Physical Modeling Synthesis

ECE 272/472 Audio Signal Processing

Yujia Yan

University of Rochester

Table of contents

1. Introduction

2. Basics of Digital Waveguide Synthesis

3. Fractional Delay Line

4. Deconvolution

Introduction

Physical Modeling Synthesis

The term physical modeling synthesis refers to models that have physical meanings, therefore:

1. It can encode and reproduce sound that is similar to the real sound more efficiently: it requires less storage space.

Physical Modeling Synthesis

The term physical modeling synthesis refers to models that have physical meanings, therefore:

1. It can encode and reproduce sound that is similar to the real sound more efficiently: it requires less storage space.
2. Sound can be modified with interpretable parameters that have a physical meaning.

Physical Modeling Synthesis

The term physical modeling synthesis refers to models that have physical meanings, therefore:

1. It can encode and reproduce sound that is similar to the real sound more efficiently: it requires less storage space.
2. Sound can be modified with interpretable parameters that have a physical meaning.

Example:



Figure 1: Pianoteq, An authentic piano synthesizer that is only 50MB

Different Kinds of Models

A lot of sound synthesis models can be described as *Physical Modeling*:

Different Kinds of Models

A lot of sound synthesis models can be described as *Physical Modeling*:

1. ***Digital Waveguide Synthesis***, where strings, junctions, and other kind of physical components are abstracted into DSP blocks. It models how wave propagates.

Different Kinds of Models

A lot of sound synthesis models can be described as *Physical Modeling*:

1. ***Digital Waveguide Synthesis***, where strings, junctions, and other kind of physical components are abstracted into DSP blocks. It models how wave propagates.
2. ***Modal Synthesis***, where a particular sound is synthesized by adding up decaying oscillators or passing the excitation signal through a set of resonators in parallel. Assuming Rayleigh Damping, the vibration at any point on a surface with a complex geometry can be represented as a linear combination of those oscillators. Therefore it is more often used in Computer Graphics. See [Ren et al., 2013].

Different Kinds of Models

A lot of sound synthesis models can be described as *Physical Modeling*:

1. ***Digital Waveguide Synthesis***, where strings, junctions, and other kind of physical components are abstracted into DSP blocks. It models how wave propagates.
2. ***Modal Synthesis***, where a particular sound is synthesized by adding up decaying oscillators or passing the excitation signal through a set of resonators in parallel. Assuming Rayleigh Damping, the vibration at any point on a surface with a complex geometry can be represented as a linear combination of those oscillators. Therefore it is more often used in Computer Graphics. See [Ren et al., 2013].
3. ***Partial Differential Equation***, where a PDE solver is employed, however, it is computationally expensive.

Different Kinds of Models

A lot of sound synthesis models can be described as *Physical Modeling*:

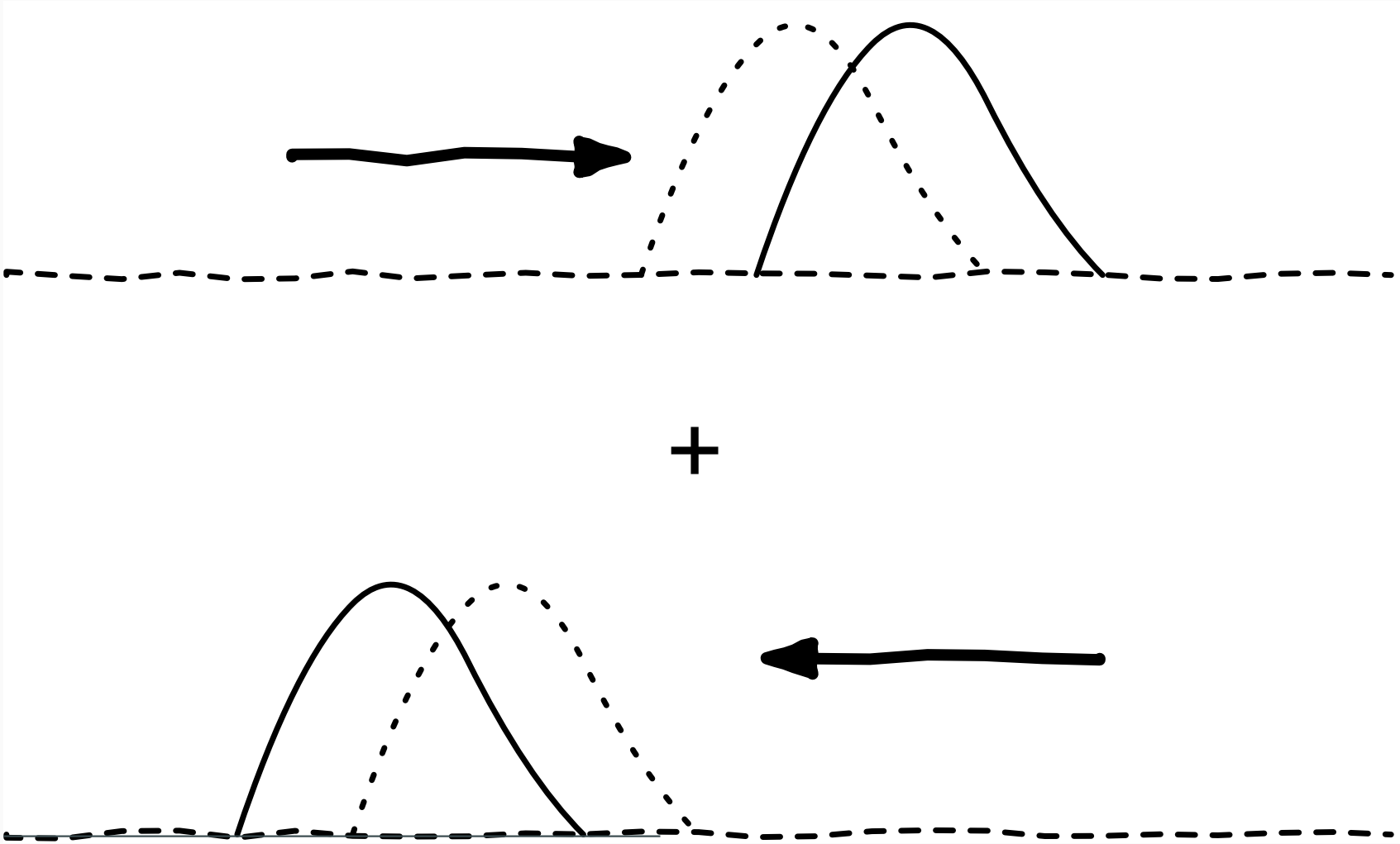
1. ***Digital Waveguide Synthesis***, where strings, junctions, and other kind of physical components are abstracted into DSP blocks. It models how wave propagates.
2. ***Modal Synthesis***, where a particular sound is synthesized by adding up decaying oscillators or passing the excitation signal through a set of resonators in parallel. Assuming Rayleigh Damping, the vibration at any point on a surface with a complex geometry can be represented as a linear combination of those oscillators. Therefore it is more often used in Computer Graphics. See [Ren et al., 2013].
3. ***Partial Differential Equation***, where a PDE solver is employed, however, it is computationally expensive.

In this lecture, we focus on Digital Waveguide Synthesis.

