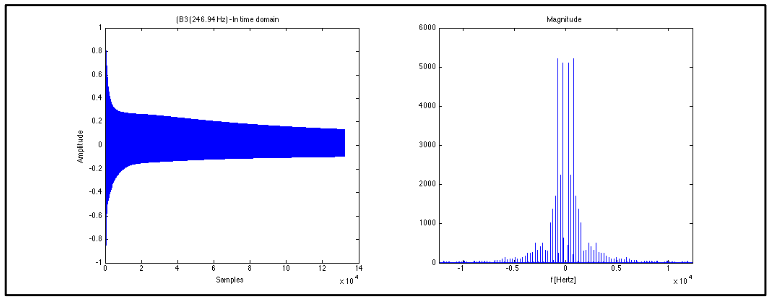
ForKarplus-Strong algorithm, these figures show us how synthesized guitar string notes behave in time domain and frequency domain. Here, we have all six guitar strings.

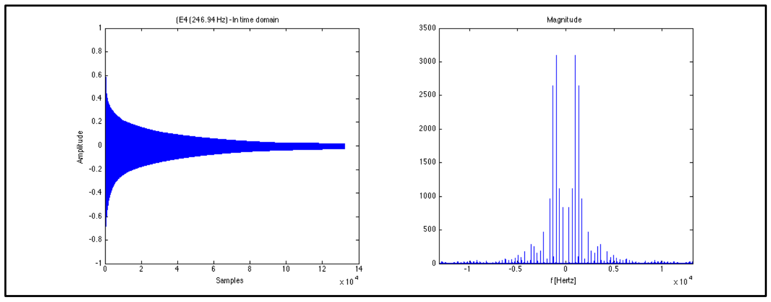












If we compare the spectrums of Karplus-Strong algorithm implementations and the ones with Digital Waveguide method , we can see that the ones with Karplus-Strong algorithm are richer in harmonics. This helps the naturalness of the sound and makes it more realistic.

Listening Evaluations for Karplus-Strong Algorithm

Listening evaluations are performed to determine which synthesized guitar string (frequency) sounds more **realistic**.

Methodology: There are six synthesized guitar open string sounds(E2,A2,D3,G3,B3,E4), I took every two combinations out of these six and put those two string sounds in sound files in which the two notes played consecutively.

As there are six synthesized guitar sounds, there are (6 choose 2) = 15 sound files, that allows test subjects to compare each string sound with the other string sounds.

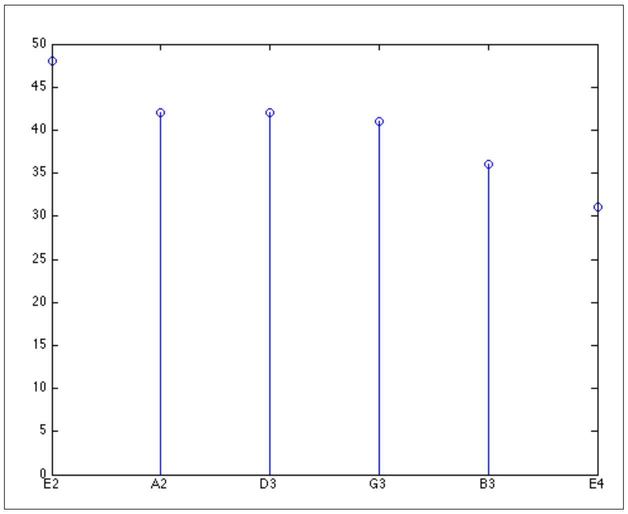
Although all sounds are synthesized, I told the test subjects that in each file, one of the sounds is a real recording and the other is synthesized and asked them to try to recognize the real one. This way, I forced them to find the one that sounds more realistic.

This table shows the data I collected for this experiment. The notes on the table are the ones that the test subjects thought it was the real one compared to the other.

