EFFICIENT EMULATION OF TAPE-LIKE DELAY MODULATION BEHAVIOR

Vadim Zavalishin, Julian D. Parker

Native Instruments GmbH
Berlin, Germany
firstname.lastname@native-instruments.de

ABSTRACT

A significant part of the appeal of tape-based delay effects is the manner in which the pitch of their output responds to changes in delay-time. Straightforward approaches to implementation of delays with tape-like modulation behavior result in algorithms with time complexity proportional to the tape speed, leading to noticeable increases of CPU load at smaller delay times. We propose a method which has constant time complexity, except during tape speedup transitions, where the complexity grows logarithmically, or, if proper antialiasing is desired, linearly with respect to the speedup factor.

1. INTRODUCTION

Delay and echo effects have been fundamental tools for manipulating space and rhythm in music production since the 1950s, with the first commercial units being based on loops of magnetic tape. Over the following decades, other methods for producing such effects were developed, including the use of magnetic drums, and bucket-brigade chips [1, 2, 3]. Starting in the 1970s, digital implementations of delay-lines became available.

A delay line can broadly be thought of as a black-box into which a signal is passed, and which outputs it at some later ('delayed') time. Independent of the technology involved, all delay lines work in fundamentally the same way. The signal is injected into a medium at a particular point, it travels through that medium for some time, and is received at another point. This medium can be magnetic tape, a chain of capacitors, a digital ring-buffer, or even potentially an acoustic or mechanical system. Despite the change in sound-character imposed by the medium, the broad difference between different types of delay-line is the way in which they allow the delay time to be varied. Some allow the distance between the entry point and exit point to be varied (we call these *length-type* delays), whilst others instead manipulate the speed at which the sound traverses the medium (we call these *speed-type* delays). This difference is exemplified by two famous tape-based delay devices - the EchoPlex [4], and the Roland Space Echo series. The former allows the read head of the tape machine to be moved, whereas the latter allows the speed of the motor driving the tape to be changed. This distinction is important, because it greatly influences the pitch-change perceived when manipulating the delay-time, especially when the system is subjected to feedback as is the case in echo effects. In the case of length-type delays, the pitch-change perceived in the output (and recirculated when feedback is present) is dictated purely by the rate of change of the length. In the case of speed-type delays, the change in pitch is defined by the ratio of speeds between the instant the signal entered the medium and the instant it exits. It turns out that the latter behaviour is desirable musically, as it leads to much more consistent control over pitch. For example, a repeating echoing sound can be cleanly pitched up and down by varying the delay-time in the latter case, whereas in the former the same manipulation will result in erratic overlapping pitch changes.

Typical implementations of digital delays are based on a variable length ringbuffer, where the delay time parameter controls the distance of an interpolated read-point from the write-point [5]. This implementation clearly falls into the length-type category, and exhibits the expected problems. Straightforward speed-style digital delays can be implemented using a fixed array and a varying internal sample-rate. Thus, there must be sample rate conversion at the write head (from the outside sampling rate to the sampling rate implied by the speed) and at the read head (from the implied to the outside sample rate). Moreover, the number of processed "internal samples" per one "outside sample" is proportional to the speed. Thus, the time complexity of such emulation is $O(v)$, where v is the speed. This particularly means that at low delay times, where the tape speed is high, the CPU load significantly increases. Another interpretation of variable-sample-rate digital delay is given by Rocchesso [6] who frames the process using interpolated read and write points. Holters extends this variable-sample-rate paradigm to use the BBD circuits input and output filters to perform the necessary interpolation [3].

Huovilainen [1] proposed a way to recompute speed variations into length variations, which can then be used to simply control the read position of a ringbuffer-based digital delay. However his method of recomputation, even though reducing the overall computation costs, still has an $O(v)$ complexity.

In this paper, we propose a new method of implementing a digital speed-style delay which in steady-speed situations and during slowdowns has an $O(1)$ complexity, regardless of the actual speed. During speedup transitions the method has an $O(\log K)$ complexity, where K is the speedup factor. If proper antialiasing of speedups is desired, then during speedup transitions the computation complexity grows to $O(K)$.

For the sake of a more intuitive language we will be talking of emulation of a tape delay. However the discussion will equally apply to other types of speed-style delays.

In Sec. 2, we introduce a continuous-time model for the relationship between tape speed and delay-time, which we call the *tape equation*. We then discuss some results that can be derived directly from the model. In Sec. 3, we describe a numerical scheme for solving the tape equation in the case of arbitrary changes in speed, including consideration of aliasing distortion in cases where tape speed is increasing. In Sec. 4 we present some measurements of the computation efficiency of the described method, in comparison to existing methods. In Sec. 5, we conclude.

2. THE TAPE EQUATION

In real-world tape-delays the tape is often looped, with an erasing head placed before the recording head. Since the erasing head removes the previous signal recorded to the tape, without loss of generality we can consider this configuration to be equivalent to a tape of infinite length in both directions. We will associate a one-dimensional coordinate system with the tape, so that each point on the tape has a coordinate.

