

Introduction to Physical Modelling

Physical Modelling Synthesis is a technique that differs from most of the traditional approaches in that it is not based on a spectral/timbral/physical description of a target sound.

Instead it attempts to model the physical description of what makes a sound, in such a way that the model synthesises a sound in an analogous way to the real sound generator.

It looks at the key sound-producing parts of an instrument and tries to re-create them computationally. Depending on the complexity of the model, this synthesis technique can be very efficient or not. However, it generally produces very good results, although parameter specification is a problem for complex models.

Digital waveguides

A Digital Waveguide is a model of a travelling wave on an instrument. It is a basic component of a number of physical model systems.

The idea here is that a travelling wave can be modelled by a block of samples that represent it at points along its length.

Take a string: a wave travelling up and down the string, if sampled at equally-spaced points along its length would yield such block of samples, which could be stored in computer memory.

Of course, the wave is not static, it keeps moving, so we can move the samples from one position to the other in that memory block, which would work as a *digital delay line*.

Delay lines

Digital delay lines are simple processors that store audio samples for an specified amount of time, releasing them afterwards.

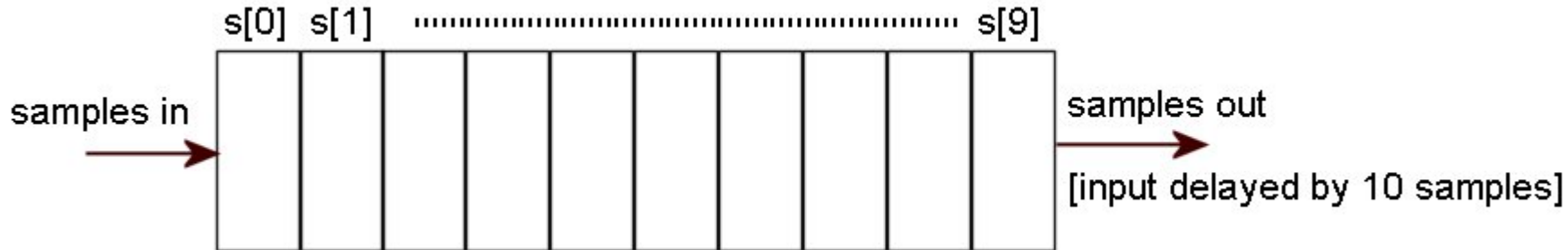
In a delay line, audio samples enter at one end and exit at the other, after a certain time delay, which will be proportional to the size of the delay line.

Delay lines are very useful for many applications, including echo, reverberation, pitch shifting, flanging, filtering, etc.

In physical modelling, they are used to implement *digital waveguides*. They can model a travelling wave on a string, a pipe, a bar, etc.

Delay time

The amount of delay is known as delay time. This can be given, independent of the sampling rate, in samples:

