AN ANALYSIS OF STARTUP AND DYNAMIC LATENCY IN PHASE VOCODER-BASED TIME-STRETCHING ALGORITHMS

Eric Lee, Thorsten Karrer, and Jan Borchers
Media Computing Group
RWTH Aachen University
52056 Aachen, Germany
{eric, karrer, borchers}@cs.rwth-aachen.de

ABSTRACT

The phase vocoder has become a popular method for time-stretching audio (altering its play rate without changing its pitch) in recent years. Despite continuing improvements to the algorithm itself for enhanced audio quality, the latency introduced by the processing is less well-understood. Such an understanding is crucial for accurate synchronization in the context of a larger interactive multimedia or computer music system. Our analysis shows that the phase vocoder has an effective startup latency of $2\,(R_a - R_s)$, and a dynamic latency (in response to rate changes) of $2R_s$, where R_a and R_s are the input and output hop factors used for time-stretching.

1. INTRODUCTION

Computers and processing capacity continue to advance at rates that exceed Moore's original prediction in 1965 [11]. Certain types of processing that were once a fantasy are now possible to perform in real-time. One example is using the phase vocoder for altering the play rate of an audio stream while preserving its original pitch (a process also known as *time-stretching*) – while it was originally developed in 1966 [3], it wasn't until recently that real-time implementations became possible [6].

Any non-trivial processing of signals will introduce some degree of latency. If this latency is small, it can usually be ignored without any significant impact on the system behavior, and this is often assumed in many interactive media and computer music systems today. The most obvious artifact of improperly handling latency in a system is a loss of synchronization between, for example, the audio and video.

Two recent trends in multimedia systems and computer technology, however, motivate the need for a re-examination of processing latency for these systems.

Firstly, computers are increasingly being used for professional multimedia applications, replacing both specialized and expensive hardware. A professional studio VTR (video tape recorder) capable of frame-accurate synchronization, for example, can cost upwards of ten thousand dollars. Television and film production studios are slowly migrating to digital production – *Star Wars II, Attack of the Clones*, for example, was the first major Hollywood film to be captured digitally, rather than on film [10]. More recently, even media companies, such as *Current TV*, a news broadcaster in the United States, have moved away from tape to a completely digital and computer-based production pipeline [15]. This trend requires system designers to migrate to what Greenebaum [4] refers to as a "sample-accurate" mentality when dealing with latency, rather than the current "best-effort" one.

Secondly, with the increased availability of computing power, it is now possible to incorporate increasingly complex processing and still maintain real-time performance. Interactive conducting systems such as our *Personal Orchestra* family [9], for example, employ a multitude of processing to recognize gestures, stream compressed audio and video from disk, and time-stretch the audio – all in real-time. More specifically, let us compare the complexity of an audio resampler, which was employed in an early version of *Personal Orchestra* – a resampler requires a few tens of multiply-add operations per output audio sample. In contrast, PhaVoRIT, a phase vocoder-based algorithm employed in our latest system [6], performs the time-stretching in real-time, but requires many orders of magnitude more processing per output audio sample. An unfortunate side-effect of this increased complexity in processing is increased latency.

We will divide our discussion of latency into two aspects: *startup latency* and *dynamic latency*. Startup latency is introduced when the filter is initially fed with data – many filters require some "priming" before they can begin to produce output. A 64-point sinc kernel used for resampling an audio signal, for example, requires the first 32 samples of input data before it can produce the first output sample. If these samples are being streamed from a real time data source, this introduces a 32 sample latency at startup (see Figure 1). Dynamic latency occurs when filter parameters (for example, the resampling factor) are changed; if the filter cannot respond immediately to a parameter change, latency will be introduced. Resampling using a sinc kernel has, for example, zero dynamic latency – it is theoretically possible to immediately switch from a resampling factor of 0.5 to 2 from one output sample to the next. In contrast, a phase vocoder algorithm is limited to rate changes at specific block intervals defined

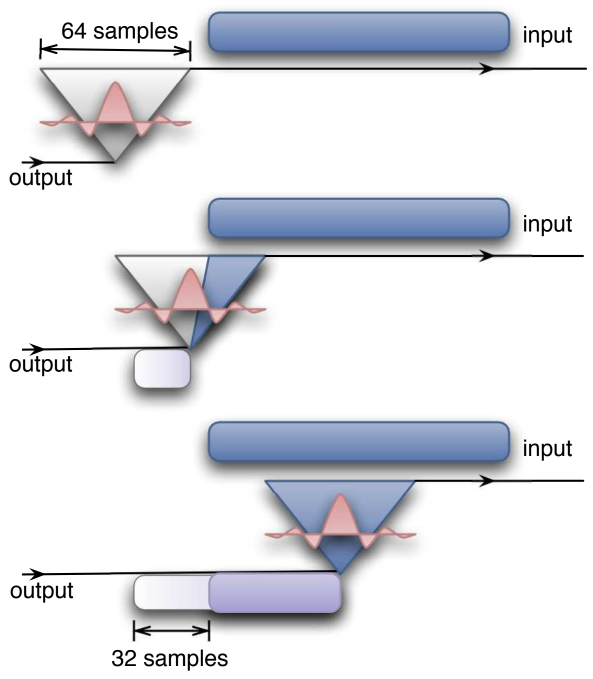

Figure 1. Latency introduced by a 64-bit sinc kernel for resampling. A sinc kernel interpolation filter will start producing output samples as soon as input is available. The first 32 output samples, though, do not correspond to any of the input samples – this is the startup latency. The samples corresponding to the input do not show up until sample 33 of the filter's output.

by the block size used for processing. Moreover, as we will show in this paper, there is a non-zero latency in response to rate changes.