Let $x_w(t)$ be the coordinate of the tape point positioned against the write head at the time moment t, and let $x_r(t)$ be the coordinate of the tape point positioned against the read head. Let the tape move in the direction from the write head to the read head and let's orient the tape coordinate axis x so that $x_w(t)$ and $x_r(t)$ will increase with time as the tape moves. Then, if $v(t)$ denotes the speed of the tape at time moment t,

$$\dot{x}_w(t) = \dot{x}_r(t) = v(t) > 0$$

Let the distance between the heads be fixed at

$$x_w(t) - x_r(t) \equiv L > 0 \tag{1}$$

and let $T(t)$ denote the *effective delay time* at time moment t, meaning that the signal value which is being picked up by the read head at the time moment t was written by the write head at the time moment $t - T(t)$:

$$x_r(t) = x_w(t - T(t)) \tag{2}$$

Clearly, during the time range $[t-T(t), t]$ the tape has travelled the distance equal to $\int_{t-T(t)}^{t} v(\tau)\,\mathrm{d}\tau$ and this distance must be equal to L:

$$\int_{t-T(t)}^{t} v(\tau)\,\mathrm{d}\tau = L \tag{3}$$

The above formula is the *tape equation* in the integral form. It relates the tape speed function $v(t)$ to the effective delay time $T(t)$. In case of constant speed on the range $[t - T(t), t]$ the formula turns into $v \cdot T(t) = L$ giving

$$T(t) \equiv L/v \tag{4}$$

We can refer to L/v as *steady-state delay time*.

By introducing the "total distance travelled by the tape"

$$V(t) = \int v(\tau)\,\mathrm{d}\tau \tag{5}$$

(which is understood in the sense that $V(t)$ is *some* arbitrary antiderivative of $v(t)$) the tape equation can be rewritten in the difference form

$$V(t) - V(t - T(t)) = L \tag{6}$$

It is convenient to choose the constant of integration in $V(t)$ in such a way that

$$x_w(t) = V(t) \tag{7a}$$
$$x_r(t) = V(t) - L \tag{7b}$$

Alternatively, if we wish to choose the constant term of $V(t)$ from some other considerations, (7) can be enforced by the choice of the origin of the coordinate axis x. From this point on we will assume that (7) always holds.

By taking the time derivative of (6):

$$v(t) - v(t - T(t)) \cdot \left(1 - \frac{\mathrm{d}}{\mathrm{d}t}T(t)\right) = 0$$

and peforming algebraic transformations, we obtain the tape equation in the differential form:

$$\frac{\mathrm{d}}{\mathrm{d}t}T(t) = 1 - \frac{v(t)}{v(t - T(t))} \tag{8}$$

By describing the relationship between tape-speed and delay-time, the tape equation can allow us to produce tape-like behaviour using an ordinary variable-length ringbuffer-based digital delay. In order to achieve this, the equation must be solved either analytically or numerically.

2.1. Suitability of equation forms for numerical solution

The differential form (8) of the tape equation doesn't contain information about the distance between the heads. Thus, there is no "built-in error correction mechanism" in (8) and a straitforward numerical solution could exhibit errors which accumulate into an indefinitely large drift of $T(t)$. Intuitively, consider the following example. Suppose $v(t)$ varies for a while and then the variations stop: $v(t) = \mathrm{const}\ \forall t \geq t_0 - T(t_0)$. In this case the right-hand side of (8) will be zero $\forall t \geq t_0$ and thus any error accumulated in $T(t)$ will stay there forever.

Consequently, the differential form is not well suited for use in practical delay implementations. However in theoretical work it can be useful in combination with differential equation solvers, such as the ones found in CAS (computer algebra system) software, which often expect the equation to be supplied in the differential form.

The integral form (3) contains a different potential numerical drift source. (3) suggests an incrementally computed moving sum as a numerical implementation, where the new terms of the form $v(t)\Delta t$ will be incrementally added and the old terms $v(t - T(t))\Delta t$ will be subtracted. However, even if what we add exactly equals what we subtract later, addition and subtraction of the same value might not totally cancel each other due to limited precision of floating point calculations. This error also can accumulate.

Note that, if we instead use fixed point calculations in the moving sum, the addition and subtraction of equal values will exactly cancel. So, if we can make sure that we add and subtract exactly the same values, there will be no drift. However, as we shall see in the further discussion, it will be even easier to simply use the difference form (6), where we spare the subtraction of $v(t-T(t))\Delta t$ and therefore don't need to consider the resulting error.

2.2. Analytical solution for an instantaneous jump in speed

It is educative to consider the case of a single instantaneous jump in the tape speed, the speed being constant at all other times. In this case there is a simple analytic solution to the tape equation. Indeed, let

$$v(t) = \begin{cases} v_0 & \text{if } t < 0 \\ v_1 & \text{if } t \geq 0 \end{cases}$$

Let $V(0) = 0$, giving

$$V(t) = \begin{cases} v_0 t & \text{if } t < 0 \\ v_1 t & \text{if } t \geq 0 \end{cases} \tag{9}$$

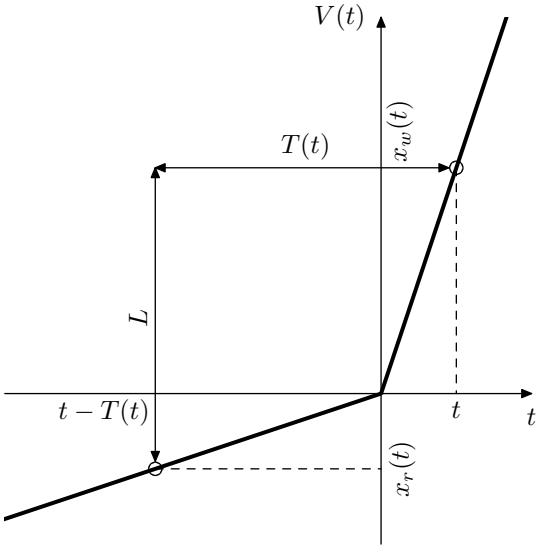

Figure 1: Graphical interpretation of (6).

and we choose the origin of coordinate x so that (7) holds.

