LIVE CONVOLUTION WITH TIME-VARIANT IMPULSE RESPONSE

Øyvind Brandtsegg *

Department of Music,
Norwegian University of Science and Technology
Trondheim, Norway
oyvind.brandtsegg@ntnu.no

Sigurd Saue

Department of Music,
Norwegian University of Science and Technology
Trondheim, Norway
sigurd.saue@ntnu.no

ABSTRACT

This paper describes a method for doing convolution of two live signals, without the need to load a time-invariant impulse response (IR) prior to the convolution process. The method is based on stepwise replacement of the IR in a continuously running convolution process. It was developed in the context of creative live electronic music performance, but can be applied to more traditional use cases for convolution as well. The process allows parametrization of the convolution parameters, by way of real-time transformations of the IR, and as such can be used to build parametric convolution effects for audio mixing and spatialization as well.

1. INTRODUCTION

Convolution has been used for filtering, reverberation, spatialization and as a creative tool for cross-synthesis ([1], [2] and [3] to name a few). Common to most of them is that one of the inputs is a time-invariant impulse response (characterizing a filter, an acoustic space or similar), allocated and preprocessed prior to the convolution operation. Although developments have been made to make the process latency free (using a combination of partitioned and direct convolution [4]), the time-invariant nature of the impulse response (IR) has inhibited a parametric modulation of the process. Modifying the IR traditionally has implied the need to stop the audio processing, load the new IR, and then re-start processing using the updated IR. This paper will briefly present a context for the work before presenting the most common convolution methods. A number of observations along the way will motivate a strategy for convolution with time-variant impulse responses based on stepwise replacement of IR partitions. Finally we present a number of use cases for the method with focus on live performance.

2. CONTEXT

The current implementation was developed in the context of our work with cross-adaptive audio processing (see [5] and also the project blog http://crossadaptive.hf.ntnu.no/) and live processing (previous project blog at [6] and the released album "Evil Stone Circle" at [7]). Our previous efforts on live streaming convolution area are described in ([8], [9]). We also note that the area of flexible convolution processing is an active field of research in other environments (for example [10], in some respects also [11] and [12] due to the parametric approach to techniques closely related to convolution).

Our primary goal has been to enable the use of convolution as a creative tool for live electronic music performance, by allowing two live sources to be convolved with each other in a streaming fashion. Our current implementation allows this kind of live streaming convolution with minimal buffering. As we shall see, this also allows for parametric transformations of the IR. Examples of useful transformations might be pitch shifting, time stretching, time reversal, filtering, all done in a time-variant manner.

* Thanks to NTNU for generously providing a sabattical term, within which substantial portions of this research have been done

3. CONVOLUTION METHODS

Convolution of two finite sequences $x(n)$ and $h(n)$ of length N is defined as:

$$y(n) = x\,(n) * h\,(n) = \sum_{k=0}^{N-1} x\,(n-k)\,h\,(k) \qquad (1)$$

This is a direct, time-domain implementation of a FIR filter structure with filter length N. A few observations can be made at this stage:

- Filter length is the only parameter involved.
- There is no latency: $y(0) = x(0)h(0)$.
- The output length N_y is equal to $2N - 1$
- Computational complexity is of the order $O(N^2)$.

Eq. 1 can be interpreted as a weighted sum of N time-shifted copies of the input sequence $x(n)$. The weights are given by the coefficients of the filter $h(n)$. In acoustic simulations, a room impulse response is a typical example of such a filter, with wall reflections represented as non-zero coefficients. The result of running a source signal through this filter is an accumulation of delayed reflections, into what we recognize as a reverberant sound.

If instead the filter $h(n)$ is live-sampled from a musical performance, the filter coefficients will typically form continuous sequences of non-negligible values. Hence, when the sound of another musical instrument is convolved with this filter, the output will exhibit a dense texture of overlays, leading to a characteristic time smearing of the signals involved.