	A2vB3	A2vD3	A2vE4	A2vG3	B3vE4	D3vB3	D3vE4	D3vG3	E2vA2	E2vB3	E2vD3	E2vE4	E2vG3	G3vB3	G3vE4
Ameya	B3	A2	E4	G3	B3	B3	E4	G3	E2	B3	E2	E2	G3	B3	E4
Douglas	A2	D3	A2	A2	E4	B3	D3	D3	E2	E2	E2	E2	E2	B3	G3
Javier	A2	D3	E4	G3	B3	B3	D3	G3	E2	E2	E2	E4	G3	B3	G3
York	A2	A2	A2	A2	B3	D3	E4	D3	A2	B3	D3	E2	E2	B3	G3
Cody	B3	A2	A2	A2	B3	B3	E4	G3	A2	E2	E2	E2	E2	G3	E4
Binh	A2	D3	A2	A2	E4	D3	D3	D3	E2	E2	E2	E2	E2	G3	G3
Ashtyn	A2	D3	E4	G3	B3	D3	E4	G3	A2	B3	E2	E4	G3	G3	G3
Curtis	B3	D3	A2	A2	B3	D3	D3	D3	A2	E2	D3	E2	E2	G3	G3
Silva	A2	D3	A2	G3	B3	D3	D3	D3	E2	B3	E2	E4	E2	G3	E4
Dinesh	A2	A2	E4	G3	B3	D3	D3	G3	A2	B3	D3	E4	G3	B3	E4
Oguz	A2	D3	E2	A2	B3	D3	D3	G3	A2	B3	D3	E4	G3	G3	E4
Berkay	B3	A2	E4	A2	E4	D3	E4	G3	E2	B3	E2	E4	E2	G3	G3
Eren	B3	D3	E4	G3	E4	D3	E4	D3	E2	E2	D3	E4	G3	B3	G3
Chris	B3	D3	A2	A2	B3	B3	D3	G3	E2	E2	E2	E2	G3	B3	G3
Sean	A2	A2	A2	A2	E4	D3	D3	G3	A2	E2	E2	E2	E2	G3	G3
Jesse	A2	D3	A2	A2	E4	D3	E4	D3	E2	E2	E2	E2	E2	B3	G3

Results of Listening Evaluation

The plot shows how many times a string is chosen as the 'real' one compared to the other.

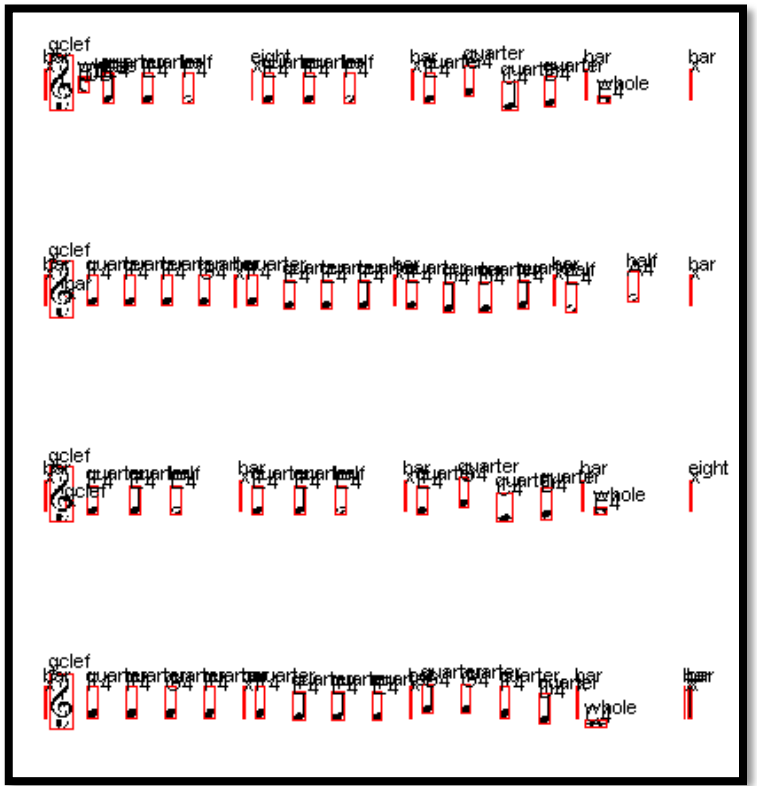


String Note	Number of times chosen
E2	48
A2	42
D3	42
G3	41
B3	36
E4	31

The string 'E2' is the one that has the lowest frequency and the sequence (E2,A2,D3,G3,B3,E4) goes to higher frequencies. The results show that the strings that have lower frequencies sound more realistic.

In addition to this result, the subjects that chose the string that have lower frequencies commented that lower ones have more emphasized and noticeable pluck sound which, made them choose the lower one over the higher one.

Playing a simple musical piece with synthesizing



The last step of the project was to play a simple musical piece with the synthesized sounds when the notes of the piece are given as input. For this purpose, I used an image processing system that was implemented by me. This system takes the music sheet image as input, then identify the notes and gives it as output. This notes are played with the sounds that are synthesized with Karplus-Strong algorithm.

Future Work

For future work, advanced digital signal processing techniques can be applied in addtition to Digital Waveguide Synthesis and Karplus-Strong algorithm to make the synthesized sounds more real. Moreover, although Karplus-Strong algorithm is one of the most popular technique, because it is relatively easy to implement on both hardware and software, different techniques for synthesizing can be researched.