Now we want to substitute (9) into (6), thus obtaining $T(t)$. It is highly instructive to use the graphical interpretation of (6) given in Fig. 1. The figure represents the graph of $V(t)$ defined by (9) with highlighted points $x_w(t)$ and $x_r(t)$ corresponding to the write and read head positions at time t. By (1) the vertical distance between these points is L, and by (2) the horizontal distance between these points is $T(t)$. Thus, both write and read heads move along the curve $V(t)$ in such a way that the vertical distance between these points is always L, thereby defining the horizontal distance between them, which is $T(t)$.[1]

We could use Fig. 1 to construct the explicit formula for $T(t)$. From (7a) we have $x_w(t) = V(t)$. Then using Fig. 1 we obtain $x_r(t) = x_w(t) - L$ and $t - T(t) = V^{-1}(x_r(t))$. Combining all these formulas together yields

$$T(t) = t - V^{-1}(V(t) - L) \tag{10}$$

where $V^{-1}(V(t) - L)$ is simply the time at which the signal, which is currently being picked up, was recorded.[2]

Now returning to the specific form of $V(t)$ given by (9) and looking at Fig. 1 we are having two obvious results:

$$\begin{aligned} T(t) &= L/v_0 \quad \text{if } t \le 0 \\ T(t) &= L/v_1 \quad \text{if } t \ge L/v_1 \end{aligned} \tag{11}$$

For $0 \le t \le L/v_1$ (and this is specifically the case shown in Fig. 1) to find the total time $T(t)$ we have to add the time duration corresponding to the right semiplane part of $T(t)$ and the one corresponding to the left semiplane part:

$$T(t) = t + \frac{L - v_1 t}{v_0} = \frac{L}{v_0} + \left(1 - \frac{v_1}{v_0}\right) t \tag{12}$$

[1]Note that this interpretation and Fig. 1 itself are not limited to the specific shape of $V(t)$ defined by (9), but apply for arbitrary $V(t)$.

[2]Of course, (10) could have been directly obtained from (6). By obtaining it using Fig. 1 instead, we have given an intuitive interpretation to (10).

(note that (12) gives $T(0) = L/v_0$ and $T(L/v_1) = L/v_1$, the same values as given by (11), corresponding to the fact that $T(t)$ must be continuous).

Thus $T(t)$ varies linearly on the transition range $t \in [0, L/v_1]$. More specifically, $T(t)$ changes from the old steady-state delay time to the new one and the transition duration is equal to the new steady-state time. This change produces the commonly known pitch jump effect from a sudden change of the tape speed. This jump has an obvious explanation in terms of tape speed, but now we can also explain in in terms of delay time. The duration of the pitch-shifting transition is exactly equal to the new steady-state delay time, thus, if feedback is present, the pitch-shifted signal will be recorded back into exactly one echo period of the "new steady-state", staying in the feedback loop until it decays or until a new speed change occurs.

3. A NUMERICAL SOLUTION FOR ARBITRARY VARIATIONS IN SPEED

If $v(t)$ is not known in advance, we can't compute (10) analytically and need to develop a numerical method. We want this method to be usable for practical delay implementations, therefore we want it to be computationally efficient and not suffer from the drift problem explained in Sec. 2.1.

3.1. Properties of a digital ringbuffer

We intend to implement a "tape delay" by combining a numerical solution of the tape-equation with a variable-length ringbuffer-based delay. Before we continue to discuss the solution of the tape equation, it is helpful to address some details regarding ringbuffer-based delays. The following discussion assumes that the sampling period and, respectively, the sampling frequency are unity: $f_s = 1$.

Consider the range of delay times supported by an ordinary ringbuffer-based delay. Clearly there is maximum delay time, limited by the ringbuffer's capacity. The minimum delay time is in principle zero. So, we have

$$0 \le T(t) \le T_{\max} \tag{13}$$

However there are two factors further limiting that range.

The first factor is interpolation, which is necessary to support delay times which are not an integer number of samples. Most interpolators need to consider some samples both before and after the interpolation point. Thus (13) (for a symmetric interpolator) turns into

$$\max\{\Delta - 1, 0\} \le T(t) \le T_{\max} - (\Delta - 1) \tag{14}$$

where Δ is half-width of the interpolator's kernel. This limitation can be however worked around by reducing the interpolator's order and/or window size when the interpolation is done at the edges of the range. This might be particularly desirable for $T \approx 0$, e.g. if we're looking for a comb filtering effect.