Basics of Digital Waveguide Synthesis

The 1d lossless string

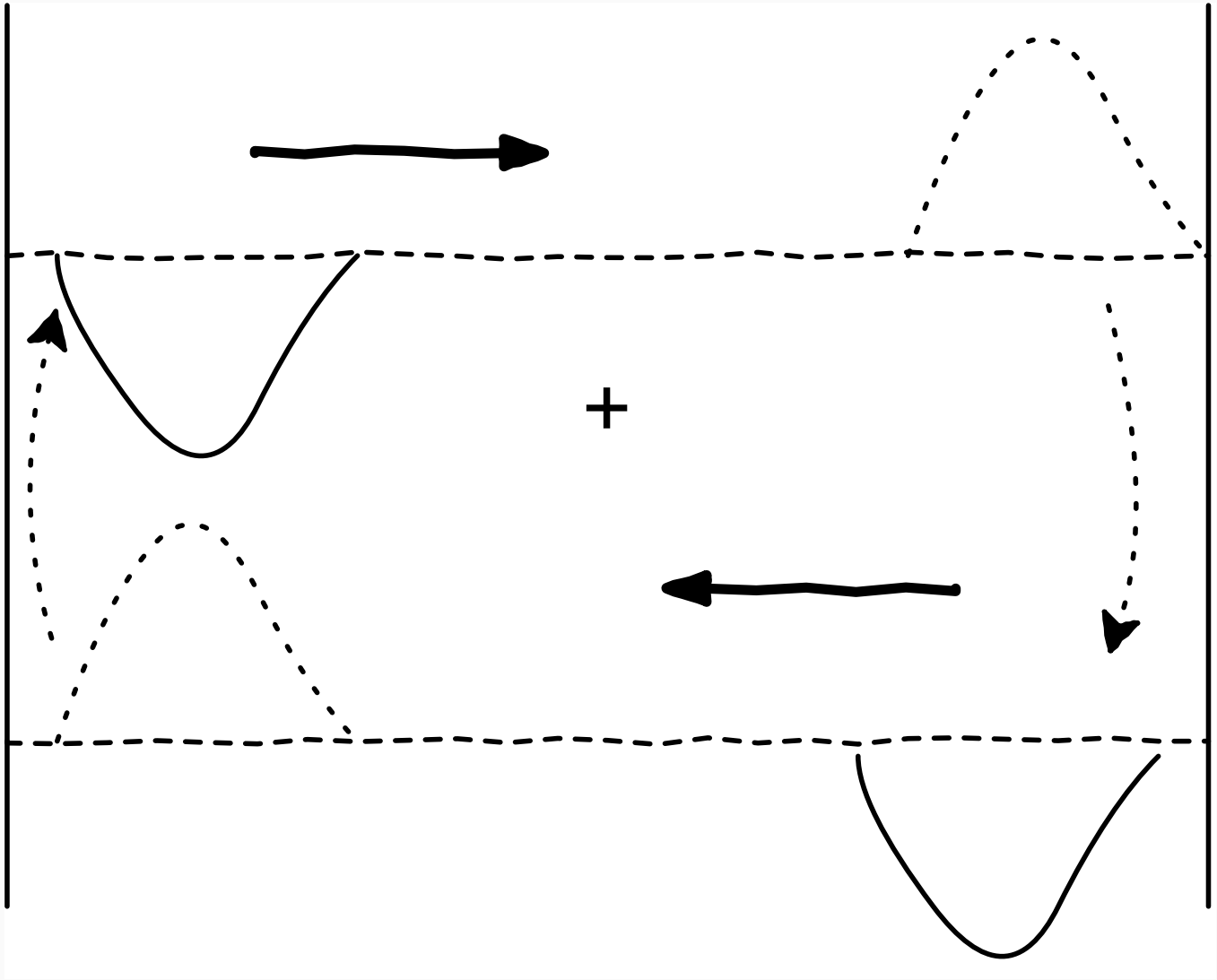
An one dimensional ideal string can be conceptualized as the summation of two waves that propagate to the left and to the right[1].



[1]Also called d'Alembert Solution

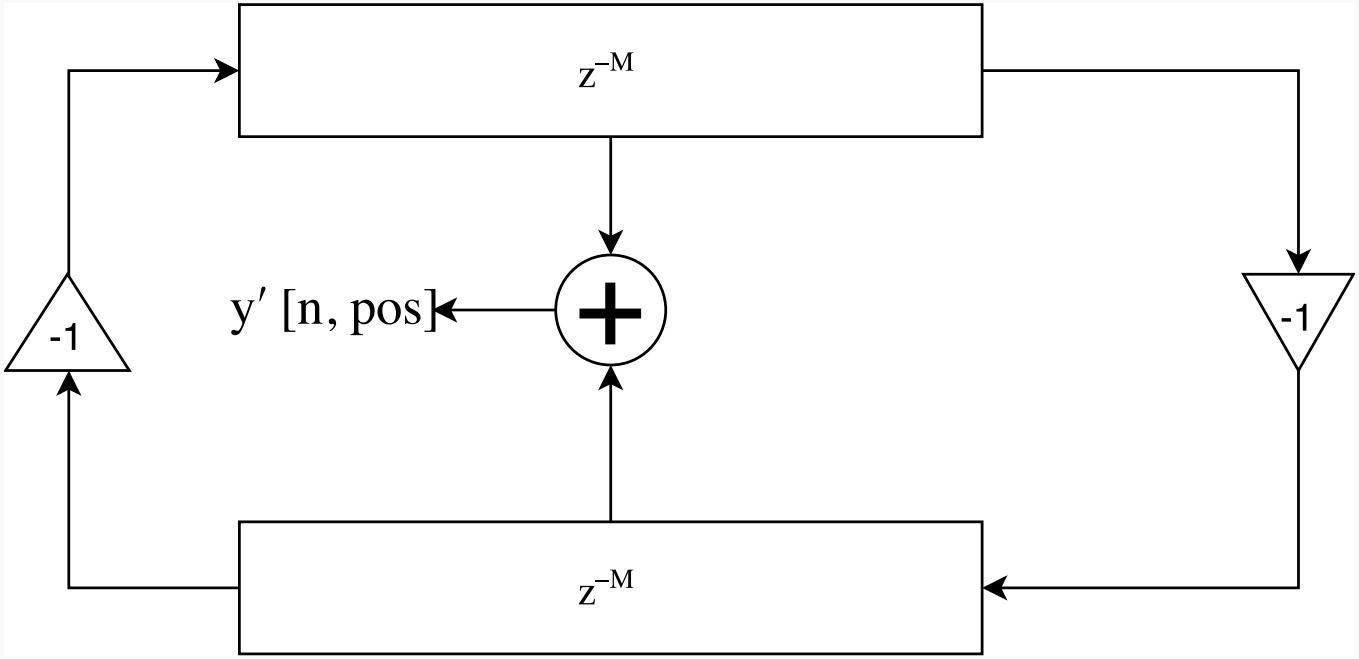
The 1d lossless string

When two terminals of the string are fixed, the wave reflects towards reverse direction with gain -1



