ECS512U Sound Design • 2014

Lab Assignment 1: Pluckable String

Handed out: Friday, 7 February 2014

Due: Tuesday, 18 February 2014

The purpose of this assignment is to help you gain confidence in using the PureData patching environment. You will build a realistic model of a plucked string using objects and techniques that have been covered in previous labs and lectures. Your submission should consist of PureData patches for each of the steps, as well as a short report describing your procedure and results (more details below).

Required Software:

• Pd-extended, available free for all platforms from http://puredata.info/downloads

Important note:

When working with headphones, don't wear the headphones while patching in PureData as you might be subjected to very loud noises, especially when working with feedback delays, which can easily become unstable. Never work with loud volume levels unless you are certain that your audio chain is stable. To turn off the audio in PureData use [Ctrl] + [.] on Windows or Linux, and [Cmd] + [.] on OS X.

Step 1: *Pass a simple noise burst generator through a delay line*

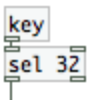
a) You will need to start with a `[noise~]` object which is passed through a `[*~]` object. You also need another signal generator to control the amplitude of the noise signal over time. Use the `[line~]` object together with a message box to generate a burst of noise that decays over the period of 2 milliseconds. Check the help file of `[line~]` if you have forgotten how to make it generate an output signal.

Question: How would you change the amplitude of the noise burst without using another multiplication object?

b) Use the `[delwrite~]` and `[vd~]` objects to delay the signal by 100 milliseconds. Note that we are using `[vd~]` rather than `[delread~]` because we will be varying the delay time at audio rate and need to avoid unwanted artifacts. Give the objects a unique name as a first argument and give the `[delwrite~]` object a second argument specifying a maximum delay time in milliseconds (e.g. 1000). Don't forget to connect the output of your signal chain to the audio output (`[dac~]` object).

Question: You should only be able to hear the sound 100 milliseconds after generating the noise burst. What would you need to do to hear both the original and the delayed signal?

(Optional): Connect the following objects to the message box attached to the line object in order to trigger noise bursts with the spacebar:



Step 2: *Use send and receive objects to introduce feedback into the delay line*
Create a `[send~]` and a `[receive~]` object and give them a unique name (e.g. *feedback1*). Connect them in such a way that the delayed noise burst is 'fed back' into the delay system (i.e. routed back into the same `[delwrite~]` object). Remember to multiply the feedback signal by a number **smaller than 1** and greater than 0. Try setting the delay time to a smaller number (e.g. 5 milliseconds).
Question: Why does the feedback signal need to be multiplied by a number smaller than 1? What would happen if we were to use a number greater than 1?

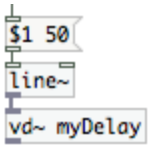
Step 3: *Introduce loss of high frequencies and a variable frequency parameter*

a) When setting the delay time to smaller numbers we start to hear audible frequencies resembling the sound of a metallic string. The decay of the 'string' can be set by varying the feedback parameter (i.e. the decimal number between 0 and 1 by which the feedback signal is multiplied). Use a horizontal slider (`[hslider]`) to control the string's decay.

b) Bearing in mind that frequency is equal to the inverse of the wavelength expressed in seconds:

$$f = \frac{1}{T}$$

Add a parameter that allows you to set the frequency of the string in Hz. Use the following chain of objects to smooth between values at audio rate (replacing 'myDelay' with the name you have given your `[vd~]` and `[delwrite~]` objects):



