A Lossless, Click-free, Pitchbend-able Delay Line Loop Interpolation Scheme

Scott A. Van Duyne
David A. Jaffe
Gregory Pat Scandalis
Timothy S. Stilson

Center for Computer Research in Music and Acoustics, Stanford University
savd@ccrma.stanford.edu, daj@ccrma.stanford.edu,
gps@stanford.edu, stilti@stanford.edu

Abstract

An efficient method for signal controllable fractional delay implementation has been found. It combines the flexible control of linear interpolation with the frequency independent losslessness of allpass interpolation, avoiding the undesirable effects of each method.

1 Problem Statement

The development of high quality, well calibrated, physical modeling synthesis algorithms based on Karplus-Strong-style feedback loops [3][2] requires the use of delay lines with non-integer lengths. Ideally, these delay lines should have two important features:

(1) Their lengths must be smoothly and fractionally variable by some control signal in order to implement pitch bend, glissando, and vibrato effects.

(2) They must be lossless at all frequencies to minimize unwanted decay in physical modeling feedback loop structures.

Unfortunately, no standard methods for interpolation of non-integer length delay lines are in general use which have *both* of these required features. On the one hand, non-integer length delay lines that use *linear interpolation*, or other FIR interpolation methods, can be varied smoothly in length by a control signal, but they have unsatisfactory energy losses caused by the FIR interpolation filter itself in the high frequency region and, in particular, in high pitched loops. This causes high pitched musical notes to decay away too quickly.

On the other hand, standard *allpass interpolation*, known since the beginnings of Karplus-Strong-style string modeling [2], solves the energy loss problem for the fixed pitch case. However, when implementing pitch bend, glissando, or vibrato effects, allpass filters introduce undesirable artifacts, such as audible clicks. This is primarily due to the internal state in the recursive allpass filter, which must be handled carefully when changing the filter coefficient. Until now, the practical choice has been between allpass interpolation for high frequency sustainability on the one hand, and linear interpolation for flexibility of pitch bend control on the other.

We have formulated a new delay line interpolation structure which has the time-varying delay length flexibility of simple linear interpolation, while retaining the energy conserving effects of fixed allpass interpolation. This was achieved by combining previous results from three different quarters: a smooth waveguide legato implementation crossfading trick [1], some initial results in click reduction in time varying fractional allpass interpolation [6], and a few psychoacoustical observations about just noticeable differences in pitch [5].

2 Puzzle Pieces

2.1 Linear Interpolation

For good tuning of a Karplus-Strong loop, the delay line length must be equal to:

$$DelayLength = SamplingRate/Frequency$$

This generally comes out to a non-integer number of samples, and rounding to the nearest integer is just not good enough at current sampling rates, e.g., 44.1 kHz. It is easy to delay a signal by 25 samples, but delaying a signal by 25.3 samples is problematic in a sampled system since 0.3 samples is undefined!

Linear interpolation solves the problem by taking a weighted average of the two closest delay lengths. The

following linear interpolation, illustrated in Figure 1, implements a delay of 25.3 samples:

$$OUT(n) = 0.7 \times IN(n - 25) + 0.3 \times IN(n - 26)$$

The problem with linear interpolation is that high frequencies tend to get wiped out quickly with repeated averaging. Remember that the original Karplus-Strong plucked string algorithm [3] called for a two point average filter in a delay line feedback loop, similar to the equation above. The purpose of the averaging filter was to smooth the waveform a little bit each period, thereby forcing the high frequencies to die away faster than the low frequencies, and to create the qualitative effect of a plucked string decay. This effect is very strong for high pitched loops since the delay line part is short and the loss is greater for high frequencies. This means getting a good long sustain on a Karplus-Strong-style high guitar string sound is impossible (unless you don't care if it is in tune!).

On the other hand, the linear interpolation approach is very flexible. If the you want to change the delay length continuously, as in bending the pitch of a note in the plucked string model, then just slide the linear interpolator along the delay line adjusting the sample weights appropriately.

More extravagant weighted averages, known as FIR interpolation methods, take advantage of more than just two adjacent samples and can improve the frequency response of the interpolation some, but also are increasingly more difficult to compute. Laakso et al. give a comprehensive review of both FIR and allpass interpolation design methods in [4].

2.2 Allpass Interpolation

Allpass interpolation solves the problems of high frequency energy loss in feedback loops which are presented by linear interpolation and FIR schemes. It trades error in the magnitude response, which causes unwanted decay in the high frequencies, for error in the phase response, which only causes incidental detuning of the highest partials. Allpass interpolation in the context of Karplus-Strong models was first proposed by Jaffe and Smith [2] in 1983. They noted that, for a desired fractional delay of d samples, an allpass coefficient of

$$a \approx (1-d)/(1+d)$$

could be chosen as a reasonable approximation.

The following allpass interpolation scheme, illustrated in Figure 2, implements a delay of 0.3 samples:

$$OUT(n) = a \times x(n) + x(n-1) - a \times OUT(n-1)$$
where, $a \approx (1-0.3)/(1+0.3) \approx 0.5385$.