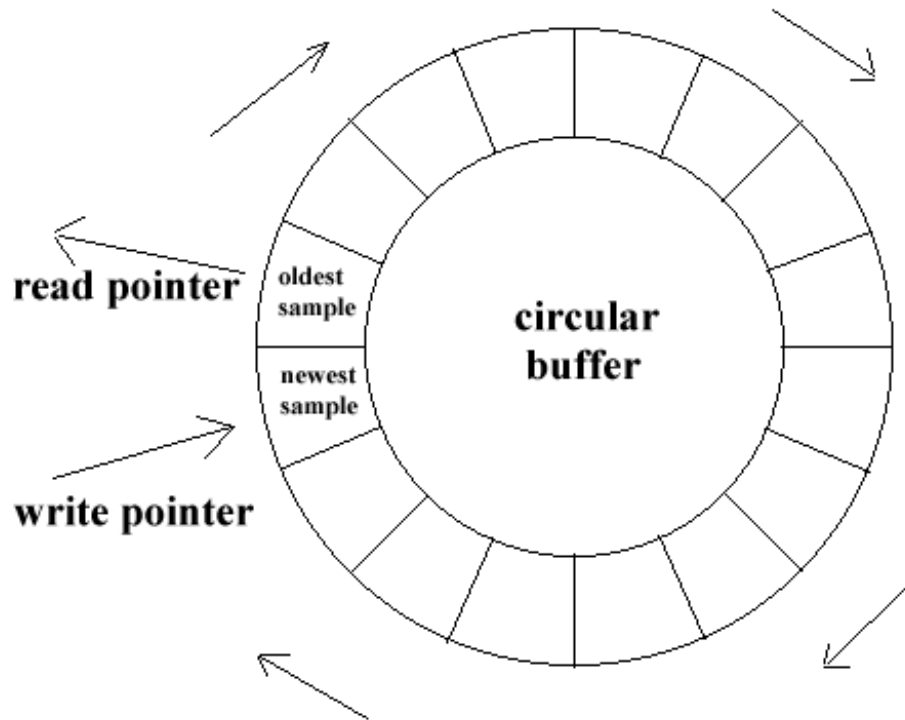


The delay line basically moves samples from one end to the other, one at a time, pushing one in and pop one out every sampling period.

In computing terms, the delay line is a memory block or *buffer*, which stores the required amount of samples.

Circular buffers

Delay lines are implemented digitally using *circular buffers*, which are memory locations accessed circularly by the computer. These are used to store the delayed samples.

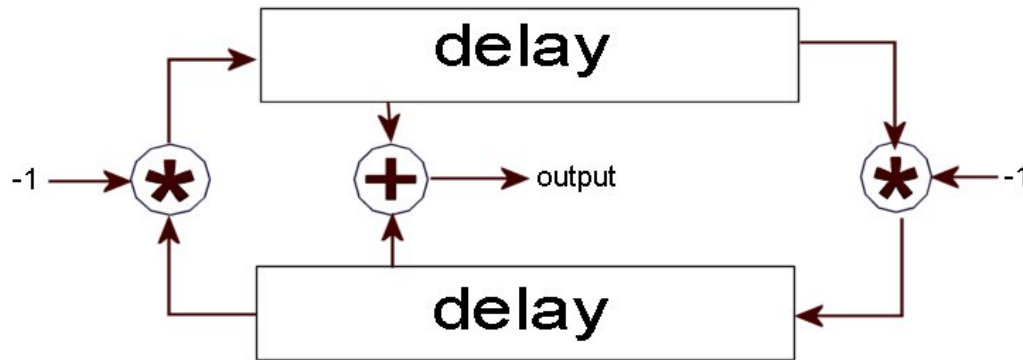


The time delay will be proportional to the size of the buffer.

For instance a *441-location* buffer will create a *0.01 s* delay (with $sr=44100$).

The basic string waveguide model

The basic theory of string waveguides was developed by J Smith in the late 80s/early 90s. He describes a basic waveguide as:



Two delay lines are used: one for the right-travelling wave and another for the left-travelling wave. The string is mounted on two rigid terminations that invert the phase of the wave (as a reflection). The output is taken from a certain position along the wave.

Fundamental Frequency

The string length L determines the pitch of a string of a certain mass and tension. If we apply the formula for the fundamental frequency of a string, we will be able to determine its pitch:

$$f_0 = \frac{c}{2L}$$

where c is the speed of sound along the string.

Waveguide pitch

The pitch of the waveguide model is determined in a similar way. It is clearly seen that the length L is equivalent to a delay line size S (in samples) of each of the two delaylines.

The speed of sound in a delay line is basically determined by the sampling rate. An audio sample travels N positions in a delay line in $N*sr$ seconds. So we can say that its speed is sr samples/s (of course !).

The fundamental of the waveguide model is then:

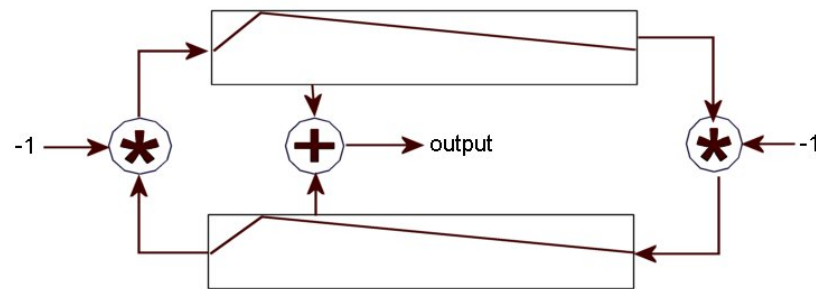
$$f_0 = \frac{sr}{2S} = \frac{sr}{2(\text{delay}(s) \times sr)} = \frac{1}{2 \times \text{delay}(s)} \text{ Hz}$$

the reciprocal of twice the delay time in seconds

Initial Excitation

The waveguide model needs an excitation to produce sound. For a plucked string, we can initialise the delay lines with the shape that the string takes when it is plucked.