Another factor affecting (13) and (14) appears if the delay is used inside a feedback loop. If the ringbuffer is structured so that output and input happen synchronously in the algorithm, a unit delay will implicitly be introduced into the loop, as the current output of the delay is never known when calculating the input. (Fig. 2). For an ordinary digital ringbuffer-based delay this solely means that the delay time is off by one sample, which can be either tolerated or compensated by adjusting the offset of the read head

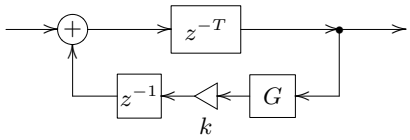

Figure 2: Unit delay in delay's feedback loop. k is feedback amount. G denotes some additional processing (not necessarily linear) which might occur in the loop.

by one sample. However if the ringbuffer delay is used as a basis for a tape delay emulation, either of the two mentioned options will distort the tape equation's solution and thus might potentially break the exact-repetition nature of the tape delay feedback in case of modulated tape speed. Therefore we would rather avoid the introduction of the extra unit delay altogether.

This can be achieved by splitting the processing of the delay's sample tick into two separate parts: the reading and the writing part. The read is processed first, then the feedback path, then the writing part. This eliminates the extra unit delay. In this case, however, the current input sample of the delay is not being written into the delay buffer until the end of processing and thus is not available for the read interpolation. This increases the lower boundary of supported $T(t)$ by one sample compared to (14):

$$\max\{\Delta, 1\} \le T(t) \le T_{\max} - (\Delta - 1) \tag{15}$$

unless we would be willing to solve the implicit equation arising out of the instantaneous dependency of delay's output signal on delay's input.

In this paper we will assume that the tape delay is to be used in a feedback loop and will develop the algorithm details under the assumption of split read/write processing.

3.2. Tape equation variables in discrete time

Let n be the discrete time-index. Since we assumed that the sampling period is unity, we have: $t = n$. Let's also choose the length scale so that the distance L between the heads is also unity: $L = 1$. The tape speed v expressed in these units means the fraction of the distance between the heads travelled over one sample period.

In order to be able to numerically apply the tape equation we need to keep the information about the tape speed values in the past, where the time range of interest is $[t, t - T(t)]$, where t is the current time. In Sec. 2.1 we gave reasons to choose the difference form (6) of the tape equation as the basis of the numerical solution, therefore, rather than storing the past values of $v(t)$, we will store the past values of its integral $V(t)$.

We want to use $T(t)$ obtained via the tape equation to control the delay time of a ringbuffer-based digital delay. It is a natural choice therefore to extend the ringbuffer elements to also contain the values $V[n]$ along with the stored audio signal samples. The value $V[n]$ will be contained in the same element as the audio signal recorded by the delay at time n. Intuitively, the write head simultaneously records the audio signal and the signal $V(t)$ onto the tape. Assuming that V is defined by

$$V(t) = \int_0^t v(\tau)\,\mathrm{d}\tau$$

or, in discrete time

$$V[n] = \sum_{i=0}^{n} v[i] \tag{16}$$

we can compute $V[n]$ incrementally

$$V[n] = V[n-1] + v[n] \tag{17}$$

Note that (16) and (17) imply that $v(t) = v[n]$ is assumed to be constant over a duration of one sample period, which is a common simplification when dealing with changing control values in discrete time. We will work further under this assumption, unless otherwise noted.

3.3. Representation of tape coordinates

$V[n]$ is an infinitely growing sequence, and thus there are dangers of increasing precision losses and/or overflow in (17), if floating point representation is used. Instead of trying to estimate whether this could be an issue with practical sampling rates and running times, we are going to use fixed point representation which will provide an elegant way to avoid such concerns altogether. We will be using this fixed point representation for $V[n]$ and any other values expressing the tape coordinate or derived values such as tape speed.

(15) effectively provides the upper and the lower bound to $v[n]$:

$$\max\{\Delta, 1\} \le 1/v[n] \le T_{\max} - (\Delta - 1)$$

that is

$$0 < \frac{1}{T_{\max} - (\Delta - 1)} \le v[n] \le \frac{1}{\max\{\Delta, 1\}} \le 1 \tag{18}$$

Under the restriction (18) our fixed point numbers will need to have a sign bit and two integer bits,[3] the remaining bits are to be used for the fractional part. Thus from a 64 bit integer we'll make a 2.61 signed fixed point number. So we'll have better precision than if we used 64-bit IEEE 754 floats, which have only 52 fractional bits of mantissa. The fixed point representation also gives constant precision across the entire value range, which is more appropriate for our purposes.

At any particular time the integral (3) and respectively the difference (6) are dependent only on the history within time range $[t - T(t), t]$. Thus, the tape coordinate values, which we are interested in are contained within the range $[V(t) - L, V(t)]$ (or marginally outside). Since $L = 1$ and $t = n$, this range can be written as $[V[n] - 1, V[n]]$. This suggests that we should be fine using fixed point arithmetic modulo 8 for the computations involving tape coordinates.

Under the assumption of two's complement binary representation of 64-bit integers, arithmetic computations modulo 8 will occur automatically for 64-bit integer-based 2.61 fixed point numbers. More precisely, the addition, subtraction and multiplication will be automatically done modulo 8, while comparisons will require special care.

Instead of simply comparing two numbers for $>$, $\ge$, $<$ or $\le$, we will need to compare their signed difference to zero, e.g.

$$(a > b)_{(\mathrm{mod}\ 8)} \overset{\mathrm{def}}{\iff} a - b > 0 \tag{19}$$

[3] We will use the fixed point representation not only for speed, but for anything which includes length into its dimension. That is the reason to have the extra bits in the integer part, which should become more clear through the discussion that follows.

This way the comparison is made "on the shortest path" between a and b (or, more precisely, between elements of the respective congruence classes).