Note that allpass interpolation is recursive; that is, the interpolation uses not only a combination of input samples, $x(n)$ and $x(n\text{-}1)$, but also adds in part of its previous output sample, $OUT(n)$. If you continuously change the coefficient, a, to create a pitch bend or glissando effect, then very special attention must be paid to correcting for the recursive effect if you want to avoid clicks and glitches in the sound as you change pitch [6]. Furthermore, the problem of what to do when you change the integer part of the delay line length, as well as the fractional allpass interpolation part, is an other complicated problem.

Minimizing the transient effect The discontinuity resulting from changing the coefficient, a, can be minimized by keeping the coefficient value as close to zero as possible. The transient effect of changing the coefficient rings out at a rate proportional to the series: a, a^2, a^3, … We note that if the delay, d, is kept within the unit range, 0.618 to 1.618, then the coefficient, a, remains between -0.236 and +0.236. This means that, with d in this range, the transient effect after 5 samples is a maximum of $(0.236)^5$ or about 62 dB down. In effect, the allpass interpolation filter may be held to a 5 sample warm up time. [6] makes a similar observation.

2.3 The Legato Crossfade Trick

Another piece of the puzzle is a solution to the problem of producing legato transitions between tones of different pitch using the a single feedback loop. If you change the delay length suddenly there is a click in the sound. If you gradually glide the delay length from the first value to the second, using, for example the sliding weighted average linear interpolation method described above, you will hear and unwanted glissando effect, rather than a legato effect.

A legato crossfade method is described briefly in [1] in the context of legato commuted synthesis violin bowing. The first part of the legato trick was the use of a *circular buffer* delay line implementation. Circular buffer simply means that the delay line is implemented in a large fixed length piece of memory with a read pointer chasing a write pointer around, always the appropriate number of samples behind it. Although the actual delay length is shorter than the full length of the memory being used, nevertheless, the full memory is filled with similar looking waveform. That is, as the write pointer progresses through the cicular buffer memory, it lays down perfectly good waveform at the currently specified pitch throughout the full memory buffer.

Jaffe noted that if you simply introduce *two* read pointers, one set at the delay of the *first* note, and the second set for the delay of the *second* note, then you can crossfade between the two read pointers, over the course of about 15-30ms, to produce a very realistic legato, *not glissando*, effect. This structure is illustrated in Figure 3. There is no glitch in the tone since the full memory buffer is filled with reasonable looking waveform at the current pitch. Therefore, the *second* read pointer is looking are perfectly good data initially and the crossfade is gradual. Stilson developed a similar trick, independently, for use in a pitch shifting algorithm.

3 Glissable Allpass Interpolation

By combining elements of the interpolation and legato methods described above, we can find a practical structure for a flexible lossless fractional delay line. The basic inspiration is this: Let's view glissando as a lot of very fast, tiny, legato transitions. Start with a circular buffer delay line with two *allpass interpolated* readers. Then send new fractional delay length values to the alternating allpass interpolated readers every 16 samples, for example.

What is the problem with this? The allpass interpolation filters will be producing clicks every time a reader is set to a new position! *But* the transient effect lasts only 5 samples if the fractional delay range is maintained between 0.618 and 1.618! The 5 warm-up can be ignored by using a special crossfading function which waits 5 samples before crossfading over to the newly set allpass interpolated reader. When using a 16 sample alternation rate, this leaves 11 samples to do the actual crossfade, which, in practice, is enough. See Figure 4.

Psychoacoustical Detail The human hearing system is only able to detect a finite number of different pitches. Two tones which are sufficiently close together in pitch become indistinguishable. There is a *just noticeable difference* (JND) threshold for human pitch differentiation. The number of JNDs per octave varies with the register, but a representative worst case for us is that there are about 280 JNDs between 1000 Hz and 2000 Hz (or, approximately, between c6 and c7) [5]. A JND comes out to about 0.1 samples in a Karplus-Strong feedback loop delay length for a 1000 Hz tone being computed at a sampling rate of 44.1 kHz. It is easy to show that running the alternating crossfader at a tick rate of once per 16 samples, and using a maximum glissando rate of one JND per tick, that we can gliss an octave from c6 to c7 in about one tenth of a second $(280 \times 16 / 44100 \approx 0.1)$.

DSP Implementation Trick Many fixed-point DSP chips can perform very fast multiply-add operations but do *not* support a fast divide operation; so, computing the allpass coefficient,

$$a \approx (1-d)/(1+d),$$

every 16 samples is actually rather inconvenient. Fortunately, we may expand the expression in a Taylor Series about the point, $d = 1$, giving,

$$a \approx -\frac{(d-1)}{2} + \frac{(d-1)^2}{4} - \frac{(d-1)^3}{8} \cdots$$

a very efficient computation using only multiplies, adds and, possibly, right shifts. Maximum error in three terms is 0.024 samples. Recall a JND at 1000Hz is about 0.1 samples.