The ideal shape is that of a triangle:



However, as the triangle has sharp corners, in practice we will have to smooth it (with a lowpass filter) in order to avoid aliasing.

Losses and decay

The model presented here does not include an important aspect of strings: the initial waveform energy is lost and dispersed after a certain period.

Take the example of a plucked string, after the initial attack, the sound decays due to the losses/dispersion of the original energy.

Losses are generally modelled by lowpass filters that can be inserted somewhere in the waveguide loop. Filters will then extract a bit of high-frequency energy every time the wave is reflected.

The Karplus-Strong algorithm

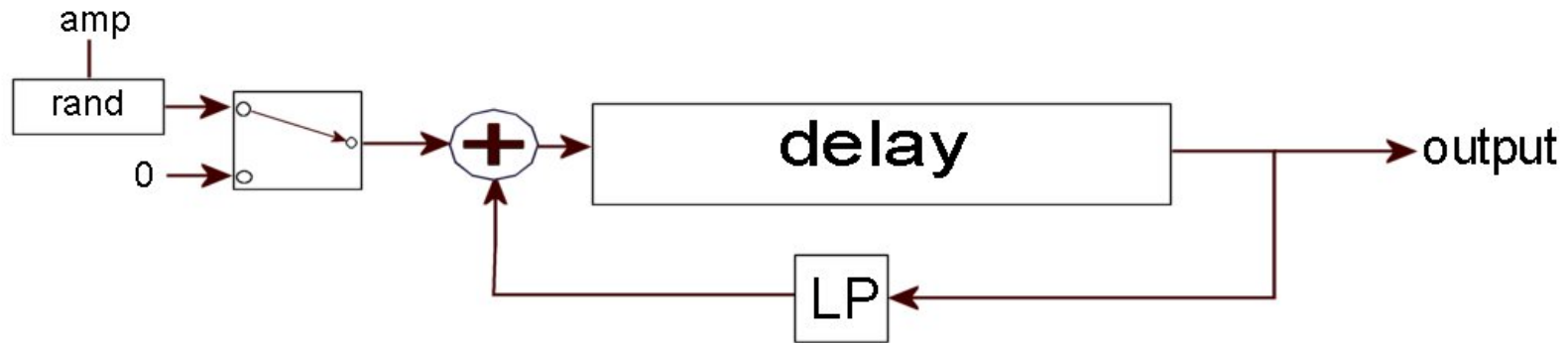
The Karplus-Strong (KS) algorithm was the first practical example of a waveguide use in physical modelling of plucked strings (and similar sounds). The KS idea in fact preceded the modern waveguide theory.

The idea is the same, a delay line is used to model a string, with a filter in the feedback loop to effect the decay in the signal.

There are two basic differences to the model presented before: (a) a single delay line is used and (b) we initialise it with random numbers (noise).

A basic KS design

The original algorithm is very simple (and efficient!):



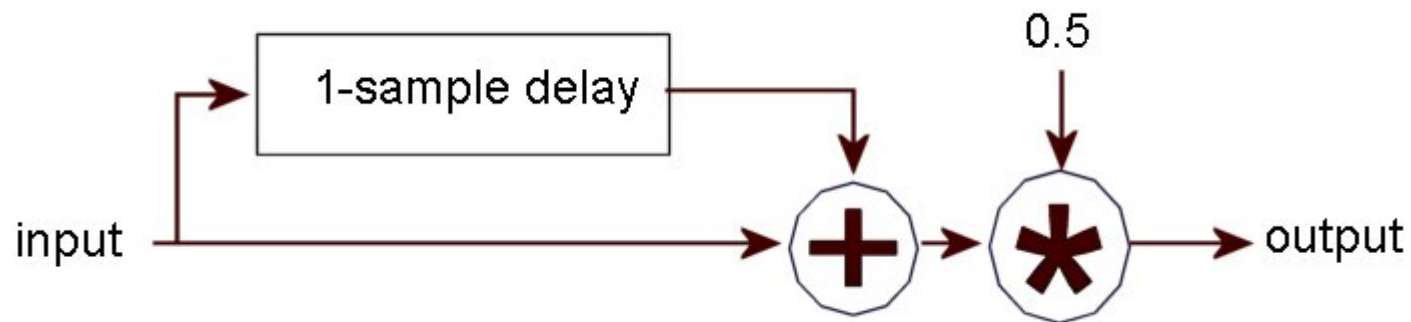
Here, the delay is going to be *twice* the size of the previous example to obtain a similar pitch height, as the delay line holds one complete wave period. The delay is filled with a noise input, which switches off after *delaytime* seconds.