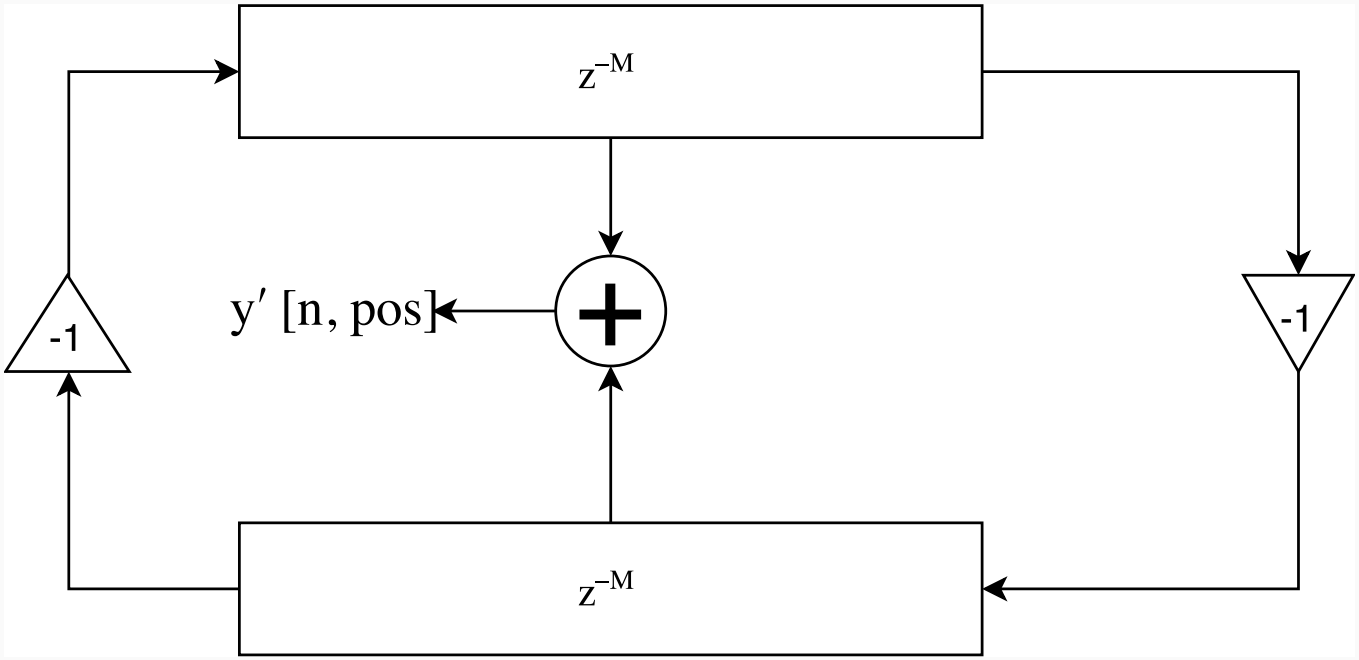
The lossless string: the diagram

We can convert it to a loop with two delay lines, with each representing one direction of propagation.



The lossless string: the diagram

We can convert it to a loop with two delay lines, with each representing one direction of propagation.

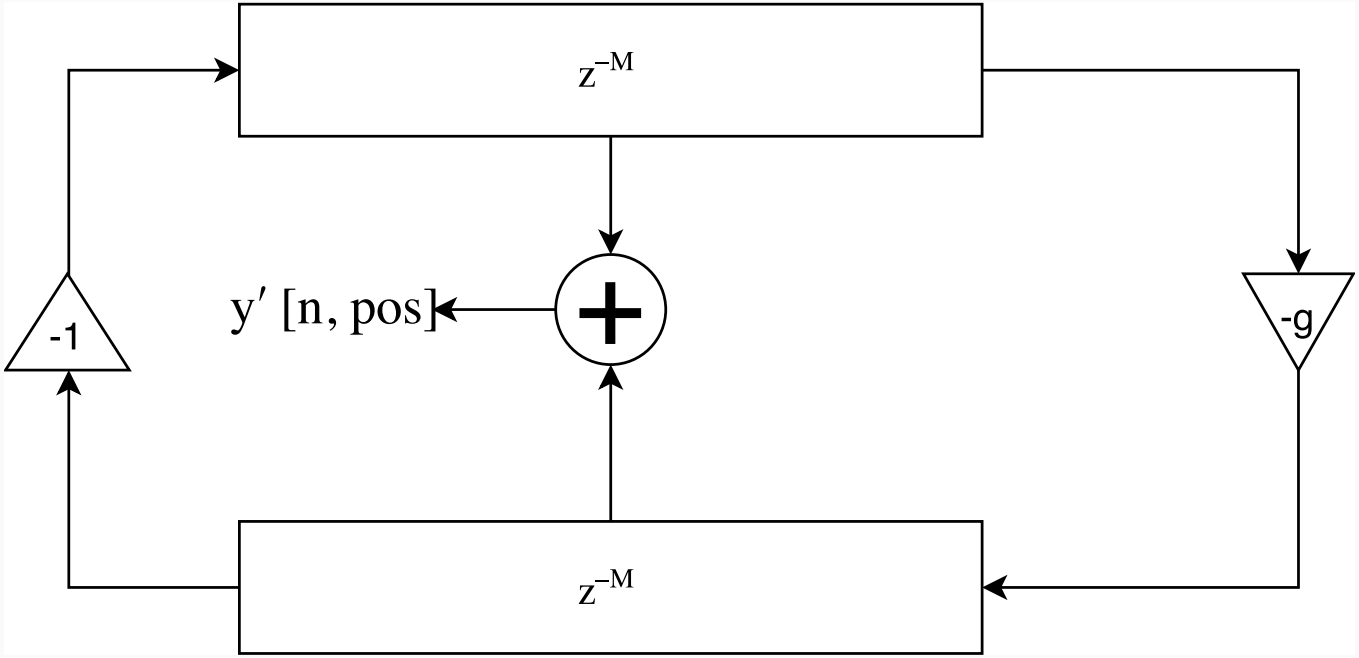


The fundamental frequency is

$$f_0 = \frac{F_s}{2M}$$

Add Energy Losses to the string

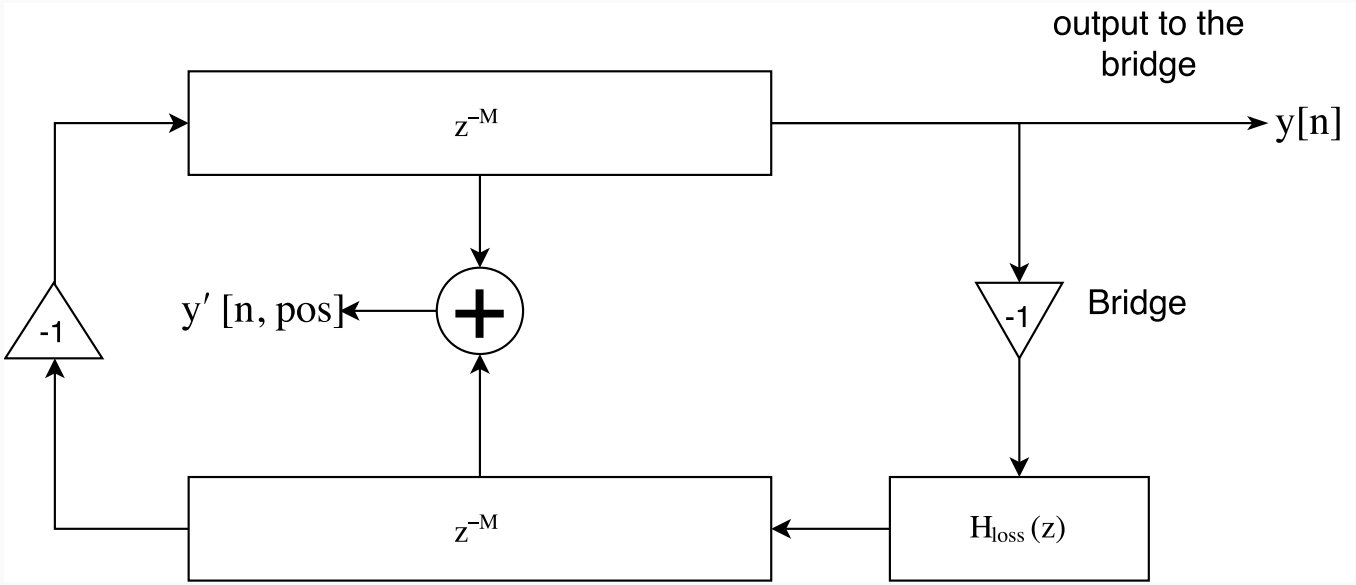
A real string does not vibrate forever, therefore we add a loss gain inside the delay loop:



The Loss (Loop) Filter

In a real situation, higher frequencies usually decay faster.