In an interactive media system, an accurate handling of dynamic latency is critical, as the error introduced by improper handling has the potential to create a cumulative error that worsens over time. Let us consider again an interactive conducting system such as *Personal Orchestra*, where rate changes occur on the order of ten times per second, or more. An error of just 0.1 ms (just over 4 samples of audio sampled at 44.1 kHz) per rate change, can result in a worst case cumulative error of 100 ms in under two minutes, which is sufficient to produce a noticeable loss of synchronization between audio and video [2].

This paper is structured as follows: we will begin with a brief summary of the phase vocoder algorithm to provide the context for this work. Next, we will propose a number of schemes for mapping the timeline of time-stretched audio back to the original input. The results from this discussion will then be used to analyze the startup and dynamic latency of phase vocoder-based algorithms. We will conclude with a discussion of design implications.

2. RELATED WORK

To the best of our knowledge, no similar discussion of latency in phase vocoder-based time-stretching algorithms, or even any time-stretching algorithms of similar complexity, exist. [14] describes some of the challenges of synchronizing audio time-stretched using the phase vocoder to an external clock. However, the discussion is limited to calculating an appropriate input hop factor for input to the phase vocoder, and the fact that rate changes are limited to block boundaries.

DIRAC is a time-stretching algorithm that uses wavelet-based processing [1]. The DIRAC SDK documentation claims that the processing framework has zero processing latency. However, "processing latency" is defined to be the *startup* latency when the time-stretcher is set to its nominal rate; we feel this is an oversimplification, as the processing latency of a filter of this complexity is almost certain to be parameter dependent (as we will show in this paper for phase vocoder-based algorithms). Moreover, the document contains a subsequent discussion that describes how there is no way to predict which, or even how many, input samples are required to produce a specific block of output samples; this supports the conclusion that the input to output sample mapping is, in fact, non-trivial. The documentation for élastique, another commercial time-stretcher [17], offers a similar discussion, and also implies that the input to output sample mapping is non-trivial because of internal buffering of the input data.

There may be multiple reasons for this lack of rigorous discussion of latency in time-stretching: from a signal processing perspective, it is not clear how a time-stretching signal can be interpreted, since the nature of the processing "smears" a single sample from the original signal across a range of output samples. This smearing is frequency dependent, resulting in a reverberation-like effect. There is also no mathematically "correct" answer to time-stretching of arbitrary signals on which to base such an analysis; current work on time-stretching algorithms aim to minimize the perceptual artifacts introduced by the processing to produce a psychoacoustically pleasant result [6]. Finally, such a discussion becomes important only when sample-accurate synchronization is required, or when there are frequent changes to rate. The former is a topic that is typically neglected in software systems, as discussed by Greenebaum [4]; the latter seldom occurs in traditional multimedia systems such as video editing, where the stretch factor is typically held constant over a long period of time (one use for time-stretching in video editing would be to fit, for example, ten seconds of audio into nine). In interactive media systems, however, rate adjustments occur orders of magnitude more frequently, and so accumulation errors from an improper treatment of dynamic latency also manifest that much more quickly.

3. THE PHASE VOCODER IN A NUTSHELL

The phase vocoder has become popular algorithm in recent years for time-stretching (altering the play rate of audio without changing its pitch), as it is able to produce much higher quality results over a wider range of stretch factors than existing algorithms such as time-

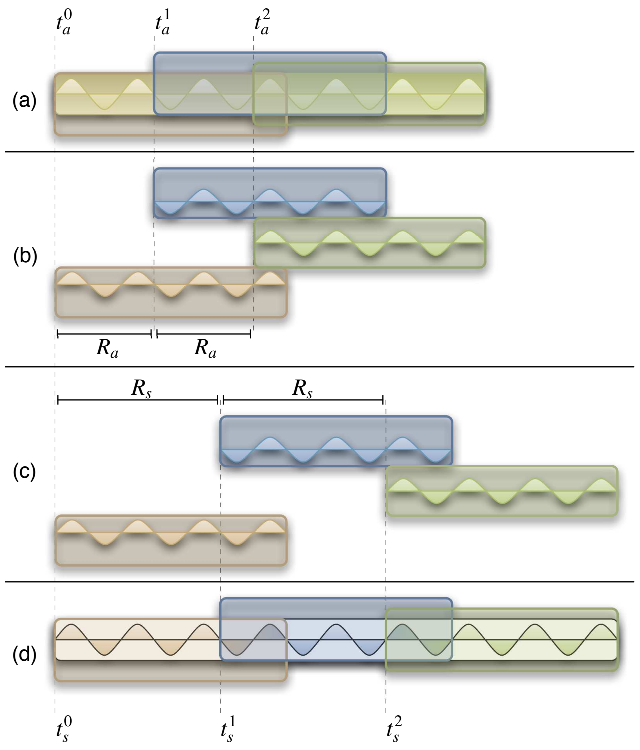

Figure 2. Illustration of time-stretching by dividing the audio into overlapping windows (a), respacing them (b, c), and overlap-adding them (d). The resulting audio has a rate of $r = \frac{1}{\alpha} = \frac{R_a}{R_s}$.

domain harmonic scaling (TDHS) [12] and waveform similarity overlap-add (WSOLA) [16]. The algorithm is described in detail in existing literature [3, 7] – the goal here is to provide a brief summary of how the phase vocoder works to set the context for our subsequent discussion.