Figure 1 illustrates what happens when a simple sinusoidal signal is convolved with itself ($x(n) = h(n)$). As expected the output is stretched out to almost double length. What is more interesting is the ramp characteristics caused by the gradual overlap of the sequences $x(n)$ and $h(n)$ in eq. 1. The summation has a single term at $n = 0$, reaches a maximum of N terms at $n = N - 1$, and returns to a single term again at $n = 2N - 2$. This inherent "fade in/fade out"-property of convolution will be exploited in our approach.

The computational complexity of direct convolution will be prohibiting when filter length increases. We normally work with filters of duration 1-2 seconds or more, which can amount to 96000

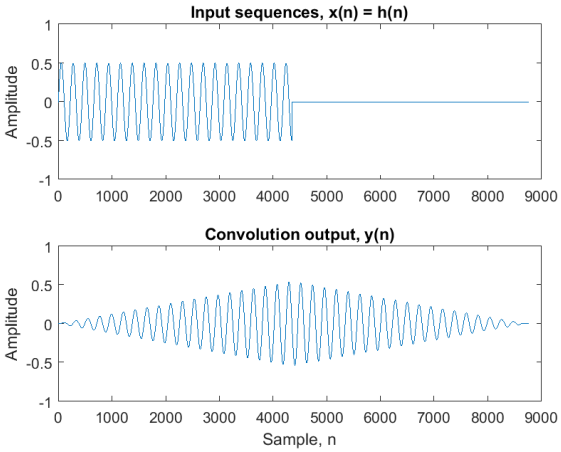

Figure 1: *Convolving a time-limited sine sequence with itself*

multiplications per output sample at 48 kHz. A far more efficient method takes advantage of the *circular convolution* property of the Discrete Fourier Transform (DFT) to perform convolution as multiplication in the frequency domain [13]:

$$y(n) = DFT^{-1}\{DFT\{x(n)\} \cdot DFT\{h(n)\}\} \tag{2}$$

where DFT and DFT^{-1} are the forward and inverse Discrete Fourier Transform, respectively.The Fast Fourier Transform (FFT) provides an efficient implementation. It is worth noting that circular convolution assumes periodicity of the input sequences. Fortunately it can be made applicable for linear filtering, provided that the FFT block size K satisfies the condition $K \geq M + N - 1$ where M, N are the lengths of the input sequences $x(n)$ and $h(n)$, respectively. If the condition does not hold there will be time aliasing [14]. The input sequences must be zero padded to length K prior to the FFT.

Equation 2 also calls for some observations:

- Computational complexity is reduced to $O(N \log N)$ (the complexity of the FFT).
- Latency has increased to the full filter length.

It is now obvious that convolution is simply multiplication in the frequency domain. From which we may further observe that:

- Output amplitude strongly depends on degree of frequency overlap between the two inputs. Hence convolution can be a challenge to work with in live performance due to lack of amplitude control.
- We may expect a relative loss of high frequency energy. For non-synthetic sounds the energy typically falls off in the high frequency range. When two such frequency spectra are multiplied the relative imbalance between high and low frequencies is magnified.

The increased latency is undesirable for real-time applications. *Partitioned convolution* reduces the latency by breaking up the inputs into smaller partitions [15]. Assume that the input sequences in eq. 1 are segmented into P partitions of uniform length $N_P = N/P$:

$$x(n) = [x_0(n), x_1(n), \ldots, x_{P-1}(n)] \tag{3}$$

$$h(n) = [h_0(n), h_1(n), \ldots, h_{P-1}(n)] \tag{4}$$

with:

$$x_i(n) = \begin{cases} x(n), & \text{if n} \in [i \cdot N_P, (i+1) \cdot N_P - 1] \\ 0, & \text{otherwise.} \end{cases} \tag{5}$$

and:

$$h_j(n) = \begin{cases} h(n), & \text{if n} \in [j \cdot N_P, (j+1) \cdot N_P - 1] \\ 0, & \text{otherwise.} \end{cases} \tag{6}$$