Referances:

[1] K. Karplus and A. Strong, ``Digital synthesis of plucked string and drum timbres,'' Computer Music Journal, vol. 7, no. 2, pp. 43-55, 1983
[2]Cory McKay, "A survey of Physical Modeling Techniques for Synthesizing the classical guitar", Faculty of Music, McGill University
[3] http://en.wikipedia.org/wiki/Karplus–Strong_string_synthesis
[4] http://www.ee.columbia.edu/~ronw/dsp/
[5] M. Karjalainen, V. Välimäki, and Z. Jánosy, ``Towards high-quality sound synthesis of the guitar and string instruments,'' in Proceedings of the 1993 International Computer Music Conference, Tokyo, pp. 56-63, Computer Music Association, Sept. 10-15 1993
[6]] D.A. Jaffe and J.O. Smith, ``Extensions of the Karplus-Strong plucked string algorithm,'' Computer Music Journal, vol. 7, no. 2, pp. 56-69, 1983

Appendix: Matlab Codes

Digital Waveguide

```
function [y] = myDWSoundGenerator(freqOfNote,duration,pickLoc,
pickUpLoc,Fs)
%duration: duration of the note that is going to be played
%pickLoc: relative picking position of the string [0-1]
%pickUpLoc: relative picking up position of the sound.
%Fs: Sampling frequency

%In this method freqOfNote is determined by the string length.
%lambda = waveLength, f = freqOfNote, v = speed of the wave
%v = lambda * f;
%v = Fs samples/second
%-> lambda = Fs / f
%-> stringLength = lambda / 2;
%-> stringLength = (Fs/f) / 2;
stringLength = round((Fs/freqOfNote) / 2);
pickPosition = round(stringLength * pickLoc);
pickupPosition = round(stringLength * pickUpLoc);



%the wave traveling to the right, amplitude values
rightWave = zeros(stringLength,1);
%the wave traveling to the left, amplitude values
leftWave  = zeros(stringLength,1);
%initial conditions of the string(generate triangular wave)
initialHeight = 0.5;

x = 0:stringLength-1;
initialCondition=initialHeight/pickPosition.*x(1:pickPosition+1);
initialCondition(pickPosition+1:stringLength)=-
initialHeight/(stringLength-
pickPosition).*x(pickPosition+1:stringLength)+initialHeight*stringLengt
h/(stringLength-pickPosition+1);
rightWave = initialCondition; leftWave = initialCondition;

dampingFactor = 0.99;
y = zeros(Fs*duration,1);
for i=1:Fs*duration
    y(i) = rightWave(pickupPosition) + leftWave(pickupPosition);

    %Here we are going to simulate travelling of both right and left
waves
    %shift the waves traveling to the right
    rightLast = rightWave(stringLength);
    rightWave(2:stringLength) = rightWave(1:stringLength-1);
    rightWave(1) = (-1) * dampingFactor * leftWave(1);

    leftWave(1:stringLength-1) = leftWave(2:stringLength);
    leftWave(stringLength) = (-1) * dampingFactor * rightLast;

end

%normalize y
y = y - mean(y);
```

```
y = y / max(abs(y));

FigHandle = figure;
set(FigHandle, 'Position', [100, 400, 1050, 400]);
subplot(1,2,1);
plot(y); xlabel('Samples'); ylabel('Amplitude'); title('(G3 (196 Hz) -
In time domain');
subplot(1,2,2);my_fft_plot_abs(y,Fs,freqOfNote*100);

%sound(y,Fs);

end
```

Karplus-Strong

```
function[soundVector] = myKsSoundGenerator(freqOfNote,duration,Fs)

delayLineLength = round(Fs / freqOfNote);
initialRandomWaveTable = rand(delayLineLength,1);

soundVector = zeros(round(duration * Fs) , 1);
soundVector(1:delayLineLength,1) = initialRandomWaveTable;

for i=delayLineLength+1:length(soundVector)
    soundVector(i) = 0.5 * (soundVector(i-delayLineLength+1) +
soundVector(i-delayLineLength));
end

%Normalizing the output to -1 and +1
soundVector = soundVector-mean(soundVector);
soundVector = soundVector/max(abs(soundVector));


FigHandle = figure;
set(FigHandle, 'Position', [100, 400, 1050, 400]);
subplot(1,2,1);
plot(soundVector); xlabel('Samples'); ylabel('Amplitude'); title('(E4
(329.63 Hz) - In time domain');
subplot(1,2,2);my_fft_plot_abs(soundVector,Fs,freqOfNote*40);

%sound(soundVector,Fs);
end
```

Helper Function: Spectrum plotting

```
function [X,f,N_point_DFT] = my_fft_plot_abs(x,Fs,xlimFreq)
%Parameters:
%x : is the function to calculate its fourier
%fs: sampling frequency of x

NFFT = 2^nextpow2(length(x)); %N-point DFT

%calculating fourier transform
X = fft(x,NFFT);
X=fftshift(X);
```

```matlab
%arranging f axis
f = Fs/2 * linspace(-1,1,NFFT);

X = abs(X);

stem(f,abs(X),'marker','none');
xlim([-xlimFreq xlimFreq]);
title('Magnitude');
xlabel('f [Hertz]');

N_point_DFT = NFFT;
```