3.1. The Concept

Audio can be time-stretched similar to how the length of a telescope or car radio antenna can be changed – the audio is divided into overlapping blocks that are spaced apart by some interval R_a (the analysis, or input, hop factor), and then reassembled with a different spacing R_s (the synthesis, or output, hop factor, see Figure 2). The resulting audio has a rate of $r = \frac{1}{\alpha} = \frac{R_a}{R_s}$ with respect to the original.

While this naïve method fulfills the basic requirements of time-stretching – namely, the play rate is changed without altering the pitch – the resulting audio will have significant artifacts. These artifacts are phase discontinuities caused by respacing the blocks of audio. For example, the frequencies f of two subsequent blocks will cancel each other out if $\frac{1}{2f}(2k+1) = R_a - R_s$ $(k \in \mathbb{Z})$, as they will be 180° out of phase when they are accumulated.

One way to solve the problem of phase jumps in the time-stretched signal is to adjust the starting phases of every block in the time-stretched signal to match the phases of the previous block at the overlap point. This would have to be done for all of the partials (dominant frequencies) in a block, and the problem is determining the phases of all of the partials at this overlap point. The starting phases of a block are known, and so is synthesis spacing R_s; the last factor for calculating the phases is the exact instantaneous frequency of each partial, which could be measured using a bank of bandpass filters.

3.2. The Algorithm

The phase vocoder algorithm uses this basic strategy to perform time-stretching of audio signals. Amplitude and phase information for all frequency channels at specific analysis points $t_a^u = u \cdot R_a$ $(u \in \mathbb{N})$ of the input audio signal are obtained using the short-time Fourier Transform (STFT). The STFT partitions the input signal into blocks using a window function of length N, and then transforms these blocks into the frequency domain using a Fast Fourier Transform (FFT). Note that the window function represents the impulse response of the bandpass filters (shifted to the base band) of an equivalent filter bank, and thus the calculated amplitudes and phases are an estimate for only the time point at the center of the block.

$$\begin{aligned} X(t_a^u, \Omega_k) &= \sum_{t=-N/2}^{N/2-1} h(t - t_a^u) x(t) e^{-j\Omega_k t} \\ &= |X(t_a^u, \Omega_k)| \cdot e^{j\angle X(t_a^u, \Omega_k)} \end{aligned} \tag{1}$$

In the frequency domain, the phases are adjusted to fit the audio at the new spacing:

$$\angle X'(t_s^u, \Omega_k) = \angle X'(t_s^{u-1}, \Omega_k) + R_s \hat{\omega}_k(t_a^u) \tag{2}$$

The instantaneous frequency $\hat{\omega}_k(t_a^u)$ of the kth frequency channel can be approximated by looking at the deviation $\Delta\Phi_k^u$ of the actual phase increment between two consecutive analysis points t_a^{u-1} and t_a^u from what we would expect when looking at the channel's center frequency Ω_k:

$$\Delta\Phi_k^u = \angle X(t_a^u, \Omega_k) - \angle X(t_a^{u-1}, \Omega_k) - R_a \Omega_k \tag{3}$$

Since that deviation happened over the duration of R_a we can determine the frequency deviation from Ω_k and thus the instantaneous frequency $\hat{\omega}_k(t_a^u)$:

$$\hat{\omega}_k(t_a^u) = \Omega_k + \frac{\Delta_p \Phi_k^u}{R_a}, \tag{4}$$

where $\Delta_p \Phi_k^u \in [-\pi; \pi]$ is the principal determination of $\Delta\Phi_k^u$.

Now, the pairs of amplitudes $|X(t_a^u, \Omega_k)|$ and adjusted phases $\angle X'(t_s^u, \Omega_k)$ are transformed back to blocks of samples using an inverse FFT and a synthesis window function. These blocks are inserted at the synthesis points $t_s^u = u \cdot R_s$ $(u \in \mathbb{N})$, and overlap-added to produce the time-stretched audio signal.

3.3. Improvements to the Phase Vocoder

The basic phase vocoder described above introduces a number of reverberation and transient-smearing artifacts in the time-stretched audio. Subsequent work has attempted to address these issues [6, 7, 13]. However, they are all based on the basic phase vocoder algorithm described above, and do not impact our discussion here.