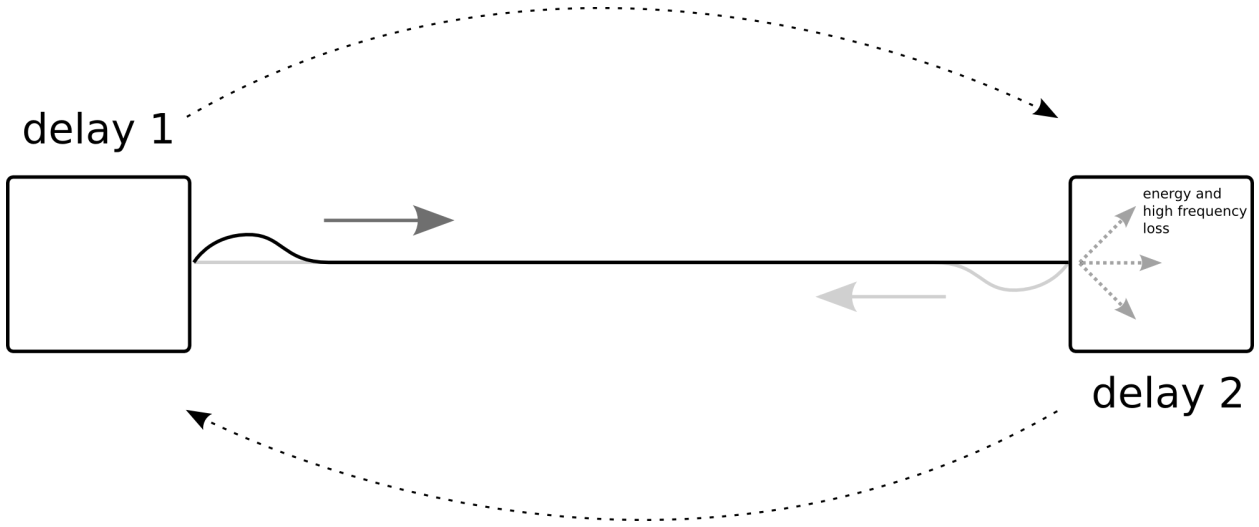
c) When plucking a real string high frequencies tend to decay faster than low frequencies. Introduce a lowpass filter in your signal chain (`[lop~]`) that gradually filters out the high frequencies of the feedback signal. Attach a horizontal slider or a numberbox to add variable

control of the centre frequency (an ideal range is between 500Hz and 15000Hz).

Question: Why do we hear a gradual decay of high frequencies rather than only hearing sounds below the specified centre frequency?

Step 4: *Use two feedback delay systems to simulate the way sound waves travel along a string*

In its current state, the model resembles an early synthesis technique known commonly as Karplus-Strong synthesis (named after the people who invented it in 1983). It was known for producing a sound uncannily similar to that of a guitar string. In this step you will extend the model to produce a more accurate simulation of a plucked string's behaviour. Consider the following diagram:



Karplus-Strong synthesis works so well because the delay line simulates the way energy travels from one end of the string to the other. However, using one delay line is like assuming that energy only travels in one direction along the string. To create a more accurate simulation we need two delay lines: one for each end of the string. Once energy has travelled from one end to the other (i.e. arrives at the output of a `[vd~]` object) it needs to travel back to the other end (i.e. into another (`[delwrite~]`) object).

a) Extend your model to enable bidirectional movement. Remember to send the excitation signal (the noise burst) to *both* ends of the string.

Question: why does the noise burst need to be sent to both ends of the string?

b) Two delay lines don't make much of a difference to the sound, unless we were to change the position at which we pluck the string. Implement a variable pluck position into your string model.

Use a horizontal slider to vary the plucking position. Note that you can change the range of the horizontal slider by right-clicking it and selecting 'properties'.
Hint 1: you will need another two `[delwrite~]` objects.
Hint 2: you know already how many milliseconds it takes for energy to travel from one end of the string to the other.
Hint 3: look at the helpfile for the `[swap]` object - it may come in handy!

Question: Considering that plucking a string generates a harmonically rich signal, how does varying the plucking position alter the sound?

Your final model should be controllable using the following four parameters:

- Frequency
- Energy preservation (or string decay)
- Frequency preservation (i.e. centre frequency of low pass filter)
- Pluck position

Bonus step (extra credit): *Add a variable pick-up position*
We are currently listening to the audio as it arrives at the very end of the string. Guitar pick-ups typically capture sound from a specific point along the string, usually close to the plucking position. Modify the model such that you can vary the point at which the sound is captured. Think about what you needed to do in the last step to achieve a variable plucking position. You will need to do something similar at the output of your two `[vd~]` objects. Parameterise the model so that you can vary the pick-up position.

Your submission should consist of the following:
• Your PureData patches for each step
• Short report (PDF, 1-2 sides of A4) describing your procedure and results, and briefly addressing the questions above
• *Make a .zip or .tar.gz file of your code and report and submit it via Intranet*