The lowpass filter

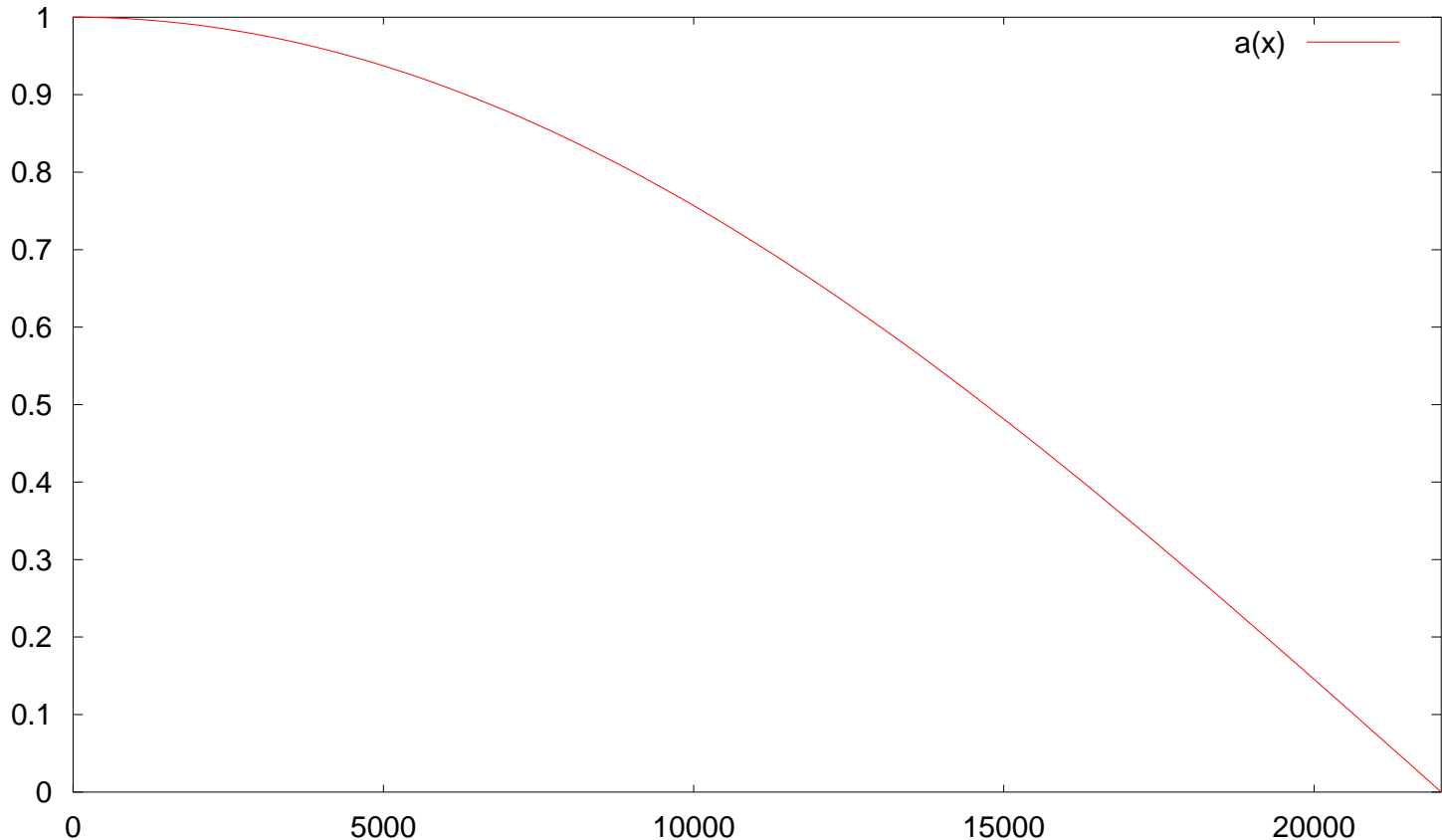
The lowpass filter used here is a simple averaging feedforward filter. It combines the current sample with the previous input sample, averaging them:

$$output[n] = 0.5(input[n] + input[n - 1])$$

This filter is implemented with a 1-sample delay



Filter characteristics



lowpass amplitude response: $A(freq) = \cos(\pi \times freq / sr)$

1/2 sample of phase delay: $\theta(freq) = -\pi \times freq / sr$

Implementing the KS algorithm

The basic KS algorithm can be implemented in `csound` by using the following components:

- (a) a noise generator
- (b) a signal switch/gate
- (c) a delay line
- (d) a low-pass averaging filter

The excitation signal can be taken from a `rand` opcode. The switch/gate can be implemented with a `if ... kgoto` control-flow statement. We only need to see now how to implement the delay line and lowpass filter.

Delay lines in csound

Csound features two basic opcodes implementing *fixed* delay lines:

```
asig delay ain, idel
```

and the read/write pair:

```
asig delayr idel  
      delayw ain
```

These will take delaytimes [idel] **in seconds**. The minimum delay allowed is $1/kr$ (1 control period).

Lowpass filter implementation

There are two ways of implementing the LP filter. One is using the `delay1` opcode:

```
asig delay1 ain /* delays a signal by 1 sample */
```

The other one is simpler. Since we will have to make $kr=sr$, because of the delay feedback loop, we can implement the filter equation directly

```
aprev init 0 /* initialise delay var */
alp = (asig + aprev)*0.5 /* filter equation */
aprev = asig /* fill the 1-smp delay */
```

A basic KS instrument

```
instr 1
iamp = p4
ifun = p5
idel = 1/ifun
aprev init 0

        kcount line 0, p3, p3      /* time counter */
        if kcount > idel kgoto off /* switch/gate */
        aexc rand iamp           /* noise excitation */
        kgoto wguide
off:   aexc = 0
wguide: aks delayr idel        /* delay output */
        alp = (aks + aprev)*0.5 /* filter */
        aprev = aks
        delayw alp+aexc         /* delay input */
        out aks

endin
```

Improving the KS design

Two main problems exist with the KS design. The first one is that as the pitch is determined by the total delay time, the instrument will always be tuned to

$$f_0 = \frac{sr}{(S + 1/2)}$$

where S is the delay line size in samples. This is always an integral value, so that fine-tuning of the instrument is impossible. For low-frequencies, the tuning discrepancy is tolerable, but it becomes an issue at high-frequencies, when delay sizes are increasingly smaller.

Tuning solutions

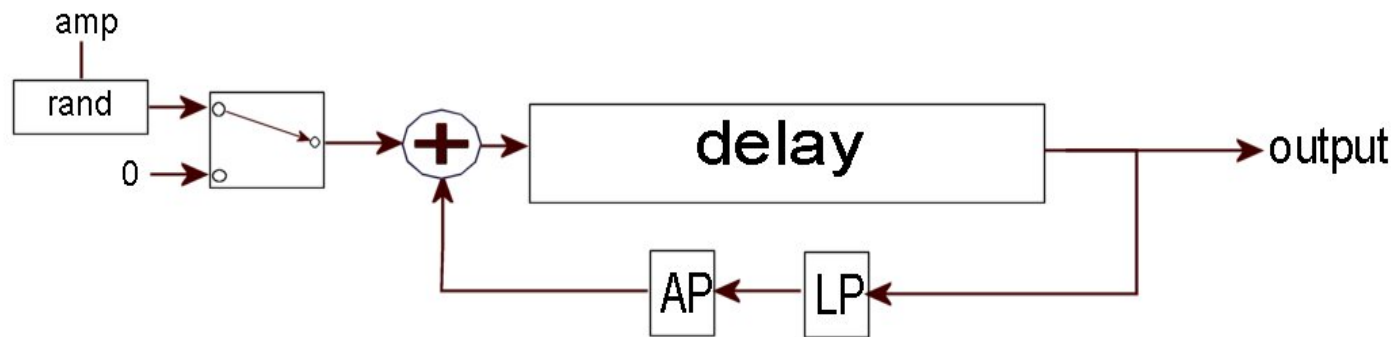
One solution for the tuning problem that is found in the csound KS opcode implementation (`pluck`) is to use the delayline more or less like a wavetable and read it like an interpolated oscillator would do. The 'table' is updated regularly (at every wrap-around) with the filter equation to effect the decaying characteristic.