3.4. Ringbuffer index arithmetic

Ringbuffer indexing also uses modulo arithmetic. A standard ringbuffer implementation generally needs only index increments, in which case modulo arithmetic technically means doing a trivial wraparound. Usually such wraparound is implemented either by a conditional check, or, if ringbuffer size N is a power of 2, by bitmasking with $N-1$. For the purposes of this algorithm the index arithmetic will need more of the modulo techniques.

Assuming ringbuffer indices n always take values within the range $[0, N-1]$, let's introduce the concept of a modular range of ringbuffer indices:

$$[n_1, n_2)_{(\mathrm{mod}\ N)} = \begin{cases} [n_1, n_2) & \text{if } n_2 \geq n_1 \\ [n_1, N) \cup [0, n_2) & \text{if } n_2 < n_1 \end{cases}$$

The length of the range is thus

$$\left|[n_1, n_2)_{(\mathrm{mod}\ N)}\right| = \begin{cases} n_2 - n_1 & \text{if } n_2 \geq n_1 \\ n_2 - n_1 + N & \text{if } n_2 < n_1 \end{cases}$$

If N is a power of 2, then, under the assumption of two's complement binary integer representation, modular range length can be computed without evaluating a conditional:

$$\left|[n_1, n_2]_{(\mathrm{mod}\ N)}\right| = (n_2 - n_1)\ \&\ (N-1)$$

where $\&$ denotes bitwise "and".

Notably, we can't use a comparison definition similar to (19) for ringbuffer indices, because we cannot assume that the comparison needs to be done "on the shortest path".[4] Therefore instead of index comparisons, we will be comparing the lengths of index ranges, which is a well-defined operation.

In binary search within the ringbuffer contents we will need to be able to find the middle of a modular range. Clearly, we can't simply take the average of the range's bounds, as the result could be off by $N/2$. Instead, we'll need to divide the range's length by 2 and use this new length to obtain the middle position and the new bounds.[5]

3.5. Tracking of the read position

In a ringbuffer-based delay implementation, the "write head" progressively cycles through the underlying array of the ringbuffer, advancing by one array index per sample. The position of the "read head" is computed each time anew, by subtracting the delay time (in samples) from the write head's position. In the proposed implementation of the "tape delay" we update the ringbuffer's read position incrementally instead. The read position must be fractional in order to support $T(t)$ which are not integer sample counts. We will therefore need to incrementally track the following values:

- the write position in the ringbuffer's array n_w
- the write head's tape coordinate x_w
- the read position in the ringbuffer's array $n_r + \nu_r$, where n_r is the integer part and $\nu_r \in [0, 1)$ is the fractional part[6]
- if ringbuffer size N is not a power of two, we might want to explicitly store the size of the ringbuffer contents (which is equal to $|[n_r, n_w)_{(\mathrm{mod}\ N)}|$) and update it with changes to n_w and n_r, thereby saving one evaluation of a conditional, when ringbuffer content size is needed.

We will further assume that the formal indexing of $V[n]$ is identical to the ringbuffer element indexing modulo N. We will also notate and understand the fractional indexing of $V[n]$ as

$$V(n_r + \nu_r) = V[n_r] + (V[n_r + 1] - V[n_r]) \cdot \nu_r \qquad (20)$$

Note that the linear interpolation in (20) is chosen because it is exactly correct given the previously stated assumption that $v(t)$ is constant over the duration of one sample.

As discussed in Subsec. 3.1, we wish to implement a delay usable in a feedback context, meaning that the reading of the audio output signal from the ringbuffer should occur prior to and separately from the writing of the audio input signal. Using (10) and Fig. 1 we can construct the following algorithm for processing a single sample step of such delay.

0. In the beginning of the step the variables are set like follows:
   - n_w is pointing to the ringbuffer element which is about to be written to
   - x_w contains the previous coordinate of the write head
   - $n_r + \nu_r$ is pointing to the ring buffer element which was read from in the previous step.
1. Compute the new value of x_w using (17) and (in agreement with (7a)) write it into $V[n_w]$. This doesn't depend on the delay's input signal value and therefore can be done in the beginning.[7]
2. Compute $x_r = x_w - L = x_w - 1$ (according to (7b)) and then search for the new $n_r + \nu_r$ such that
$$V(n_r + \nu_r) = x_r \qquad (21)$$
   The details of the search will be explained in Sec. 3.6.
3. Read the delay's output from the ringbuffer at position $n_r + \nu_r$.
4. Send the delay's output sample through the feedback loop all the way to the delay's input.
5. Write the delay's input into the ringbuffer at position n_w and advance n_w by one array index.

[4] With tape coordinate representation we have introduced additional headroom into the modulus to make sure that the distance between the values which we would want to compare or to subtract doesn't exceed half of the modulus. We could have introduced similar headroom into ringbuffer indices, but that would raise efficiency concerns.

[5] It could be useful to incrementally store the range's length in a separate variable during binary search.

[6] Actually, only n_r needs to be incrementally tracked, while ν_r will be computed each time.

[7] The fact that we compute the new value of x_w in the beginning and advance n_w in the end is matched in the later proposed approach to the algorithm initialization. If cache line aliasing between the read and write positions becomes a concern, we could perform the writing of x_w into $V[n_w]$ in step 5 instead (note that such change doesn't affect the mentioned initialization). However this excludes $V[n_w]$ from the allowed range of the search in step 2, effectively decreasing the upper bound of the tape speed in (18) and respectively increasing the minimum attainable delay time, unless the need to access $V[n_w]$ is handled "manually" during the search.