The decay time is frequency dependent, therefore, we replace the loop gain with a Loss (Loop) Filter



The Loss(Loop) Filter cont.

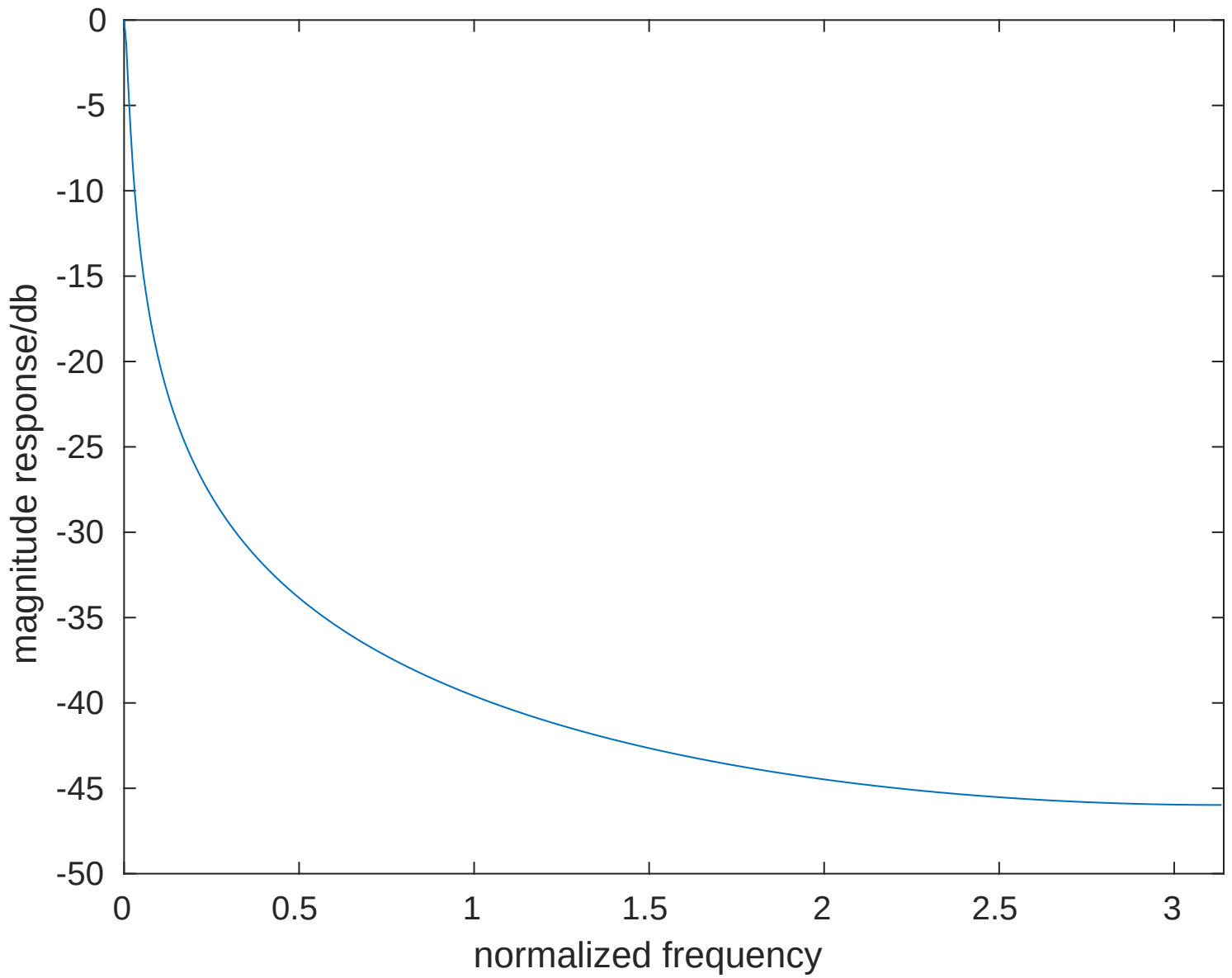
We require that the Loss Filter has a gain that is below 0db for all frequencies.

The most popular choice of the loss (loop) filter in the early literatures is a simple one pole filter:

$$H_{Loss}(z) = g\frac{1+a}{1+az^{-1}}$$

where $0 < g < 1$ and $-1 < a <= 0$

The Loss(Loop) Filter cont.



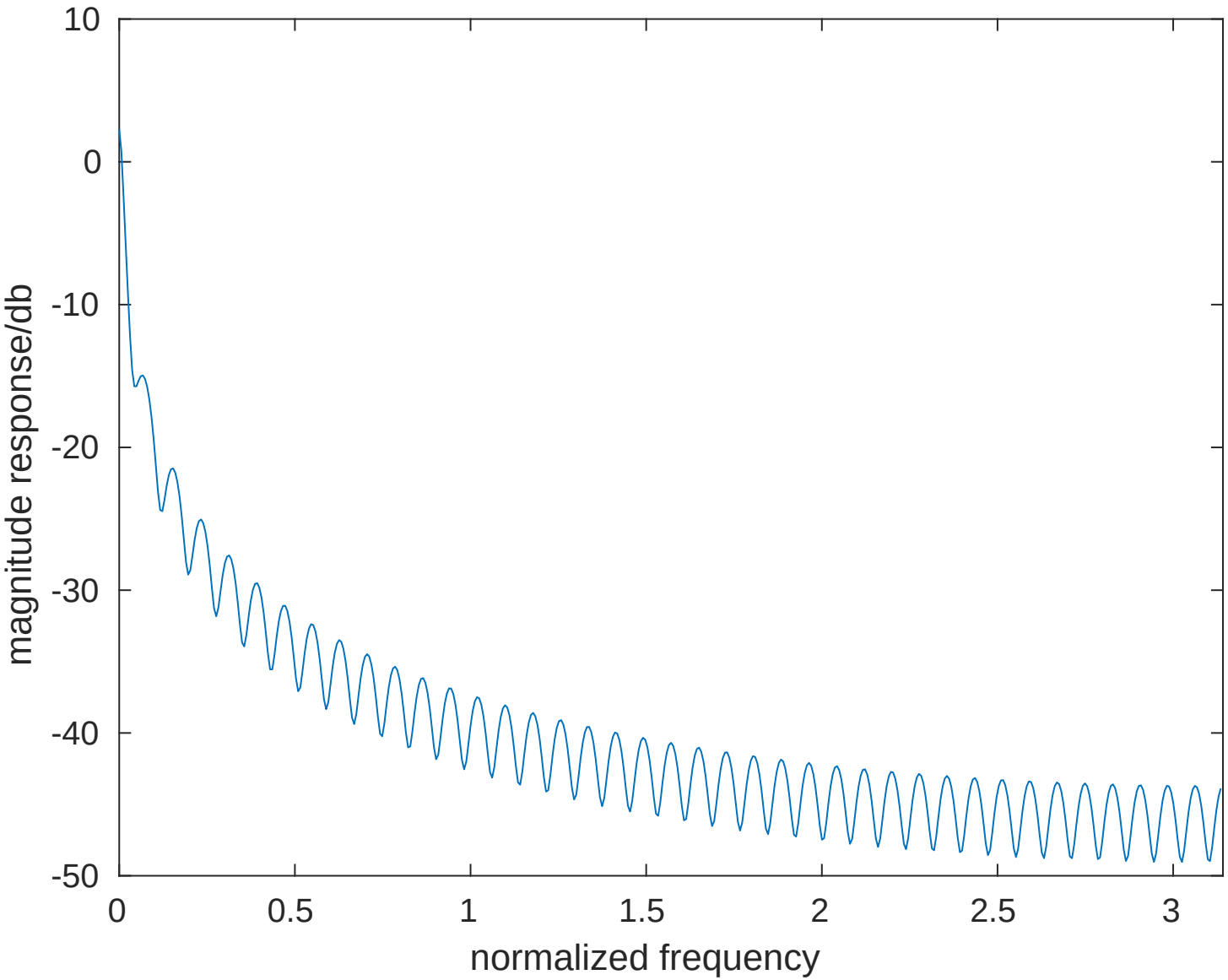
The Loss(Loop) Filter cont.