This is simple in concept, but slightly tricky to implement in the csound language and perhaps not so efficient, as the filtering has to be done separately.

The classic solution is to use a tuning filter, an allpass, which only affects the signal delay, not its amplitude.

Using an allpass filter

The idea here is to insert the allpass filter after the lowpass, so that with it we can delay the signal fractionally and tune the instrument:



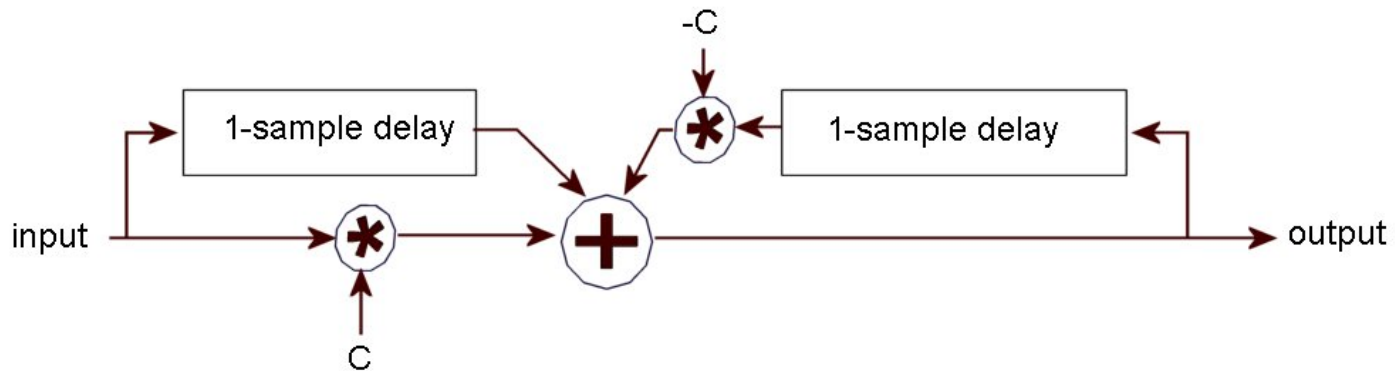
The allpass filter will effectively depend on only one parameter, which can be calculated from a desired fractional delay (between 0 and 1).

Allpass equation and parameters

An allpass filter combines a signal with a feedforward delay and a feedback delay. The delays are 1-sample, as in the LP case:

$$output[n] = input[n] \times C + input[n - 1] - output[n - 1] \times C$$

The feedback delay, is of course the previous output sample.



Allpass coefficient estimation

The allpass filter will be used to complement a fractional delay that is needed to fine-tune a certain fundamental.

We'll start by setting the (integral) delay size to $\text{int}(sr/fun)$. If that, plus the 1/2 sample delay added by the lowpass, is bigger than the desired (decimal) delay size sr/fun , we'll subtract 1 from it. The fractional delay that the allpass needs to supply will be:

$$D = sr / fun - [\text{int}(sr / fun) + 0.5]$$

The allpass coefficient C is estimated by the relationship:

$$C = \frac{1 - D}{1 + D}$$

An improved KS implementation

In order to modify the first KS design, we'll need to do the following:

- (a) set the delay size to a precise number of samples
- (b) implement the allpass filter

The allpass filter can be implemented in a very similar fashion to the LP example of the original instrument.

The instrument will require an alternative delayline implementation, so that a precise delay size in samples is achieved.

Alternative delay implementation

In order to obtain a delay with the right number of samples, we will need to set up a 'tap' in the delay line, with the opcode `deltapn` :

```
adp delay 1 /* allocates 1 sec of del memory */
adel deltapn idelsamps /* delay of idelsamps samples */
delayw ain
```

Here we have a delay line set up with a max delay of 1 sec. We tap it at `idelsamps` samples of delay. This will guarantee a correct number of samples of delay.

Delay lines can have one or more taps, and these can be set in terms of delay time in secs (truncating/interpolating) and samples (as above).