3.6. Updating of the read position

In step 2 of the "tape delay" processing algorithm introduced in Sec. 3.5 we need to perform a search for the solution of (21). We will split the search in two parts. First (the search itself) we search for n_r such that $x_r \in [V[n_r], V[n_r+1]]$. Having found n_r, we can solve (20) in respect to ν_r, obtaining:[8]

$$\nu_r = \frac{x_r - V[n_r]}{V[n_r+1] - V[n_r]} \tag{22}$$

According to (18) the sequence $V[n]$ is monotonic and we need to search only in the forward direction from the previous value of n_r. The simplest possible implementation of the search therefore is: repeatedly advance n_r by one array index, comparing $V[n_r]$ to x_r. This is actually not as bad as it may seem. Because in the most commonly occuring case, when $v(t)$ doesn't change much during $[t - T(t), t]$, or at least doesn't speedup noticeably, we will need to advance n_r only one or two times, until we found the new value of n_r. In case of a speedup by a factor of K we will however need to perform ca. K steps before we find the new position n_r. Thus, during speedup transitions, the operation count will increase by a factor of K.

This can be improved to an increase only by a factor of $\log_2 K$. Since $V(t)$ is monotone, it can be inverted by bisection [7], which in discrete time case effectively takes the form of binary search followed by the subsample position refinement at the end. Thus, we intend to do binary search within $V[n]$ between the old value of n_r and the new value of n_w.[9] In isolation, this is not such a good idea. Because now the binary search range contains the entire region of the tape between the read and write heads, and we are going to binary-search this entire range (performing ca. 10-20 search steps) each time, even in case of small speed variations. This results in not $O(\log K)$ computation complexity but rather $O(\log T(t))$.

We can, however, improve the selection of the search range. Starting from the old read position n_r we check the value $V[n_r + 2]$. If $V[n_r + 2] \geq x_r$, then we take $n_r + 2$ as the upper bound of the range and n_r as the lower bound. Otherwise we know that the new read position doesn't lie between n_r and $n_r + 2$ anyway, so we update n_r to take the value $n_r + 2$ and now take a step of 4, probing the value $V[n_r + 4]$ (formerly $V[n_r + 6]$) and taking the range from n_r to $n_r + 4$, if successful. Otherwise we update n_r to $n_r + 4$ and take a step of 8 etc, until we finally find the range containing x_r.

Notably, during this process of searching for the initial range we have to be careful not to cross the write position n_w. As mentioned before, we can't use a comparison approach like the one of (19) for ringbuffer indices. Therefore, instead of comparing n_r to n_w we need to compare the step size to the length of the modular range $[n_r, n_w)_{(\mathrm{mod}\ N)}$.

It's not difficult to see that the described way of searching for the initial range has computation complexity of $O(\log K)$ and so does the binary searching on the range found in this way, thus our entire implementation has $O(\log K)$ complexity.[10]

3.7. Initialization

The previous discussion of the tape delay algorithm was assuming that we are somewhere in the middle of the running time and all incremental variables are properly set by the previous sample's processing. However, how do we set these variables initially?

Let's assume that we are using linear interpolation for reading the audio signal at non-integer positions $n_r + \nu_r$. In this case we propose to initially let

$$\begin{aligned} V[0] &= -L = -1 \\ V[1] &= 0 \\ n_r &= 0 \\ n_w &= 2 \\ x_w &= 0 \end{aligned}$$

where $V[0]$ and $V[1]$ are stored in the ringbuffer's array elements 0 and 1, and the audio signal part of these elements is initialized to zero. Then, as long as $0 \leq x_w < 1$, the read head will interpolate between elements 0 and 1 of the array, thereby producing zero output, as if we were reading from a clean tape.[11]

If instead of linear interpolation we use e.g. cubic interpolation, then the same initialization idea will look like:

$$\begin{aligned} V[0] = V[1] &= -L = -1 \\ V[2] = V[3] &= 0 \\ n_r &= 1 \\ n_w &= 4 \\ x_w &= 0 \end{aligned}$$

Other interpolation schemes can be treated similarly.

3.8. Sparse storage of $V[n]$

The need to store $V[n]$ in addition to the audio in the ringbuffer significantly increases the memory usage by the delay. E.g. if the audio consists of two 32-bit float stereo channels, by adding a 64-bit fixed point $V[n]$ to each sample we double the amount of used memory.

The memory requirements can be reduced if the tape speed doesn't change on every sample, but at a lower "control rate". This could however introduce audible artifacts due to "steppy" pitch changes corresponding to the steppy nature of the tape speed. In such case, instead of considering the tape speed constant on the

[8] We should remember that we are using arithmetic modulo 8 when dealing with tape coordinates, and, strictly speaking, the division of two values both having the tape coordinate units in (22) hasn't been defined. This division doesn't require any special treatment though, since the distances from x_r to $V[n_r]$ and from $V[n_r]$ to $V[n_r + 1]$ on the tape coordinate axis cannot exceed $\max\{v[n]\}$, which according to (18) is 1.

[9] Remembering to use the previously discussed modular arithmetic rules, for both tape coordinate and index.