4. INTERPRETING TIME-STRETCHED AUDIO

Before we can begin to analyze the perceived latency of rate changes, we need to examine how one can interpret the timeline of time-stretched audio. Note that our goal here is not to determine an exact, sample-accurate mapping from output samples to input samples. Such an analysis is not practically feasible, since it is unclear from a mathematical perspective what it means to "time-stretch" a signal. The nature of the phase vocoder processing introduces a frequency-dependent smearing in the signal similar to reverberation, and thus the different frequency components of a particular time instant of input audio may become smeared across an interval in the output. Moreover, some of the proposed transient detection and processing schemes, such as [13], add an additional non-linear distortion to the time information of the time-stretched signal.

Instead, our aim is to provide a means of interpreting the timeline of time-stretched audio in a way such that the error is bounded, and, more importantly, does not accumulate over time. Our task is, given a block of output samples, to determine the corresponding samples in the input audio. We cannot establish this mapping by simply counting the input samples that have been requested by the time-stretcher, since it is common for filters of such complexity to pull ahead and buffer a certain amount of input data. We will use $\tau(t_s)$ to refer the input time that corresponds to an output (synthesis) time t_s; t_a is, again, the input (analysis) time.

4.1. Black-Box Approach

The simplest, albeit naïve, approach is to maintain a counter of the current input position. The input position, τ, would be incremented for each output block that is produced by the size of that block, M, scaled by the requested play rate:

$$\tau_j = \tau_{j-1} + \frac{M}{r} \; ; \; \tau_0 = 0 \tag{5}$$

The problem is that the increment factor, $\frac{M}{r}$ is only an estimation that is based on r. Recall that the rate at which the audio is time-stretched is represented by the ratio $\frac{R_a}{R_s}$, and both R_a and R_s are two integer numbers; thus, the *actual* rate at which the audio is time-stretched will not be exactly r. Moreover, rate changes can only be made at specific intervals defined by the output hop factor R_s. Since $R_s \neq M$ in the general case, a rate change that is requested in the middle of a block will not take effect until the next block. Finally, a requested rate change does not take effect immediately, as we will demonstrate in the discussion of dynamic latency in Section 6. Regardless, it is sufficient for the time being to realize that the calculation above will always introduce a small amount of error, and that this error will accumulate over time. This accumulation error also means that synchronization will be lost over time, and will become increasingly worse.

4.2. Hop-Factor Approach

To produce a better result than the one described above, we must examine how the phase vocoder produces time-stretched audio. Recall, as described in Section 3, that the phase vocoder operates on sample windows size N that are then overlap-added to produce the output blocks. In a real-time system, the output block spacing R_s is typically held constant, and time-stretching occurs by respacing the input blocks. For illustration purposes, we will use the specific example where each sample window has a length of 8 time units, and the output hop factor is fixed at 2 time units (a 75% overlap at the output, see Figure 3).

An improved approach uses the input hop factor to determine the input time:

$$\tau_u = t_a^u \tag{6}$$

For example, to create the output for $u = 4$, samples starting at the input time $t_a^4 = 10$ were fetched from the input and processed. Thus, it would seem reasonable to say that the starting time of output block 4 corresponds to time $\tau = 10$. This scheme, which we proposed in [9], is accurate enough for many applications; it was, for example, used in our third generation *Personal Orchestra* system [9]. It does, however, require an internal knowledge of how the data is buffered and how the input and output hop factors are calculated, information which is typically not available as a client of a time-stretcher module. One important characteristic of (6) is that the error does not accumulate, since the current value of τ does not depend on previous computations, as in (5).

4.3. Overlap-Add Approach

The scheme presented above, however, does not take into account the fact that each output block produced by the phase vocoder is the result of an overlap-add with the three preceding processed blocks. One could argue that this overlap-add results in an "averaging" effect, which reduces artifacts in the processed audio, but also results in a "smear" of the timeline. Moreover, these blocks are windowed with a Hanning (or similar) window during processing – which means that at the start of output block 4 in Figure 3, the input sample at time $t_a^4 = 10$ doesn't even contribute to the actual output!

This problem was uncovered during the design and implementation of DiMaß [8], a technique for audio scrubbing using the phase vocoder for feedback. With DiMaß, changes to the play rate are both more diverse and frequent