Some works (e.g. [Välimäki et al., 2004]) added one ripple filter to make the frequency response not that smooth:

$$H_{Loss}(z) = g(1+a)\frac{r+z^{-R}}{1+az^{-1}}$$

where R is the delay line length of the ripple filter, and r is the ripple depth

The Loss(Loop) Filter cont.



Stiffness and Inharmonicity

- Stiffness causes frequency dependent variations of propagation speed, producing stretched harmonics:

$$f_n = n(f_0\sqrt{1+Bn^2})$$

where $B = \frac{\pi^3 r^4 Y}{16TL^2}$, r is the radius of the string, Y is the Young's Modulus, T is the tension, and L is the string length

Stiffness and Inharmonicity

- Stiffness causes frequency dependent variations of propagation speed, producing stretched harmonics:

$$f_n = n(f_0\sqrt{1+Bn^2})$$

  where $B = \frac{\pi^3 r^4 Y}{16TL^2}$, r is the radius of the string, Y is the Young's Modulus, T is the tension, and L is the string length
- To simulate stretched harmonics, we insert one all-pass filter in the filtered delay loop to create frequency dependent delays

Stiffness and Inharmonicity

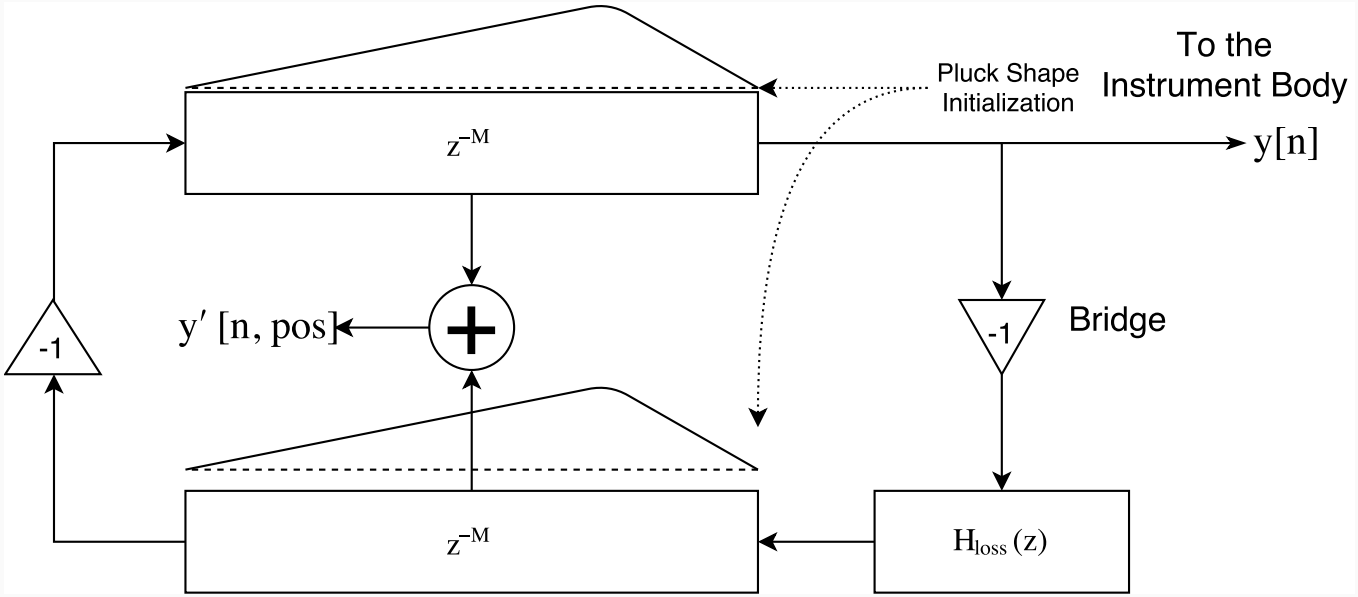
- Stiffness causes frequency dependent variations of propagation speed, producing stretched harmonics:

$$f_n = n(f_0\sqrt{1+Bn^2})$$

  where $B = \frac{\pi^3 r^4 Y}{16TL^2}$, r is the radius of the string, Y is the Young's Modulus, T is the tension, and L is the string length
- To simulate stretched harmonics, we insert one all-pass filter in the filtered delay loop to create frequency dependent delays
- See [Abel et al., 2010] for more details on how to design all-pass filters for this purpose.