[10] Obviously, if $\log K < 1$ and especially if $\log K < 0$, the complexity bound $O(\log K)$ looks highly questionable. It's not difficult to realize that dropping K below 1 (tape slowdown) doesn't further reduce the number of computations. This suggests that the computation complexity estimation for arbitrary speed changes is, strictly speaking, $O(\max\{1, \log K\})$. For the simplicity of notation, however, we can agree to understand $O(\log K)$ as a somewhat informal way of writing $O(\max\{1, \log K\})$.

[11] Thus, we have a very quick initialization procedure, not requiring to fill the entire ringbuffer, neither with the values of $V[n]$ nor with the zero audio samples. This is quite useful if the implementation is used in a plugin in a DAW, where plugin reset times may become an issue.

range from $V[n]$ to $V[n+1]$ we could consider it linearly increasing.[12] The linear function (20) is thereby replaced by a quadratic one and (22) turns into a quadratic equation solution formula.

Of course, other than linear segments of $v[n]$ could be used, as long as they can be inverted (in reasonable computation time) to find the fractional position.

3.9. Antialiasing of speedups

The interpolated readout of the ringbuffer works well if the delay speed is almost constant or is slowing down. However a speedup effectively shifts the pitch of delay's signal up, thereby creating frequencies above Nyquist threshold, which ordinary interpolation doesn't try to suppress. It is, however, possible to reformulate the polynomial interpolation in a way allowing arbitrary cutoffs below Nyquist.

Any polynomial interpolation of a sequence y_n can be alternatively expressed as a convolution with a kernel:

$$\mathcal{L}[y_n](t) = \sum_n y_n \lambda(t-n)$$

where $\lambda(t)$ is the kernel and $\mathcal{L}[y_n]$ denotes a continuous time function which is the result of the interpolation of y_n. The interpolation's kernel λ is obtained by interpolating the Kronecker delta sequence with the interpolator in question:

$$\lambda(t) = \mathcal{L}[\delta_n](t)$$

and thus consists of polynomial segments. The kernel normally corresponds to a continuous-time lowpass filter with a cutoff close to Nyquist. We can lower the kernel's cutoff by a factor of $K \in \mathbb{R}$ by stretching the kernel K times along the time axis and simultaneously reducing its amplitude K times:

$$\mathcal{L}[y_n](t) = \sum_n y_n \frac{\lambda((t-n)/K)}{K} \tag{23}$$

Thus, by expressing the interpolation as convolution we can arbitrarily change the interpolation filter's cutoff, which allows us to use the interpolator to suppress unwanted frequencies below Nyquist.[13]

The speedup situation can be detected by comparing the reading speed (which is simply the current tape speed) to the speed at which the signal was recorded. According to (17) the recording speed is simply equal to $V[n_r] - V[n_r - 1]$.[14] Note, however, that the computation complexity of such antialiasing is $O(K)$, where K is the speedup factor. We could put an upper bound on the complexity by artificially clamping the factor K used in (23) at the obvious cost of some unfiltered aliasing in case K exceeds the clamping value.

[12] This might require storing $v[n]$ alongside $V[n]$, unless we want to guess $v[n]$ from neighboring values of $V[n]$.

[13] One has to take care to make sure that the interpolator, which is stretched along the time axis by the factor K, does not attempt to read ahead of n_w or too far behind n_r, where the samples either haven't been written into the buffer yet or have been overwritten with newer values. The issue can be addressed e.g. by reducing the cutoff factor K, if we get too close to the boundaries of the valid index range.

[14] The forward difference $V[n_r + 1] - V[n_r]$ also can be in principle taken, since (23) is only an approximation of a proper resampling process anyway.

The equation (23) is only a somewhat rough approximation, done under the assumption that the tape speed doesn't change very quickly, because it assumes that the samples y_n are equally spaced in time. In reality the samples are not traversed by the read head at equal time intervals, this happens at different times and at different speeds. A more correct version of (23) therefore might be

$$\mathcal{L}[y_n](t) = \sum_n y_n \frac{\lambda((t-\tau_n)/\max\{K_n, 1\})}{\max\{K_n, 1\}} \tag{24}$$

where τ_n is the time at which the sample y_n is traversed and K_n is the respective upsampling factor. Note that some of the times τ_n in (24) occur in the future. They still may be known in advance, if we can know the variations of the tape speed in advance, in which case (24) is still perfectly implementable.[15]

3.10. Extending the bounds of tape speed

The tape speed bounds imposed by (15) can be significantly extended.

If we are having no feedback or if we are willing to solve the implicit feedback equation, then (14) applies instead of (15) and we might want to support arbitrarily large speeds. Notably, we still have quite a lot of headroom which we could add into the proposed fixed point format. E.g. we could use 11.52 signed fixed point numbers, thereby increasing the upper boundary of the tape speed headroom (18) by a factor of 512 giving $v \leq 512$.

Conversely, as $1/v$ reaches the maximum capacity of the ring buffer (more precisely defined by (14) and (15)) we attain the maximum delay time possible with our proposed approach based on (10). At this point, however, we could add the ideas of the straightforward implementation of tape delay, which would correspond to slowing down the medium below the natural speed of the 1:1 ratio (the medium moves by 1 sample during one sample tick). In our setup this would correspond to advancing n_w by a fractional amount less than 1. The writing to the ringbuffer will be occurring "in between" the sample ticks with proper interpolation, or, if antialiasing is desired, with proper "downsampling filtering".