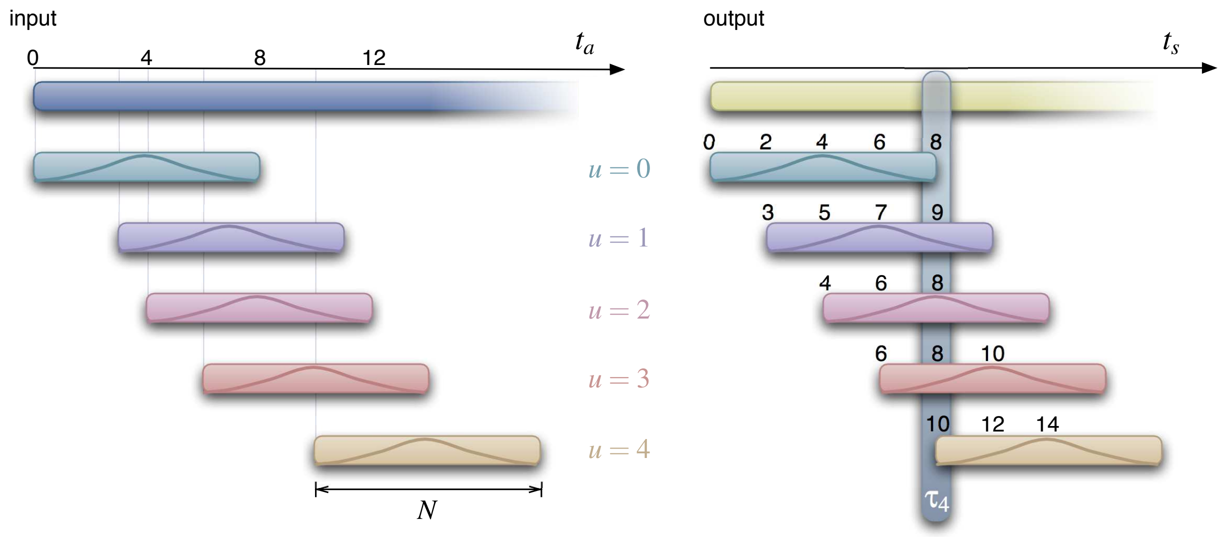

Figure 3. An interpretation of time-stretched audio. Two approaches are possible: in the first, only the input hop factor is considered, resulting in $\tau = 10$; in the second, the overlap-add nature of the algorithm is considered, and $\tau = 8.25$.

than with *Personal Orchestra*, and especially at the slow scrub rates, the error, while bounded, becomes noticeable.

We developed a solution that uses a weighted average of the time stamps of the samples that are being summed together to produce the output. The weights, $h(n)$, are determined by the STFT window (which has length N), and the start time of block is thus a weighted sum of the times with the previous four blocks (see Figure 3):

$$\tau_u = \frac{\sum_{i=0}^{4} h(\frac{i \cdot N}{4}) \cdot \left(t_a^{u-i} + \frac{i \cdot N}{4}\right)}{\sum_{i=0}^{4} h(\frac{i \cdot N}{4})} \tag{7}$$

Using, again, our example of $u = 4$ and a Hanning window for $h(n)$, we obtain:

$$\tau_4 = \frac{0 \cdot 10 + 0.5 \cdot 8 + 1 \cdot 8 + 0.5 \cdot 9 + 0 \cdot 8}{0 + 0.5 + 1 + 0.5 + 0} = 8.25$$

The astute reader may observe that this interpretation has a major flaw: namely, the STFT does not, from a mathematical perspective, preserve time intervals. Let us take, for example, block 0, which starts at time $t_a = 0$ and ends at $t_a = 8$. We assumed in (7) that, after processing, the output block also starts at $t_a = 0$ and ends at $t_a = 8$. However, by definition of the Fourier transform, the act of transforming the block into the frequency domain destroys all temporal information.[1] A more correct interpretation would thus be to set the *entire* output block to time $t_a = 4$, the time at the center of the input block. The subsequent windowing and overlap-add introduce an averaging that restores the continuity of the timeline at the output. Applying this interpretation to the scenario in Figure 3 results in the following:

[1] An exception is if the block was transformed into the frequency domain, and then immediately back into the time domain. However, this defeats the purpose of performing the Fourier transform in the first, place, and is certainly not applicable in the general case!

$$\tau_u = \frac{\sum_{i=0}^{4} h(\frac{i \cdot N}{4}) \cdot \left(t_a^{u-i} + \frac{N}{2}\right)}{\sum_{i=0}^{4} h(\frac{i \cdot N}{4})} \tag{8}$$

Repeating our calculation of τ_4 using (8) yields:

$$\tau_4 = \frac{0 \cdot 14 + 0.5 \cdot 10 + 1 \cdot 8 + 0.5 \cdot 7 + 0 \cdot 4}{0 + 0.5 + 1 + 0.5 + 0} = 8.25$$

This result is identical to that given by (7). It is, in fact, not difficult to show that the two interpretations always give the same results when the output hop factor R_s is fixed (which is usually the case with implementations for real-time systems).

4.4. Other Considerations

Our scheme could be further improved by taking into account, in the analysis, the group and phase delay of the filters that perform the phase re-estimation and transient processing; however, for our purposes, we have found the above scheme to be sufficient with respect to accuracy, and we reserve such an analysis for future work.

One further consideration is computing values of τ in the middle of a block, since equations (6) and (7) are only valid for the block boundaries. Since the rate is constant for each block, we feel it is sufficient to simply compute the values for τ at the start and end of a block, and perform linear interpolation to obtain the value for τ for in-between values of t_s.

5. STARTUP LATENCY

It is typically desirable to specify a starting point in the audio at which to begin producing time-stretched output. In an audio editor, for example, the user sets the cursor to a specific part of the audio waveform, and the audio begins playing from this position. For these type of applications, it is critical that the audio starts exactly where the user has specified, so that the audio is consistent with the visual waveform representation.