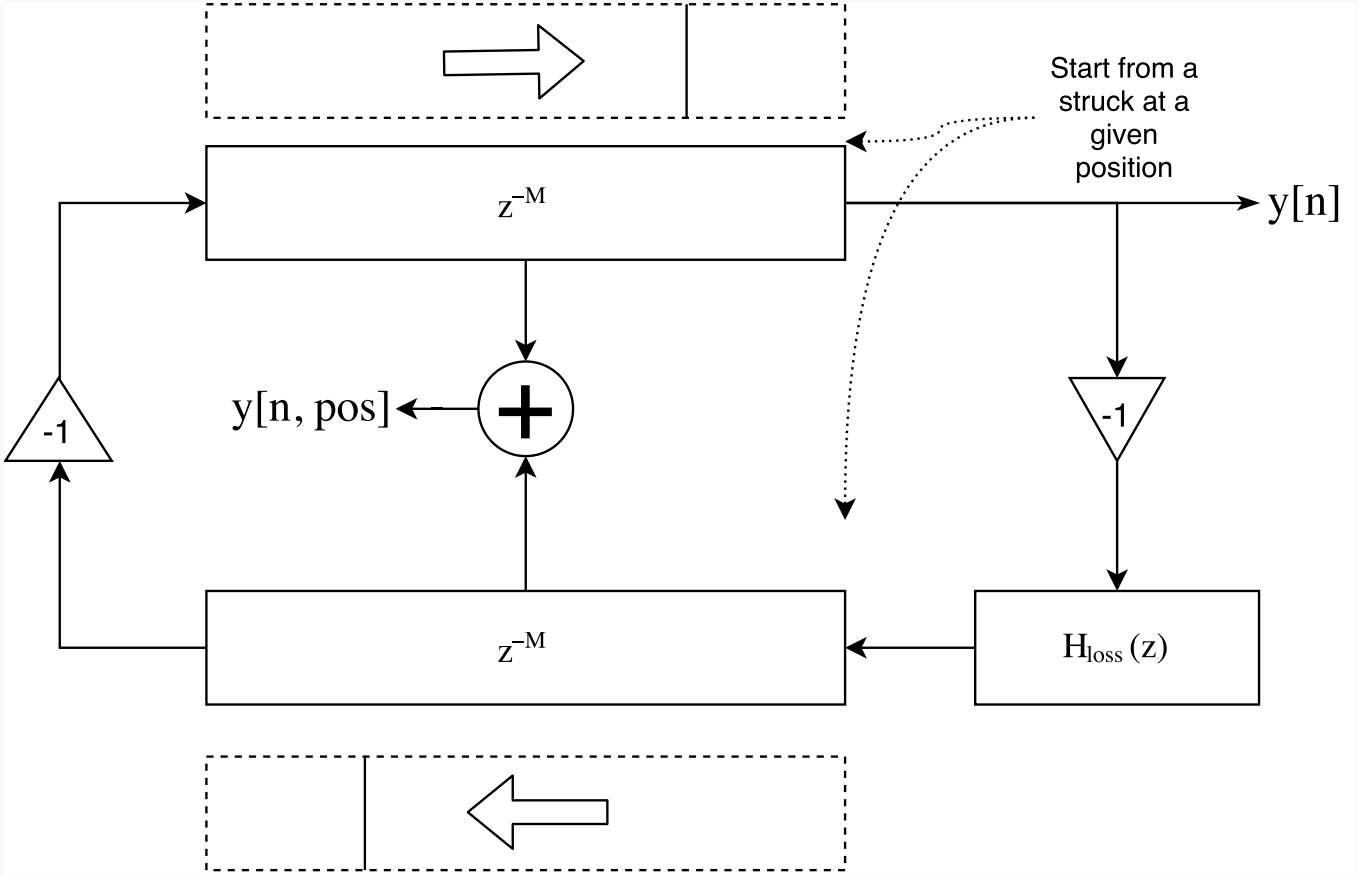
A simple plucked string

Assuming displacement wave, we initialize the delay line with the plucking shape to get a plucked string sound



A simple struck string

Assuming velocity wave, we initialize the delay line with the impulse at a given position to get a struck string sound, an integrator at the output is enough to produce the displacement signal



Commuted Synthesis

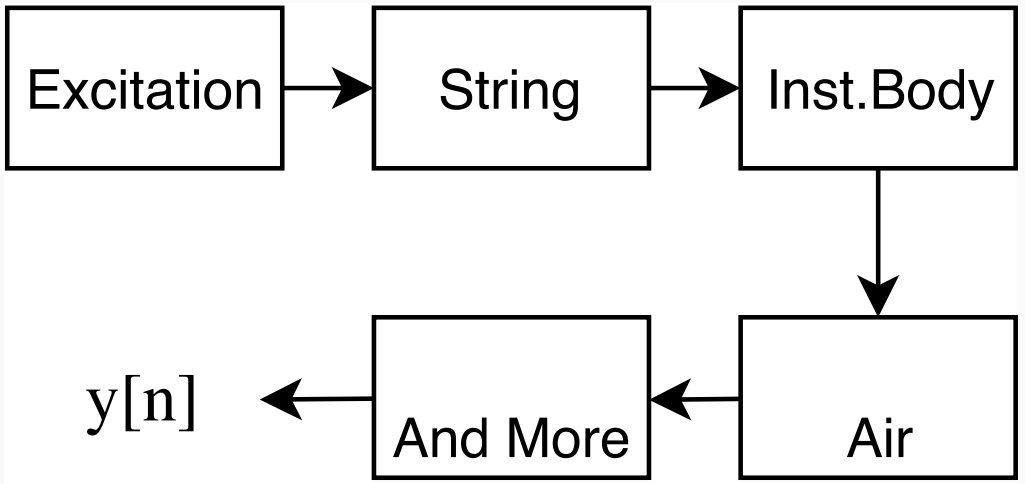
- The initialization of the delay line for the plucked string is also an impulse if we differentiate it twice, and compensate that at the output terminal.

Commuted Synthesis

- The initialization of the delay line for the plucked string is also an impulse if we differentiate it twice, and compensate that at the output terminal.
- Therefore, we can separate the excitation from the string and make the string takes an external input at a given position:

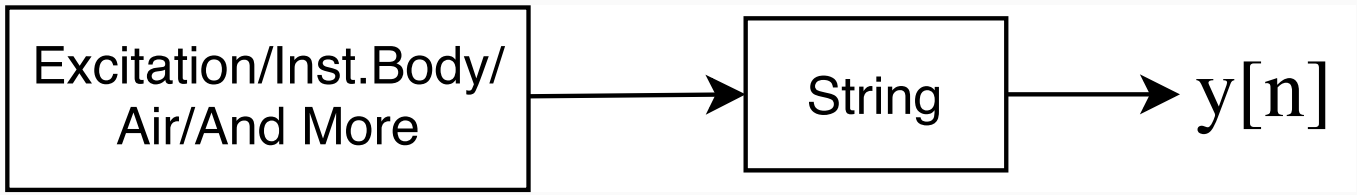
Commuted Synthesis

- The initialization of the delay line for the plucked string is also an impulse if we differentiate it twice, and compensate that at the output terminal.
- Therefore, we can separate the excitation from the string and make the string takes an external input at a given position:



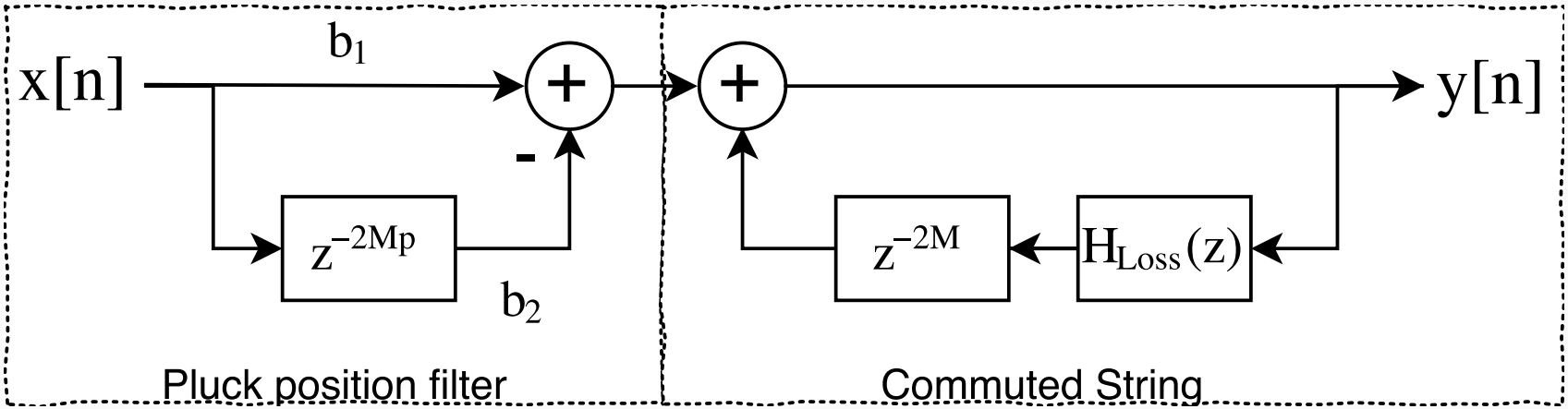
Commuted Synthesis cont.