A fine-tuned KS instrument: parameters set-up

```
iamp = ampdb(p4)      /* amplitude (dB) */
ifun = cpspch(p5)    /* fundamental (octave.pitch-class) */
idts = sr/ifun       /* total delay time (samples) */
idtt = int(sr/ifun)  /* truncated delay time */
idel = idts/sr       /* delay time (secs) */

ax1  init 0          /* filter delay variable */
apx1  init 0         /* allpass fwd delay variable */
apy1  init 0         /* allpass fdb delay variable */

idtt = ((idtt+0.5) > idts ? idtt - 1 : idtt)
ifd = idts - (idtt + 0.5) /* fractional delay */
icoef = (1-ifd)/(1+ifd) /* allpass coefficient */
```

A fine-tuned KS instrument: sound synthesis

```

        kcount line 0, p3, p3          /* time counter */
        if kcount > idel kgoto off    /* switch/gate */
        aexc rand iamp                 /* noise excitation */
        kgoto wguide
off:    aexc = 0
wguide: adp delayr 1                 /* delay output */
        adel deltapn idtt              /* delay in samples */
        aflt = (adel + ax1)*0.5        /* averaging filter */
        ax1 = adel
        alps = icoef*(aflt - apy1) + apx1 /* allpass filter */
        apx1 = aflt                    /* update fwd delay */
        apy1 = alps                    /* updated fdb delay */
        delayw alp+aexc                /* delay input */
```

Decay control

The improved KS instrument still has one slight problem: *sound decay* cannot be controlled. It is either too long, on low fundamentals, or too short in high pitches.

Shortening decays is easy, all we need is to apply an attenuating gain to the delay feedback (between 0 and 1). Lengthening the decays can be achieved by using a slightly different LP filter, instead of the averaging one.

The decay lengthening filter has the following form:

$$output[n] = (1 - K) \times input[n] + K \times input[n - 1]$$

Combined with the gain attenuation:

$$output[n] = \{(1 - K) \times input[n] + K \times input[n - 1]\} \times g$$

LP characteristics

This low-pass filter is different from the basic LP in one point, it does not have a linear phase response, so the delay it imposes is not always $1/2$ sample. For low frequencies, it is approximately the value of the coefficient K in the equation.

However for high-frequencies it can be higher/lower than that value, causing it to stretch/compress the tuning of partial frequencies.

Stretched partials is not a bad thing, something often observed in acoustic string instruments (such as the piano).

Setting the decay time

It is possible then to set a specific decay time for a KS instrument. All we need is to check whether the basic KS configuration needs decay shortening or lengthening.

If shortening is needed, all we need to do is to attenuate with a certain gain setting. If lengthening is needed, we can calculate the filter coefficients to do that.

We check if any changes are needed by comparing a required gain for a certain decay rate with the amplitude response of the averaging filter for the required fundamental.

Decay rate check

The gain for a certain decay rate at certain fundamental can be estimated as follows

$$G_f = 10^{\frac{atten}{20 \times fund \times \Delta t}}$$

where *atten* is the attenuation in dB (negative) and Δt is the decay time.

The gain imposed by the averaging filter at a certain fundamental is its amplitude response:

$$G = \cos(\pi \times fund / sr)$$

We just check which one is bigger, to see if we need to shorten or to lengthen the decay. The gain used to shorten the decay will be

$$G / G_f$$

Decay lengthening

This is done by calculating the coefficient A for the LP filter to obtain a certain gain $G_f(fund)$. This turns out to be a quadratic equation $ax^2 + bx + c = 0$, which has a well-known method of solution ($x = -b \pm [b^2 - 4ac]^{1/2}/2a$):

$$(2 - 2\cos\omega)A^2 + (2\cos\omega - 2)A + 1 - G_f(fund) = 0$$

This would yield two solutions for A , one between 0 - 0.5 and another between 0.5 and 1. The lowest will stretch the partials and the higher will flatten them.

Since stretching is probably closer to the real strings that we are trying to model, we'll take the lowest.