This problem is often known as "startup synchronization" in multimedia systems, and has been studied before in existing literature [5]. Here, we discuss the additional complexity that results from the use of the phase vocoder. Consider the scenarios illustrated in Figure 4, where we wish to slow down and speed up the audio by a factor of two ($r = 0.5$ and 2, respectively). In both cases, we wish to start the time-stretched audio at $\tau = 0$; however, the time-stretched audio *actually* begins at $\tau = 2$ when $r = 0.5$, and $\tau = -4$ when $r = 2$. Put another way, we have a latency of -2 time units when $r = 0.5$, and 4 time units when $r = 2$.

Note that this latency is calculated by extrapolating backwards in time after the block 3 has been processed at the requested rate. It could be argued that such an extrapolation cannot be correct, since it is impossible to have a negative latency, which is the case when the audio is slowed down (i.e., $R_a < R_s$). An alternative interpretation, and also one that is perhaps more mathematically correct, is that the first three output blocks are not actually produced at the requested rate: the first block is always time-stretched at rate one, and the rate gradually converges to the requested one over the next two blocks (we will revisit this in the next section on dynamic latency). However, we feel this is simply a matter of interpretation of the phase vocoder priming, and it still does not solve the problem that the audio does not start at the desired point at the requested rate.

To ensure that the time-stretched audio begins at the desired start time, we must begin pulling the input data at some offset. Based on Figure 4, we can derive a formula for this offset, $\Delta\tau_0$:

$$\Delta\tau_0 = 2\,(R_a - R_s) \tag{9}$$

As mentioned previously, the offset will be negative when R_a is less than R_s. As it is not always possible to retrieve data in the past, the data can simply be zero-padded up to that point.

6. DYNAMIC LATENCY

In addition to ensuring the time-stretched audio starts at the desired point, it is often desirable to ensure the time-stretched audio stays synchronous with a reference time-base. Using our earlier audio editor example, if the user interactively adjusts the audio play rate, we would still like to keep the play head moving across the visual waveform synchronously to the audio. Even if the audio and visual play head start synchronously, these two independent timebases may still gradually drift apart, especially if there are frequent rate changes. This is because rate changes do not occur instantaneously – they can only occur at output block intervals, and even then, as we will show below, they can take some time to take effect because the overlap-add mechanism produces a "low-pass filter" effect on rate changes.

To illustrate, let us take the example of a rate change from half speed to double speed (see Figure 5). The rate change is requested at time t_0, just after block 3 has started playing (but before processing for block 4 has begun). The requested rate change begins to take effect at time t_1, when block 4 begins to play. However, as shown in Figure 5, this output block has an effective rate, r_{eff}, of only five-eighths normal speed! This effective rate was determined by computing the input time that corresponds to the start and end of that output block using (7) – the rate, then, is ratio of the number of input samples to the number of output samples.

Using this same process, we can see that the effective rate of output block 5 is seven-eighths normal speed, and output block 6 is finally produced at the desired speed – a latency of two output blocks, or $2R_s$! If we had naïvely assumed that the rate change was applied at time t_1, then our actual audio position could be 23 ms less than what we expect![2] Recall that τ is effectively computed from a weighted average of three t_a^u values, and thus it should not be surprising that a rate change will always require two output blocks, or $2R_s$, to take effect.

$$\Delta\tau_d = 2R_s \tag{10}$$

7. DISCUSSION

As we have shown in the above sections, the processing introduced by the phase vocoder introduces a non-negligible latency both at startup and at each rate change. An analysis of these latencies requires knowledge of the underlying algorithm. It is not possible, as a client, to infer this latency by examining the behavior of a black-box time-stretcher, and in these situations, the best result than can be achieved is as described in Section 5 – clearly an unsatisfactory result.

The time-stretcher, then, must report these latency values to the client. In most situations, it is sufficient for the time-stretcher to report the input to output sample mapping and the startup latency; the dynamic latency can be inferred from the input to output sample mapping over time – however, very few time-stretchers, if any, report these properties to their clients. As demonstrated in our previous work [8, 9], such properties are necessary to precisely synchronize time-stretched audio to other media, or a reference time base.

8. FUTURE WORK

We hope to continue to develop this work by further considering the phase and group delay introduced by the

[2] Using a sample window size of $N = 4096$ samples, a constant output hop factor of $R_s = 1024$ samples, the difference would be $3(1024) - (\frac{1}{2}1024 + \frac{5}{8}(1024) + \frac{7}{8}(1024)) = 1024$ samples, or approximately 23 ms for 44.1 kHz audio.