Then it can be proven from the linear and commutative properties of convolution that eq. 1 can be rewritten as:

$$y(n) = x(n) * h(n) = \sum_{i=0}^{P-1} \sum_{j=0}^{P-1} x_i(n) * h_j(n) \tag{7}$$

In the last part of equation 1 we notice that the indices of $x(n)$ and $h(n)$ in each term of the summation always add to n. This can be extended to the partitions $x_i(n)$ and $h_j(n)$ and allows us to deduce the interval of n where a particular partitioned convolution $x_i(n) * h_j(n)$ contributes to the output $y(n)$. If we define S_{ij} and E_{ij} as the starting and ending point of this interval, respectively, we get by adding the respective ranges in equations 5 and 6 [8]:

$$S_{ij} = (i + j) \cdot N_P \tag{8}$$

$$E_{ij} = (i + j + 2) \cdot N_P - 2 \tag{9}$$

Now let's define the interval Y_k as:

$$Y_k = [k \cdot N_P, (k+1) \cdot N_P - 1] \tag{10}$$

If we compare equations 8, 9 and 10, we see that the partitioned convolution $x_i(n) * h_j(n)$ only contributes to the output $y(n)$ for n in the intervals Y_{i+j} and Y_{i+j+1}. A trivial example of the computation with only $P = 3$ partitions is shown in Figure 2 (the sample index n is omitted for simplicity).

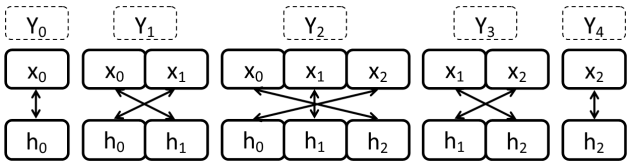

Figure 2: *Example of partitioned convolution with 3 partitions [8]*

Each partitioned convolution can be performed as multiplication in the frequency domain (equation 2). Partition size should be adjusted to make an optimal compromise between computational efficiency and latency. From this implementation we observe:

- Latency is reduced to the partition size, N_P
- The partitions $x_i(n)$ and $h_j(n)$ contributes to output interval Y_k only if $i, j \leq k$.
- The partitions $x_i(n)$ and $h_j(n)$ contributes to output interval Y_{P+k} only if $i, j \geq k$ [1].

The latter two observations play an important role for our streaming approach.

It should also be added that techniques combining partitioned and direct-form convolution can eliminate processing latency entirely [4]. Wefers [14] provide an excellent, updated review of convolution algorithms in general, with a particular focus on partitioned convolution.

[1] We will later show that the inequality can be expressed as $i, j > k$ when implementation is taken into account.

4. RESEARCH GOALS AND PREVIOUS WORK

Traditionally convolution is an asymmetric operation where the two inputs have different status: a continuous stream of input samples $x(n)$ and an impulse response (IR) $h(n)$ of finite length N. Typically the latter is a relatively short segment, a time-invariant representation of a system (e.g. a filter or a room response) onto which the input signal is applied.

Instead we are searching for more flexible tools for musical interplay and have previously presented a number of strategies for dynamic convolution [9]. Our aim has been to:

- Attain dynamic parametric control over the convolution process in order to increase playability.
- Investigate methods to avoid or control dense and smeared output
- Provide the ability to use two live audio sources as inputs to a continuous real-time convolution process.
- Provide the ability to update/change the impulse responses in real-time without glitches.

With existing tools we haven't been able to get around the time-invariant nature of the impulse response in an efficient way. In order to make dynamic updates of the IR during convolution without audible artifacts, we had to use two (or more) concurrent convolution processes and then crossfade between them whenever the IR should be modified. The IR update could be triggered explicitly, at regular intervals or based on the dynamics of the input signal (i.e. transient detection). Every update of the IR triggered a reinitialization of the convolution process.

We have also done work on live convolution of two audio signals of equal status: neither signal has a subordinate status as filter for the other [8]. Instead both are continuous audio streams segmented at intervals triggered by transient detection. Each pair of segments are convolved in a separate process. With frequent triggering the number of concurrent processes could grow substantially due to the long convolution tail ($P-1$ partitions) of each process.