4 Acknowledgments

Funding for this work was provided by the Stanford University through the Office of Technology Licensing as part of the Sondius® trademark development program, in collaboration with the Center for Computer Research in Music and Acoustics.

References

[1] Jaffe, J., and J. Smith. 1995. “Performance Expression in Commuted Waveguide Synthesis of Bowed Strings,” *Proc. ICMC*, Banff.

[2] Jaffe D., and J. Smith. 1983. “Extensions of the Karplus-Strong Plucked String Algorithm,” *Computer Music Journal*, Volume 9, Number 2.

[3] Karplus, K., and A. Strong. 1983. “Digital Synthesis of Plucked-String and Drum Timbres,” *Computer Music Journal*, Volume 7, Number 2.

[4] Laakso, T.; V. Valimaki; M. Karjalainen; and U. Laine. 1996. “Splitting the Unit Delay,” *IEEE Signal Processing Magazine*, January.

[5] Olson, H. 1967. *Music, Physics, and Engineering*, Dover Publications, Inc.

[6] Valimaki, V.; T. Laasko; and J, Mackenzie. 1995. “Elimination of Transients in Time-Varying Allpass Fractional Delay Filters with Application to Digital Waveguide Modeling,” *Proc. ICMC*, Banff.

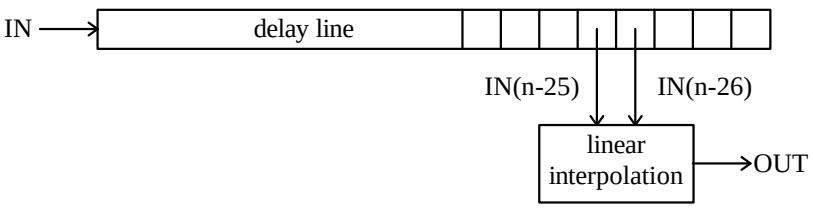

Figure 1: Flexible Linear Interpolation

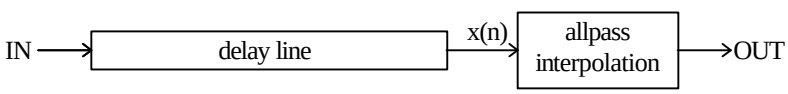

Figure 2: Fixed Allpass Interpolation

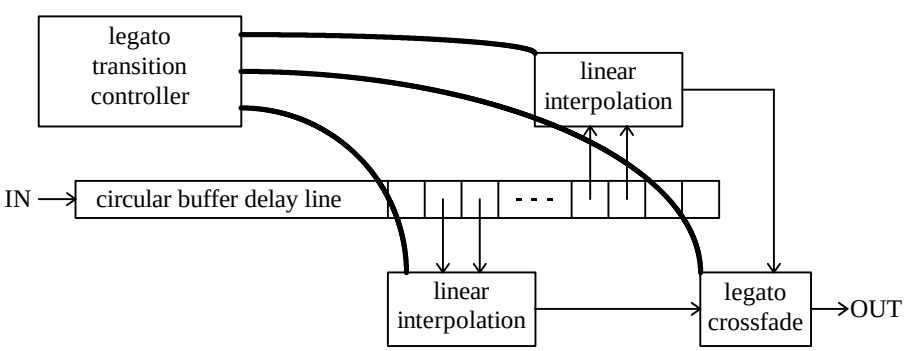

Figure 3: Legato Transition Trick

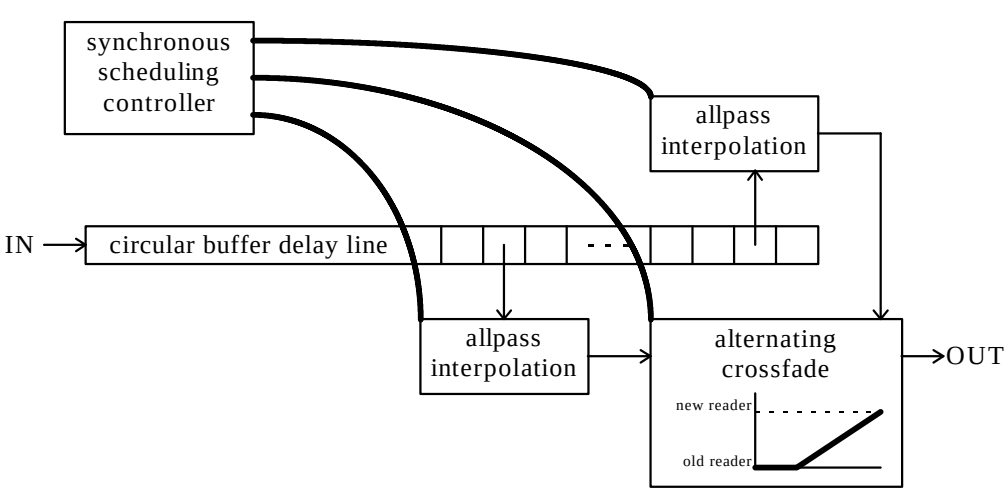

Figure 4: Glissable Allpass Interpolation