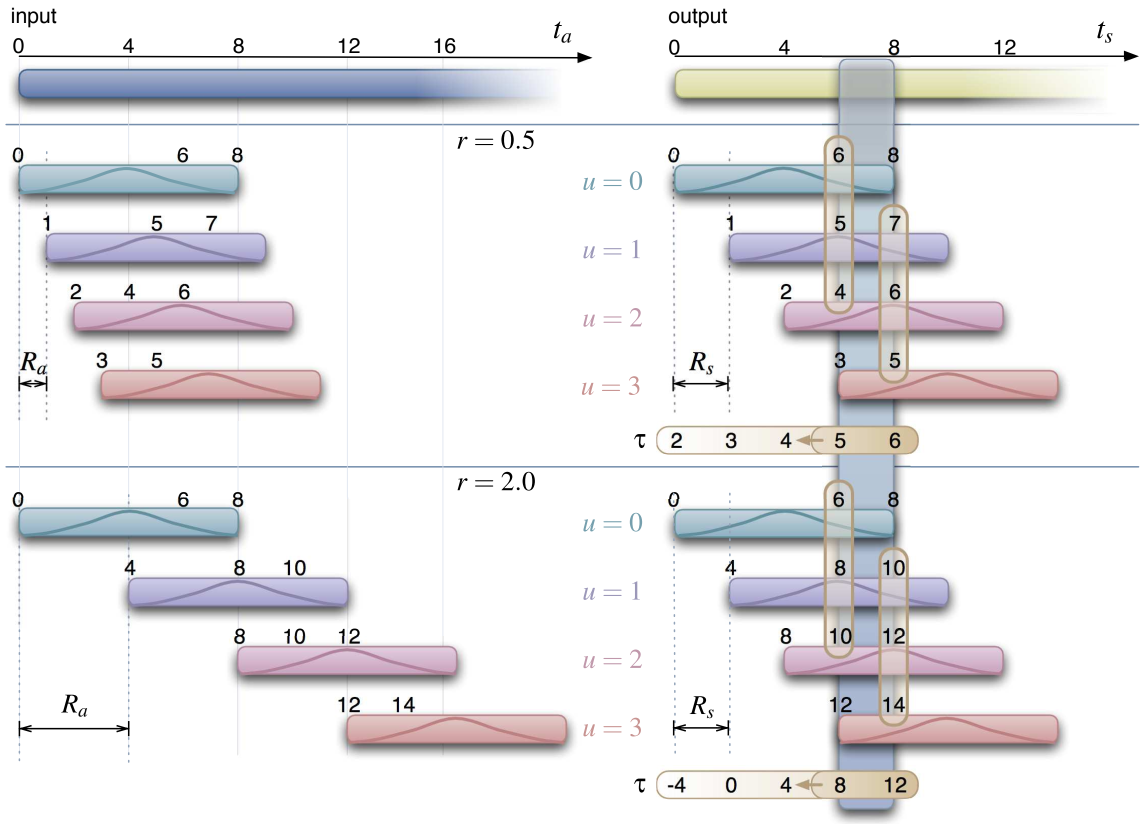

Figure 4. Illustration of startup latency. In the first case, audio is slowed down by a factor of two, and the fourth output block starts at time $\tau = 5$, computed using (7). Extrapolating backwards, the audio would start at time $\tau = 2$, which is 2 time units too late. Similarly, in the second case, audio sped up by a factor of two starts 4 time units too early.

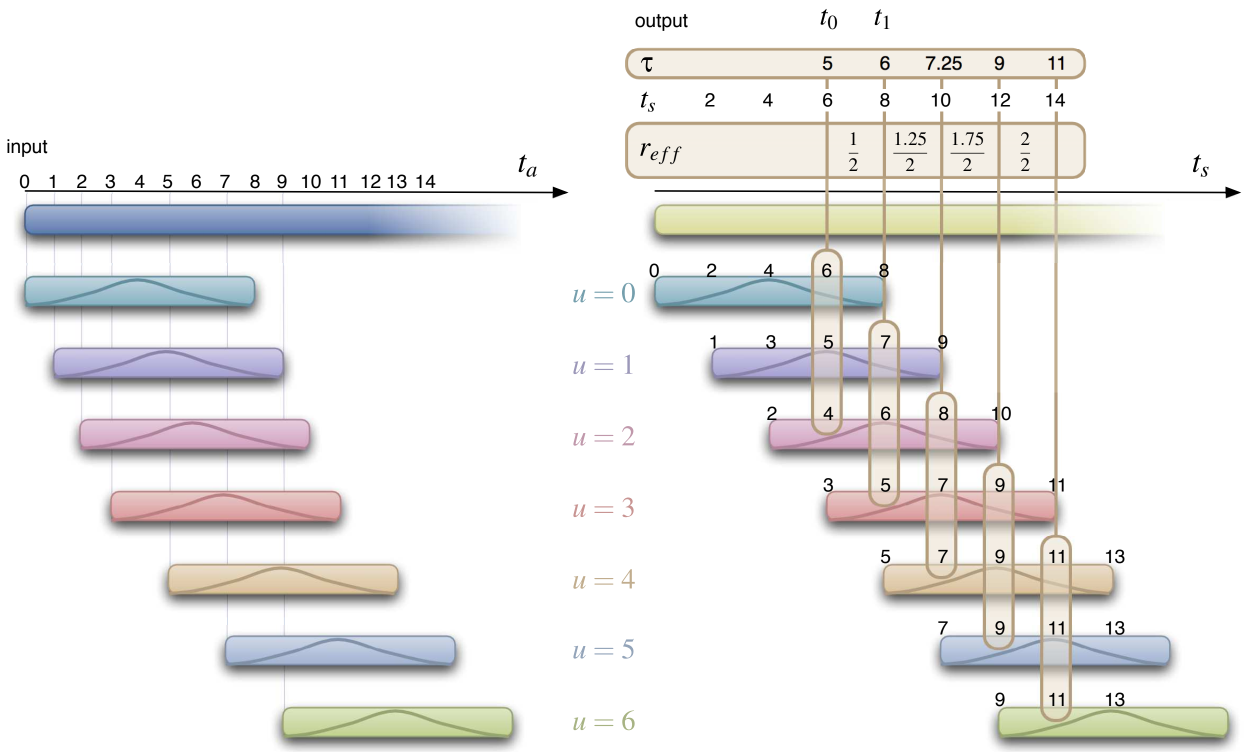

Figure 5. Illustration of dynamic latency. A rate change from half speed to normal speed is gradual, and takes two full output blocks to complete.