5. TIME-VARIANT IMPULSE RESPONSE

In this paper we present a simple, but efficient implementation of convolution with a time-variant impulse response $h(n)$ of finite length N. In this case the coefficients of $h(n)$ are no longer constant throughout the convolution process. At a point n_T in time the current impulse response, denoted $h_A(n)$ is updated with a new filter $h_B(n)$.

Wefers and Vorländer [16] points at two possible solutions to time-varying FIR filtering: instantaneous switching or crossfading. The former can be expressed as:

$$y_n = \begin{cases} x(n) * h_A(n), & \text{if } n < n_T \\ x(n) * h_B(n), & \text{if } n \geq n_T \end{cases} \tag{11}$$

It is a cheap implementation, but the hard switching is known to cause audible artifacts from discontinuities in the output waveform.

The second option, crossfading, avoids these artifacts by filtering $x(n)$ with both impulse responses, $h_A(n)$ and $h_B(n)$, in a transition interval and then crossfade between the two outputs to smooth out discontinuities. The disadvantage of this method (assuming fast convolution in the frequency domain) is the added cost of an extra spectral convolution and inverse transform, as well as the operations consumed by the crossfade. A modified approach with crossfading in the frequency domain [16] saves the extra inverse transform.

Our goal is to be able to dynamically update the impulse response without reinitializations and crossfades of parallel convolution processes. In the following we assume a uniformly partitioned convolution scheme implemented with FFTs and multiplication in the frequency domain. The key to our approach is the inherent "fade in/fade out"-characteristic of convolution (illustrated in figure 1) and the previously noted properties of partitioned convolution:

Property 1. *The partitions $x_i(n)$ and $h_j(n)$ contributes to output interval Y_k only if $i, j \leq k$.*

An immediate consequence of Property 1 is that we can load the impulse response partition by partition in parallel with the input. A fully loaded IR is not necessary to initiate the convolution process since the later partitions initially do not contribute to the convolution output. This drastically reduces the latency e.g. for application of live sampled impulse responses. Convolution can start during sampling, as soon as a single partition of the IR is available. In addition the computational load associated with FFT calculations on the IR is conveniently spread out in time, hence avoiding bursts of CPU usage when loading long impulse responses.

Property 2. *The partitions $x_i(n)$ and $h_j(n)$ contributes to output interval Y_{P+k} only if $i, j \geq k$.*

From Property 2 it is clear that we also can *unload* the impulse response partition by partition without audible artifacts, even while the convolution process is running. The reason being that the earlier partitions no longer contribute to the output. Unloading a partition simply means clearing it to zero, and it should progress in the same order as the loading process.

The shortest possible load/unload interval is a single partition, which is actually equivalent to segmenting out a single partition of the *input signal* and convolve it with the full impulse response. Figure 3 illustrates the effect.

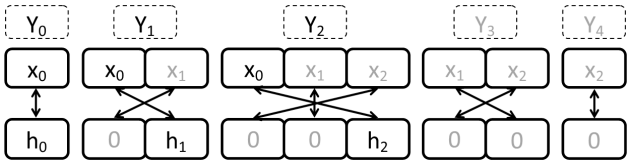

Figure 3: *Example of minimal load/unload interval*

The combination of the two strategies, stepwise loading and unloading, naturally leads to *stepwise replacement* of the impulse response. If we return to the direct form we can express the convolution with stepwise IR replacement starting at time n_T as:

$$y_n = \sum_{k=max\{0,n-n_T+1\}}^{N-1} x\left(n-k\right) h_A\left(k\right) + \sum_{k=0}^{min\{N-1,n-n_T\}} x\left(n-k\right) h_B\left(k\right) \quad (12)$$

