Patrikas Balsys

Mini project: Plucked String Synthesis

Synthesizing analog sounds is not always easy. In order to achieve synthesis of a sound similar to a plucked string, the Karplus-Strong algorithm is used.

The KS algorithm consists of a feedback comb filter, a feedforward lowpass filter, and an allpass filter.

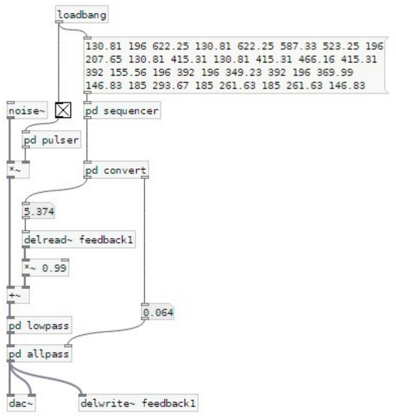
When a string is plucked, the perceived pitch is the highest (first) comb point, but the sound consists of more frequencies. The comb filter helps us achieve this effect by highlighting frequencies in equal intervals.

However, a comb filter alone will keep doing this to infinity, and this sounds unnatural. To fix this, a lowpass filter stops the filter from highlighting very high frequencies.

Lastly, an allpass filter shifts the phase, adding a delay - since most digital filters work on integer samples, this helps us regulate the exact pitch. (Note: Pure Data system can deal with fractional delays, so an allpass filter is not strictly necessary - however, it is implemented here to display how the full system works)

The system shall be implemented with Pure Data. The system also contains a sequencer to create a simple melody.

Overview



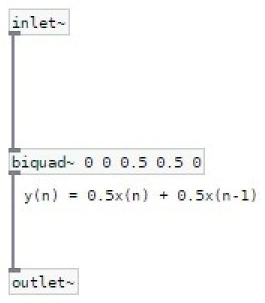
Here we can see the full KS synthesizer, plus an additional sequencer. I will go down into the components in further sections.

When the system is loaded, a sequence of frequencies is initiated into the sequencer. This is to create a simple melody. The `pd pulser` is also started, which helps to create bursts of white noise every half second.

`pd sequencer` then outputs these frequencies one by one every half second. The frequency gets passed onto `pd convert`, which calculates the variables needed for the comb and allpass filters. The first outlet outputs the delay D needed for the comb filter. The second outlet outputs the coefficient b needed for the allpass filter to add a small additional delay.

`delread` and `delwrite` help to make the comb filter, that is seen here. The comb filter uses the delay D provided by `pd convert` and sums up the periodic white noise pulses with the sum of the filtered signal D miliseconds beforehand. This sum is constantly passed through a `pd lowpass` filter and a `pd allpass` filter. Lastly, all of this is output to the speakers.

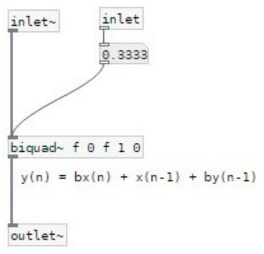
Lowpass



$$y_n = 0.5x_n + 0.5x_{n-1}$$

The lowpass filter is rather straightforward. It takes the current value, sums it up with the previous value and averages them out. This smoothens out the signal, making it harder for high frequencies that fluctuate a lot to pass through.