In this way we can achieve arbitrarily small and even zero tape speeds, corresponding to arbitrarily large to infinite delay times. In principle one could even go in negative speed direction.[16] A limitation of this approach is that in this case the bandwidth of the signal transmitted through the medium will be reduced, since the "tape's sampling rate" is lower than the outside sampling rate.

4. RESULTS

In order to get an idea of the performance of our method, we have made a comparison between an ordinary ringbuffer-based digital variable-length delay (VL), variable sample-rate delay (VS), Huovilainen's method (H) and the proposed method (P), all methods using SIMD-ified Catmull–Rom interpolation of two stereo

[15] We only need to know the future tape speed but not the future values of the audio signal. This means that (24) is implementable even for a feedback delay without introducing any additional latency into the feedback. The latency will be introduced only into how the delay responds to the speed control signal.

[16] As the speed goes to zero and negative values, $V(t)$ stops being strictly monotonic, and one needs to take additional decisions during the search for the solution of (21).

Method	x1	x2	x10	x100
(VL)	21	21	21	21
(VS)	46	58	163	1320
(H)	37	47	97	725
(P)	39	39	39	39
(P')	39	38	54	110

Table 1: Performance cost (in TSC clocks per processed sample) of different algorithms at different tape speeds.

Method	x1	x2	x10	x100
(VL)	23	75	277	2531
(VS)	48	112	419	3810
(H)	39	101	353	3235
(P')	41	92	310	2620

Table 2: Performance cost of different algorithms during speedups, with enabled antialiasing.

channels of audio[17]. The comparison is presented in Table 1 where (P') is the performance of the proposed method in the case of a speedup from 1x to the specified speed. The measurements were taken by letting the algorithm run for a large number of samples at different equivalent tape speeds, each time taking the average number of TSC[18] clocks per processed sample. The relative measurement error is ca. 5%. The 1x tape speed corresponds to 1 tape 'sample' processed per 1 outside sample. The identical performance costs of the proposed method at x1 and x2 speeds are due to the fact that the bisection method is searching over the same initial range of two entries in both cases.

We also measured the performance of the proposed method with enabled antialiasing of speedups. Whilst not measured directly, we have assumed that the antialiasing overhead would be identical between different methods and added the same overhead to other measured results.[19] The respective performance comparison is presented in Table 2, where the tape speed of the delay line is being switched from x1 to the specified speed.

5. CONCLUSION

In this paper, we introduced the tape equation, which allows the translation of variations of the medium speed into variations of delay time, thus allowing to implement tape-like modulation behavior using ordinary digital delays. In certain cases, when the speed variation pattern is known in advance, this translation can be done analytically. We have also introduced a numerical method to be used in cases where analytical solution is not possible. Compared to previous methods, which have $O(v)$ time complexity, the presented method has mostly $O(1)$ time complexity, except during speedup transitions, where the complexity is $O(\log K)$. If antialiasing of speedups is desired, the time complexity of speedup transitions grows to $O(K)$, however, the complexity can be bounded by artificially clamping the antialiasing filter's cutoff.

The proposed numerical method is also exact in the sense that the error from time discretization manifests solely as the tape speed being assumed constant over the duration of each sample, whereas precision losses occur only in the quantization of the speed values and in the final computation of the subsample read position for the ringbuffer (where it's totally negligible). Thus, all error is effectively contained in the time- and level-quantization of the tape speed, the solution of the tape equation itself being exact. Furthermore, the time-quantization error can be further improved by assuming linearly changing speed during each sample.

The proposed method can be used in implementations of delays where tape-like modulation behavior is desired. Compared to previously used approaches, our method has comparable or better CPU load at all delay times.

[17] Audio samples of each of these methods are available at the accompanying website: `https://github.com/julian-parker/DAFX-Tape`

[18] Time Stamp Counter, a processor's internal high-precision timer.

[19] In (VS) the antialiasing will need to be done constantly unless we take additional effort to store the information about the recording speed. In other methods this information is already available, thus the antialiasing may be done just during the speedups.

6. REFERENCES

[1] A. Huovilainen, "Enhanced digital models for analog modulation effects," in *Proc. Int. Conf. Digital Audio Effects (DAFx-05)*, Madrid, Spain, 2005, pp. 155–160.

[2] C. Raffel and J. O. Smith, "Practical modeling of bucket-brigade device circuits," in *Proc. Int. Conf. Digital Audio Effects (DAFx-10)*, Graz, Austria, Sep. 2010, pp. 50–56.

[3] M. Holters and J. D. Parker, "A combined model for a bucket brigade device and its input and output filters," in *Proc. Int. Conf. Digital Audio Effects (DAFx-18)*, Aveiro, Portugal, 2018.

[4] S. Arnardottir, J. S. Abel, and J. O. Smith III, "A digital model of the Echoplex tape delay," in *Proc. of the 25th Audio Engineering Society Convention*. Audio Engineering Society, 2008.

[5] J. Dattorro, "Effect design, part 2: Delay line modulation and chorus," *Journal of the Audio Engineering Society*, vol. 45, no. 10, pp. 764–788, 1997.

[6] D. Rocchesso, "Fractionally addressed delay lines," *IEEE Transactions on Speech and Audio Processing*, vol. 8, no. 6, pp. 717–727, 2000.

[7] W. H. Press, S. A. Teukolsky, W. T. Vetterling, and B. P. Flannery, *Numerical recipes in C*, vol. 2, Cambridge University Press, 1996.