For $n < n_T$ equation 12 reduces to:

$$y_n = \sum_{k=0}^{N-1} x\left(n-k\right) h_A\left(k\right) \quad (13)$$

For $n > n_T + N - 1$ it reduces to:

$$y_n = \sum_{k=0}^{N-1} x\left(n-k\right) h_B\left(k\right) \quad (14)$$

In the transition interval $n \in [n_T, n_T + N - 2]$ the filter coefficients of $h_A(n)$ are replaced one by one with the coefficients of $h_B(n)$, beginning with the first coefficient $h_A(0)$. This transition interval ensures a smooth transition between filters since the convolution tail of filter $h_A(n)$ is allowed to fade out properly after time n_T while the new filter $h_B(n)$ fades in. It is important to note that there is no extra cost, since it is solved by gradual coefficient replacement, and the total number of additions in equation 12 is equal to $N - 1$ at every step n. At the same time it avoids the artifacts caused by instantaneous switching (eq. 11).

The same logic holds for partitioned convolution where the filter is replaced partition by partition instead. The only restriction is that the replacement time n_T must be at a partition border:

$$n_T = k * N_P \text{for } k = 0, 1, 2, ... \quad (15)$$

At any time during the running convolution process we can trigger an update of the IR and start filling in new partitions from the beginning of the IR buffer. Figure 4 shows a simplified example of the procedure, where a filter update is triggered at the beginning of time interval Y_1. The impulse response partitions $H_{A,k}$ is replaced by $H_{B,k}$. In a transition period equal to $P - 1$ (where P is the number of IR partitions) the output is a mix of two convolution processes.

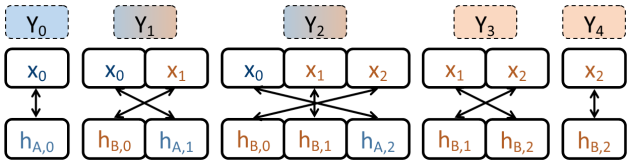

Figure 4: *Example of dynamic IR replacement starting at time interval* Y_1

It should be apparent that triggering a new IR update amounts to a segmentation of the input signal. No input partitions buffered before the trigger (x_0 in the figure) will be convolved with the new IR, and similarly, no input partitions buffered after the trigger (x_1 and x_2 in the figure) will be convolved with the old IR. To avoid discontinuities in the IR and hence audible clicks in the output signal, the impulse response buffer should always be completely refilled when updated (all N partitions).

In contrast to our earlier attempts and also to the crossfade methods suggested by [16], the proposed method does not add any extra computational cost. Impulse response partitions are simply replaced in the right order. A possible disadvantage is that the length of the transition interval is fixed to the filter length N and not available for user control.

6. THE IMPLEMENTATION

We have implemented convolution with time-variant IR as a plugin opcode titled *liveconv* written in C for the audio programming language Csound[2] . A Csound opcode normally provides three programming components: An internal data structure, an initializing function and a processing function [17]. The internal data structure is allocated in the initialization function, including all dynamic memory. In *liveconv* partition length and impulse response table are specified during this step. The content of the IR table may be updated at any time, but not its memory location or size.

The starting point for our implementation is a previous opcode, *ftconv*[3], that implements uniformly partitioned convolution with a fixed IR. The partition length must be an integer power of two: $N_P = 2^k$, and the FFT block size N_B is twice the partition length: $N_B = 2N_P$. The inputs partitions $x_i(n)$ and $h_j(n)$ are padded with N_P zeros and thereby satisfies the condition for circular convolution without time-aliasing.

To ensure correct real-time processing the implementation employs the Overlap-Add scheme (OLA): The convolution produces overlapping output segments of length N_B, which is twice the partition length N_P. Consequently, half the output samples produced by the partitioned convolution $x_i(n) * h_j(n)$ at interval Y_{i+j} must be stored and added to the output of the next interval, Y_{j+i+1}. It is worth noting that since this contribution is buffered with the output of interval Y_{i+j}, it is not necessary to recalculate the contributing convolution during interval Y_{j+i+1}. For that reason the final inequality of Property 2 can be adjusted from $\geq$ to $>$: The partitions $x_i(n)$ and $h_j(n)$ contributes to output interval Y_{P+k} only if $i, j > k$.