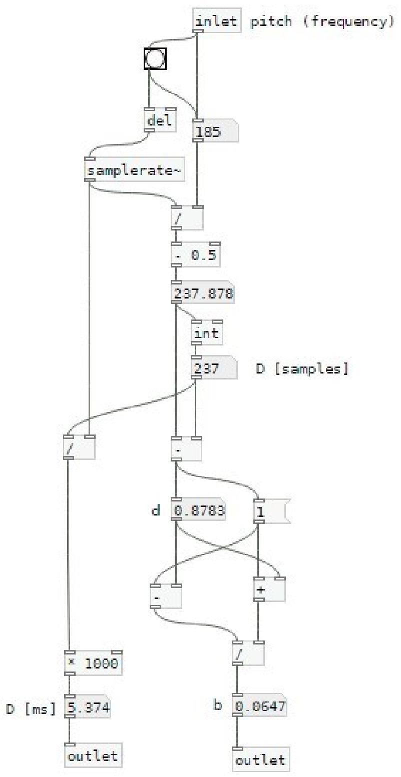
Allpass



$$y_n = bx_n + x_{n-1} + by_{n-1}$$

The allpass filter does not mess with the frequency. Instead, it alters the phase of the signal, allowing us to add a small delay. This helps achieve fractional delays on systems where that is not possible.

Convert



The `pd convert` component calculates all needed values for our filters. It takes in the pitch frequency through the inlet. It uses that together with sampling frequency to calculate a few different variables. The first outlet calculates delay D needed for the comb fitler. In general, each step should have a delay of N samples:

$$N = D + 0.5 + d = \frac{f_s}{f_0}$$

- D - comb filter delay
- 0.5 - lowpass filter delay
- d - allpass filter delay
- f_s - sampling frequency
- f_0 - pitch frequency

The first outlet outputs D, converted to miliseconds, as this is required for our comb filter. The system calculations are equal to the formula:

$$D[samples] = floor(\frac{f_s}{f_0} - 0.5)$$

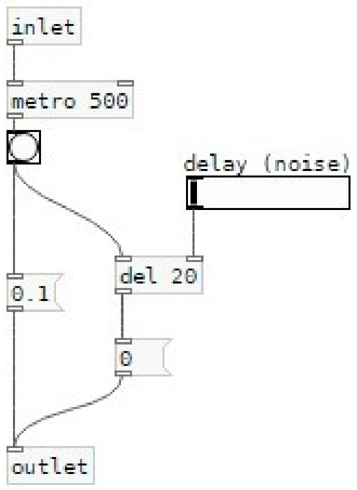
$$D[ms] = \frac{D[samples]}{f_s} 1000$$

The second outlet outputs b, the coefficient for the allpass filter. In order to calculate this, we first calculate the allpass filter delay d:

$$d = (\frac{f_s}{f_0} - 0.5) - D[samples]$$

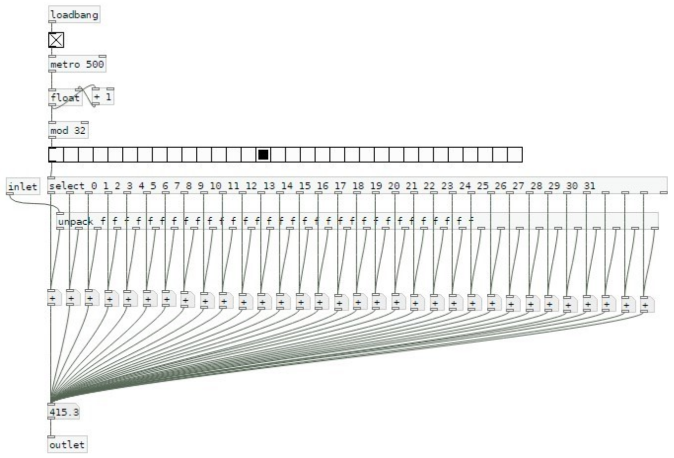
$$b = \frac{1-d}{1+d}$$

Pulser



The `pd pulser` component just outputs a value of 0.1 every half second, quickly followed up by a 0. This, combined with white noise, creates short bursts of noise which form the basis for our sound.

Sequencer


loadbang
metro 500
float
+ 1
mod 32
inlet
select 0 1 2 3 4 5 6 7 8 9 10 11 12 13 14 15 16 17 18 19 20 21 22 23 24 25 26 27 28 29 30 31
unpack f f f f f f f f f f f f f f f f f f f f f f f f f f f f f f f f
415.3
outlet

The `pd sequencer` component takes in a list of 32 values, and outputs them one by one every half second. This allows us to create a simple melody.

```
In [ ]:
```