A complete KS design

Now all we need is to incorporate those ideas in the instrument. We'll change 0.5 for K , as well as the LP equation and add the filter/G calculations to the code. These are the additions:

```
igf pow 10,-idec/(20*ifun) /* gain required for a certain decay */
ig  = cos(ipi*ifun/sr)     /* unitary gain with A=0.5, ipi is PI */

if igf > ig  igoto stretch /* if decay needs lengthening */
ifac = igf/ig                /* if decay needs shortening */
goto continue

stretch: icosfun = cos(2*ipi*ifun/sr) /* quadratic formula */
         ia = 2 - 2*icosfun
         ib = 2*icosfun - 2
         ic = 1 - igf*igf
         id = sqrt(ib*ib - 4*ia*ic)
         is1 = (-ib + id)/(ia*2)
         is2 = (-ib - id)/(ia*2)
         ik = ( is1 < is2 ? is1 : is2) /* take the lowest solution */
```

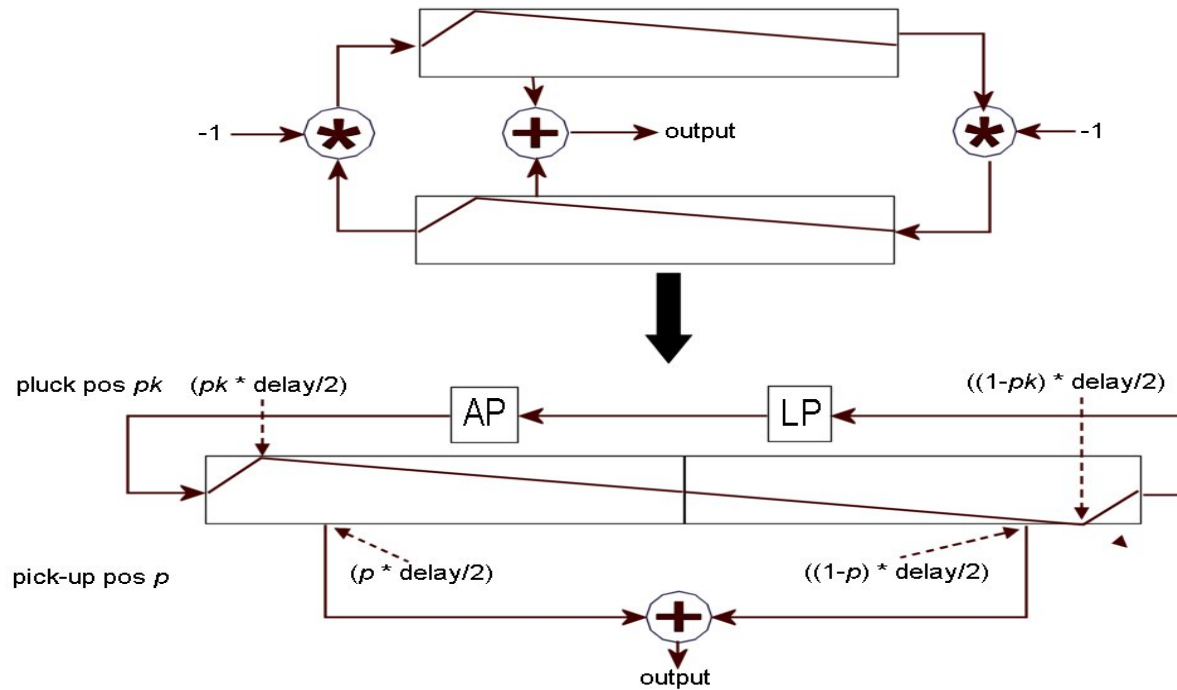
Using a modern waveguide model

It is possible to combine the ideas developed as part of the KS design into the modern string waveguide model.

We can use the loss + tuning filter (LP+AP) with the basic waveguide model. The double delay is combined into a single one, for the sake of simplicity and the phase inversion at the ends of each delay are cancelled out. This is simpler to implement than the double-delay design and is especially simpler to tune (all we need is to use the same AP arrangement from before).

We initialise it with a triangle-shape transversal wave, lowpass-filtered (we can use the same averaging LP design, or else another LP filter).

Plucked-string waveguide model



This design has some advantages:

- (a) plucking position can be altered, by altering the initial waveshape
- (b) 'pick-up' position can be controlled, by tapping the delay lines at different points
- (c) it produces a less metallic output than the KS sound (that, of course, depends on the initial waveshape)

Other physical models

Wind and percussion instruments have also been modelled with waveguides.

Other physical model approaches also offer a very primitive set of tools from which any 'real' physical instrument can be created. They normally include things like models of mass, cords, striking forces, etc., which can be combined together to make a sonic model.

In addition to waveguides, there are other PM mathematically-intensive based on the solutions of the *wave equation*, a fundamental differential equation that models vibrating systems.

Other important aspects to consider in PM are resonances and system responses. These can be added to provide support for instrument bodies, etc., using, for instance, resonators and other types of filters.