The Overlap-Save scheme (OLS) is found to compute more efficiently than OLA [14]. It remains to be ascertained if our partition replacement scheme is applicable to OLS without modifications. The integration with other convolution algorithms such as direct convolution in the time domain and non-uniform partitioned convolution should also be verified.

The outline of the processing function is as follows (details on OLA are omitted):

- Check the update control signal. If set to "load", prepare to reload the IR. If set to "unload", prepare to unload the IR. A data structure maintains control of each load/unload process. In theory there could be a new process for every partition.
- Read audio inputs into an internal circular buffer in chunks given by the control rate of Csound (*ksmps* is the number of samples processed during each pass of the processing function).
- When an entire partition is filled with input data:

[2]More information and links to source code at http://csound.github.io/. Documentation for the liveconv opcode can be found at http://csound.github.io/docs/manual/liveconv.html.

[3]Documentation at http://csound.github.io/docs/manual/ftconv.html.

  - Calculate the FFT of the input partition
  - For every running *load* process fetch a specified partition from the IR table and calculate its FFT. Note that several parts of the IR may be updated in parallel.
  - Do complex multiplication of input and IR buffers in the frequency domain
  - Calculate the inverse FFT of the result and write to output
  - For every running *unload* process clear a specified partition to zero.
- Increment load/unload processes to prepare for the next partition. If a process has completed its pass through the IR table, it is set inactive.

No assumptions have to be made on the impulse responses involved. The load process can be signaled as soon as one *ksmps* chunk of new data has been filled into the IR table.

In order to clarify the behavior of time-variant convolution with partition replacement, we will compare it with two overlapping time-invariant convolutions. Figure 5 shows the input signals. On top are the two impulse responses, $h_A(n)$ and $h_B(n)$. They are both 1 second excerpts of speech sampled at 44,100 Hz ($N = 44100$). At the bottom is the input source signal $x(n)$, which is a 10 second excerpt of baroque chamber music, also sampled at 44,100 Hz.

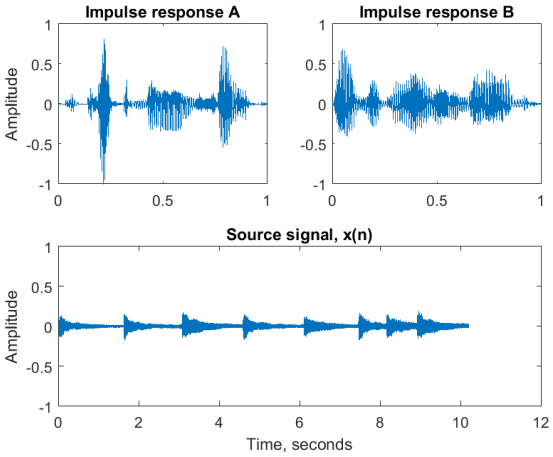

Figure 5: *Convolution demonstration. Top: The two impulse responses, $h_A(n)$ and $h_B(n)$. Bottom: The input signal $x(n)$*

In the following all convolution operations are running with partition length $N_P = 1024$ and FFT block size $N_B = 2048$. At time $n_T \approx 4.5$ seconds, a switch between impulse responses $h_A(n)$ and $h_B(n)$ is initiated (the exact time is slightly less in order to align it with the partition border).