phase estimation calculations in the phase vocoder. Such an analysis is algorithm-dependent – the methods used to perform peak-picking, phase-locking, and transient re-alignment, for example, will all affect the results. The delay is also typically signal-dependent, in which case no closed-form solution is possible. However, there remains some possibilities for future work:

- perform an analysis for a specific algorithm, such as the basic phase vocoder
- perform an analysis for the startup latency only, eliminating most data-dependent factors; these results would still be useful for performing sample-accurate startup synchronization [4]
- where a closed-form solution is not possible, try to determine an *upper bound* on the phase/group delay introduced by the processing, perhaps using a combination of analytical and empirical methods

9. CONCLUSIONS

In this paper, we presented an analysis of startup and dynamic latency for phase vocoder-based time-stretching algorithms, which operate on the principle of respacing followed by overlap-add. We presented three approaches to interpreting the timeline of audio time-stretched using such algorithms, which consider a time-stretcher as a black-box, consider only the hop-factor, and considering the full overlap-add nature of the algorithm. We then used the result of this third interpretation to analyze startup latency and dynamic latency of the phase vocoder. We determined the startup latency to be given by $2\,(R_a - R_s)$, and the dynamic latency to be $2R_s$.

These results have already been used in applications that require precise synchronization of time-stretched audio to other media and external time sources, and we hope our work will continue to advance the state-of-the-art in interactive media and computer music systems.

10. ACKNOWLEDGEMENTS

The authors would like to thank Julius Smith for his feedback on an early draft.

11. REFERENCES

[1] Bernsee, S. M. DIRAC: C/C++ library for high quality audio time stretching and pitch shifting.

[2] DiFilippo, D., and Greenebaum, K. *Audio Anecdotes*. A K Peters, 2004, ch. Perceivable Auditory Latencies, pp. 65–92.

[3] Flanagan, J. L., and Golden, R. M. Phase vocoder. In *Bell Systems Technical Journal* (November 1966), vol. 45, pp. 1493–1509.

[4] Greenebaum, K. *Audio Anecdotes III: Tools, Tips, and Techniques for Digital Audio*. A K Peters, 2007, ch. Synchronization demystified: An introduction to synchronization terms and concepts. In Print.

[5] Greenebaum, K. *Audio Anecdotes III: Tools, Tips, and Techniques for Digital Audio*. A K Peters, 2007, ch. Sample Accurate Synchronization Using Pipelines: Put a sample in and we know when it will come out. In Print.

[6] Karrer, T., Lee, E., and Borchers, J. PhaVoRIT: A phase vocoder for real-time interactive time-stretching. In *Proceedings of the ICMC 2006 International Computer Music Conference* (New Orleans, USA, November 2006), ICMA, pp. 708–715.

[7] Laroche, J., and Dolson, M. Improved phase vocoder time-scale modification of audio. *IEEE Transactions on Speech and Audio Processing 7*, 3 (1999), 323–332.

[8] Lee, E., and Borchers, J. DiMaß: A technique for audio scrubbing and skimming using direct manipulation. In *Proceedings of AMCMM 2006 Audio and Music Computing for Multimedia Workshop* (Santa Barbara, USA, 2006).

[9] Lee, E., Karrer, T., and Borchers, J. Toward a framework for interactive systems to conduct digital audio and video streams. *Computer Music Journal 30*, 1 (2006), 21–36.

[10] Magid, R. George Lucas discusses his ongoing effort to shape the future of digital cinema. *American Cinematographer* (September 2002).

[11] Moore, G. E. Cramming more components onto integrated circuits. *Electronics 38*, 8 (April 1965).

[12] Rabiner, L. R., and Schafer, R. W. *Digital Processing of Speech Signals*. Prentice-Hall, 1978.

[13] Röbel, A. Transient detection and preservation in the phase vocoder. In *Proceedings of the ICMC 2003 International Computer Music Conference* (Singapore, 2003), ICMA, pp. 247–250.

[14] Sussman, R., and Laroche, J. Application of the phase vocoder to pitch-preserving synchronization of an audio stream to an external clock. In *Proceedings of the 1999 IEEE Workshop on Applications of Signal Processing to Audio and Acoustics* (New York, October 1999), IEEE, pp. 75–78.

[15] TV Technology. News bytes, March 2007.

[16] Verhelst, W., and Roelands, M. An overlap-add technique based on waveform similarity (WSOLA) for high quality time-scale modification of speech. In *Proceedings of the ICASSP 1993 International Conference on Acoustics, Speech, and Signal Processing* (1993), vol. II, IEEE, pp. 554–557.

[17] zplane.development. élastique time-stretching.