Then by commutativity of linear time invariant system, we can commute and merge those blocks that you do not want to model separately:



Commuted Synthesis cont.

With some rearrangement, we can obtain the commuted string plus a pluck position filter that is equivalent to the string aforementioned



Where $0 < p < 1$ is the pluck position and $2M$ is the total length of the delay line.

The pluck (excite) position filter is a feedforward comb filter that creates excite position dependent response.

It simulates the fact that if you excite the string at $\frac{1}{K}$ position, then k-th harmonics and its multiples will be missing.

Tuning the parameters

So far, we have two kinds of parameters we want to get from data if we want to generate realistic sound:

1. Filter Parameters
2. Excitation Signal

Tuning the parameters

So far, we have two kinds of parameters we want to get from data if we want to generate realistic sound:

1. Filter Parameters
2. Excitation Signal

Traditionally, it is more of an art than a science

1. Tune the filter Parameters empirically: some can be automatically estimated and some have to be manually tuned, e.g., [Välimäki et al., 2004].
2. Extract excitation signals according to the filter you obtained. It is basically a deconvolution problem.
3. perform these steps many times

Tuning the parameters

So far, we have two kinds of parameters we want to get from data if we want to generate realistic sound:

1. Filter Parameters
2. Excitation Signal

Traditionally, it is more of an art than a science

1. Tune the filter Parameters empirically: some can be automatically estimated and some have to be manually tuned, e.g., [Välimäki et al., 2004].
2. Extract excitation signals according to the filter you obtained. It is basically a deconvolution problem.
3. perform these steps many times

See [Riionheimo and Välimäki, 2003] for an example of how to use global optimization (e.g., Genetic Algorithm) and perceptual similarity to estimate all parameters jointly.

Fractional Delay Line

Fractional Delay Line

The delay line length is usually a non-integer number for simulating the string length or some audio effect, e.g, vibrato.

Fractional Delay Line

The delay line length is usually a non-integer number for simulating the string length or some audio effect, e.g, vibrato.

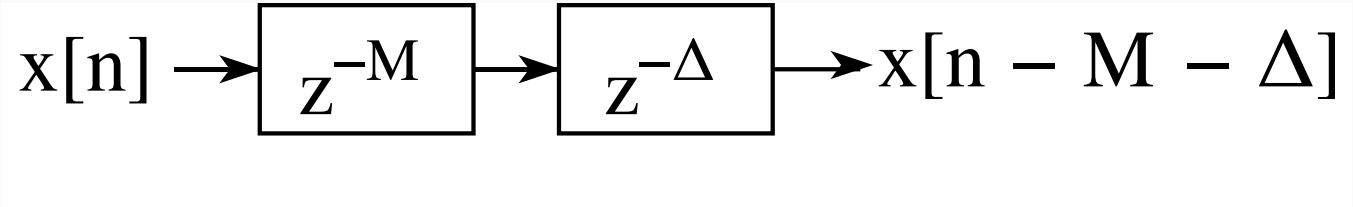
This is usually done by splitting the fractional delay line length into its integer M and fractional part Δ.

Fractional Delay Line

The delay line length is usually a non-integer number for simulating the string length or some audio effect, e.g, vibrato.

This is usually done by splitting the fractional delay line length into its integer M and fractional part Δ.

The fractional part Δ is not necessarily smaller than one: it depends on the actual filter structure



The integer delay line: Circular Buffer

A delay line with integer length seems trivial, however, we require two desired property for a real-time application:

- Constant time read and write
- Constant space

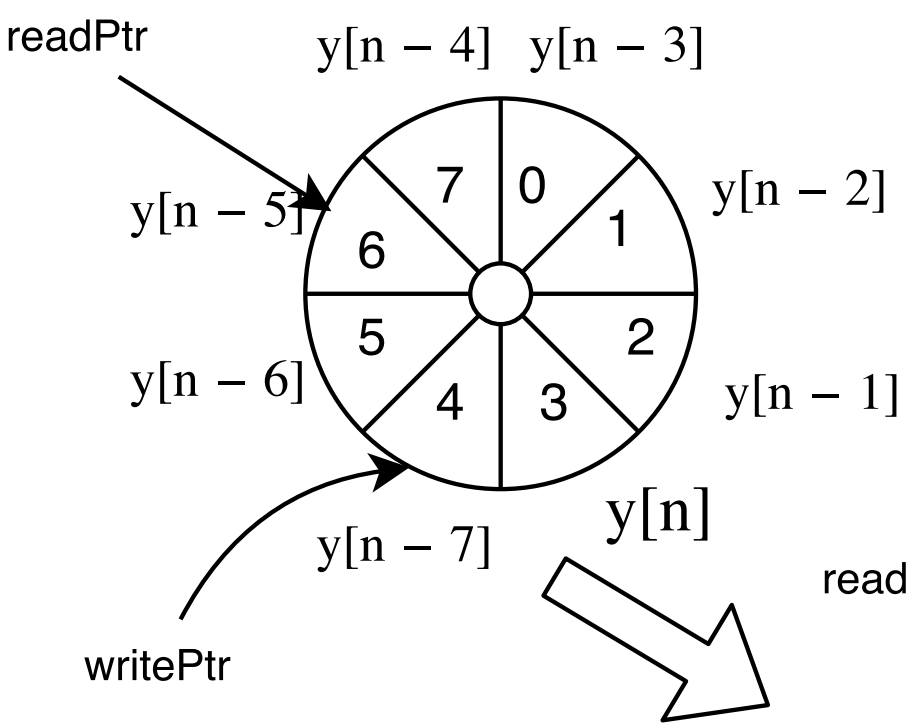
The integer delay line: Circular Buffer

A delay line with integer length seems trivial, however, we require two desired property for a real-time application:

- Constant time read and write
- Constant space

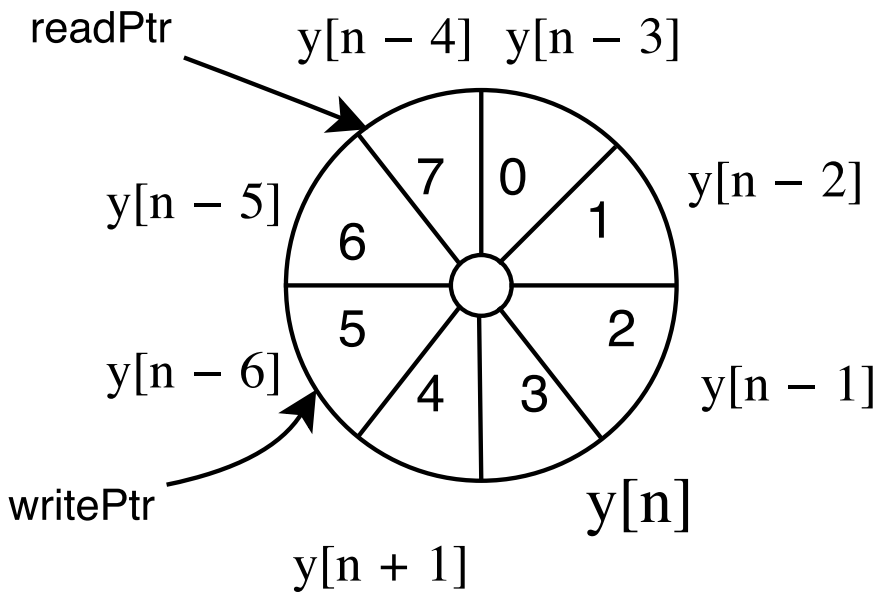
A circular (Ring) Buffer!

The integer delay line: Circular Buffer Cont.



When y[n + 1] comes in
1. Fetch the position readPtr is looking at : y[n – 5]
2. write y[n + 1]
3. readPtr = (readPtr+1)%8
writePtr = (writePtr+1)%8

What's the delay in this case?
(writePtr-readPtr+ 8)%8



The general circular buffer

- **Initialization**: writePtr = (readPtr+delayLineLength)%circularBufferLength
- **Read**: fetch buffer[readPtr] and increase the readPtr as readPtr = (readPtr+1)%circularBufferLength
- **Write**: write data to buffer[writePtr] and increase the writePtr as writePtr = (writePtr+1)%circularBufferLength

The fractional delay filter: Linear Interpolation

Given the fractional part delay length Δ, the simplest design of the fractional delay filter is linear interpolation:

$$\begin{aligned} y[n] &= (1-\Delta)x[n] + (\Delta)x[n-1] \\ &= x[n] + \Delta(x[n-1] - x[n]) \end{aligned}$$

Where $\Delta \in [0, 1]$

- Exact Delay at DC
- For the magnitude response it behaves like a low pass filter

The fractional delay filter: Lagrange Interpolation in general

For the N-th order Lagrange interpolation on a set of uniformly sampled points, it can be written as an $N+1$ length FIR filter:

$$h_\Delta(n) = \prod_{k=0:k\neq n} \frac{\Delta - k}{n-k}, n = 0, 1, 2, \ldots, N$$

The most desired delay is around $\Delta = \frac{N}{2}$, near the center of the impulse response.

The fractional delay filter: Lagrange Interpolation in general

For the N-th order Lagrange interpolation on a set of uniformly sampled points, it can be written as an $N+1$ length FIR filter:

$$h_\Delta(n) = \prod_{k=0:k\neq n} \frac{\Delta - k}{n-k}, n = 0, 1, 2, \ldots, N$$

The most desired delay is around $\Delta = \frac{N}{2}$, near the center of the impulse response.

Adjustable high order Lagrange Interpolation can be efficiently implemented with a Farrow Structure (beyond the scope of this lecture)

The fractional delay filter: first order all-pass interpolator

Unlike Lagrange interpolators, which behaves like a low-pass filter, we can use an all-pass filter:

First Order Allpass Interpolation Filter:

$$y[n] = \eta(x[n] - y[n-1]) + x[n-1]$$

where

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

In practice: $\Delta \in [0.3, 1.3]$

The fractional delay filter: Thiran all-pass interpolator in general

In general: The allpass filter:

$$H(z) = \frac{a_N + a_{N-1}z^{-1} + \ldots + a_1 z^{-N+1} + z^{-N}}{1 + a_1 z^{-1} + \ldots + a_{N-1}z^{-N+1} + a_N z^{-N}}$$

The Thiran Allpass Interpolators:

$$a_k = (-1)^k \binom{N}{k} \prod_{n=0}^{N} \frac{\Delta - N + n}{\Delta - N + k + n}, k = 0, 1, 2, \ldots, N$$

where

$$\binom{N}{K} = \frac{N!}{k!(N-k)!}$$

is the binomial coefficient

In practice, $\Delta \approx N$, the order of the filter to ensure stability

The fractional delay filter: windowed sinc Interpolator

The sinc function gives an ideal interpolation for bandlimited signal

Given $\Delta = N/2 + \delta$

This filter can be designed by windowing the ideal sinc impulse response:

$$h[i] = w[i]sinc(i - \Delta)$$

where w is the window function

It can give a very flat phase delay.

It may require many points to obtain a good frequency response.

Deconvolution

Problem Formulation

Assume our model gives:

$$h * x = y$$

where h is the impulse response of our system, y is observed data(noisy), x is the unknown excitation

We want to give an estimation of x

Deconvolution via Polynomial Long Division

For two polynomial h and g, each represented by their coefficients
$h = h_1 + h_2 x + \ldots + h_N x^{N-1}$

Deconvolution via Polynomial Long Division

For two polynomial h and g, each represented by their coefficients $h = h_1 + h_2 x + \ldots + h_N x^{N-1}$

The coefficients of their multiplication f is given by:

$$f = h * g$$

Deconvolution via Polynomial Long Division

For two polynomial h and g, each represented by their coefficients $h = h_1 + h_2 x + \ldots + h_N x^{N-1}$

The coefficients of their multiplication f is given by:

$$f = h * g$$

We can use polynomial long division to divide f by h to obtain $\hat{g}$

Polynomial long division solves the problem:

$$f = h * \hat{g} + R$$

where R is a polynomial with a degree lower than h

Deconvolution via Polynomial Long Division Cont.

$$
\begin{array}{r}
x - 10 \\
\hline
x^2 - 2x + 1 \,\big)\, x^3 - 12x^2 + 0x - 42 \\
\underline{x^3 - 2x^2 + x} \\
-10x^2 - 1x - 42 \\
\underline{-10x^2 + 20x - 10} \\
-21x - 32
\end{array}
$$

Figure 2: Example of Polynomial Long division, taken from `https://en.wikipedia.org/wiki/Polynomial_long_division`

This is what Matlab's buildin *deconv* function is doing.

The MSE Deconvolution

- The Long division algorithm takes only the degree of polynomial into consideration.
- Magnitude of the error is ignored.
- Therefore we need to reformulate deconvolution by minimizing squared error.

See my note for more details:
`https://www.evernote.com/shard/s236/sh/ac70a53a-415c-41eb-b929-7e3f6f7b13bd/058ffd37658c88dae88c6b0aad5bd7e5`

References i


Abel, J. S., Valimaki, V., and Smith, J. O. (2010).
Robust, efficient design of allpass filters for dispersive string sound synthesis.
IEEE Signal Processing Letters, 17(4):406–409.


Ren, Z., Yeh, H., and Lin, M. C. (2013).
Example-guided physically based modal sound synthesis.
ACM Transactions on Graphics (TOG), 32(1):1.


Riionheimo, J. and Välimäki, V. (2003).
Parameter estimation of a plucked string synthesis model using a genetic algorithm with perceptual fitness calculation.
EURASIP Journal on Applied Signal Processing, 2003:791–805.

References ii


Scavone, G. P.
Mumt 618: Computational modeling of musical acoustic systems.
https://www.music.mcgill.ca/~gary/618/.


Smith, J. O. (2010).
Physical audio signal processing: For virtual musical instruments and audio effects.
W3K Publishing.


Välimäki, V., Penttinen, H., Knif, J., Laurson, M., and Erkut, C. (2004).
Sound synthesis of the harpsichord using a computationally efficient physical model.
EURASIP Journal on Advances in Signal Processing, 2004(7):860718.