First we show the output of two separate time-invariant convolution processes, using traditional partitioned convolution (figure 6). The input signal is split in two non-overlapping segments at the transition time n_T. The first segment defined by $n < n_T$ is convolved with impulse response $h_A(n)$ only and the output is shown immediately below (Convolution 1). The second input segment defined by $n \geq n_T$ follows next. It is convolved with

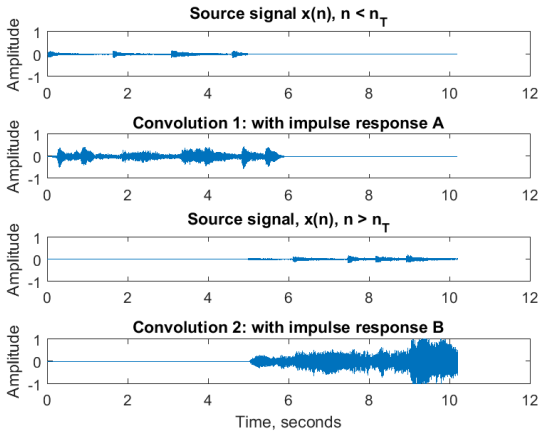

Figure 6: *Convolution demonstration: Two overlapping time-invariant convolutions. Upper pair: Source signal segment for $n < n_T$ followed by the convolution with impulse response $h_A(n)$. Lower pair: Source signal segment for $n \geq n_T$ followed by the convolution with impulse response $h_B(n)$.*

impulse response $h_B(n)$ only and the output is shown at the bottom (Convolution 2). Notice that the two outputs overlap in a time interval of length comparable to the impulse response.

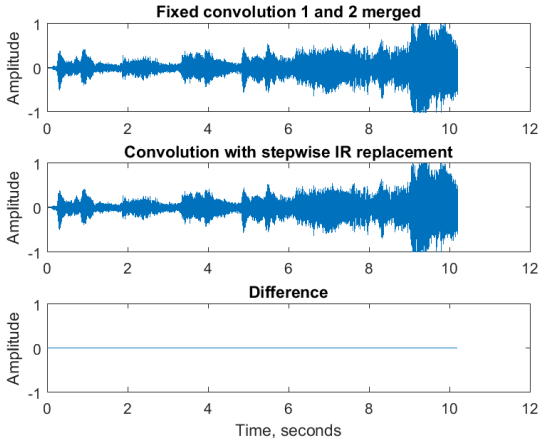

Figure 7: *Convolution demonstration: Comparing time-variant method with two overlapping time-invariant convolutions. Top: The merged result of the two time-invariant convolutions in figure 6. Middle: The output when using stepwise IR replacement. Bottom: Difference between the two.*

If we merge together the two convolution outputs in figure 6 we get the signal shown at the top of figure 7. If, on the other hand, we run the entire operation as a single time-variant convolution on the original signal $x(n)$, where the impulse response is stepwise replaced beginning at time n_T, we get the output signal shown in the middle of figure 7. The difference between the two signals are displayed at the bottom. It is uniform and for all practical purposes negligible. An estimate of the signal-to-noise ratio (SNR) returns 67.2 dB. From this demonstration it should be clear that

convolution with stepwise IR replacement preserves the behavior of time-invariant convolution, while at the same time allowing filter updates within a single process at no extra cost.

A special case appears when the IR's are sampled from the same continuous audio stream, one partition apart. Then we get a result similar to cross-synthesis, pairwise multiplication of partitions from two parallel audio streams, but still with the effect of long convolution filters. It should be added that there are more efficient ways to implement this particular condition, but that will be deferred to another paper.

The interval between IR updates is provided as a parameter available for performance control, which increases the playability of convolution. There is still a risk of huge dynamic variations due to varying degrees of frequency overlap between input signal and impulse response. The housekeeping scheme introduced to maintain the IR update processes could be exploited for smarter amplitude scaling. This is ongoing work. We would also like to look at integration of some of the extended convolution techniques proposed by Donahue & al [10].

7. USE CASE: PLUGIN FOR LIVE PERFORMANCE

A dedicated VST plugin has been implemented to show the use of the new convolution methods, built around the *liveconv* opcode. Even though the incremental IR update allows for a vast array of application areas, we have initially focused on realtime convolution of two live signals. As an indication of its primary use, we've named it "Live convolver". The plugin is implemented with a "dual mono" input signal flow, as shown in figure 8, allowing it to be used on a stereo track of a DAW. The *left* stereo signal will be used to record the IR, while the *right* stereo signal will be used as input to the convolution process.

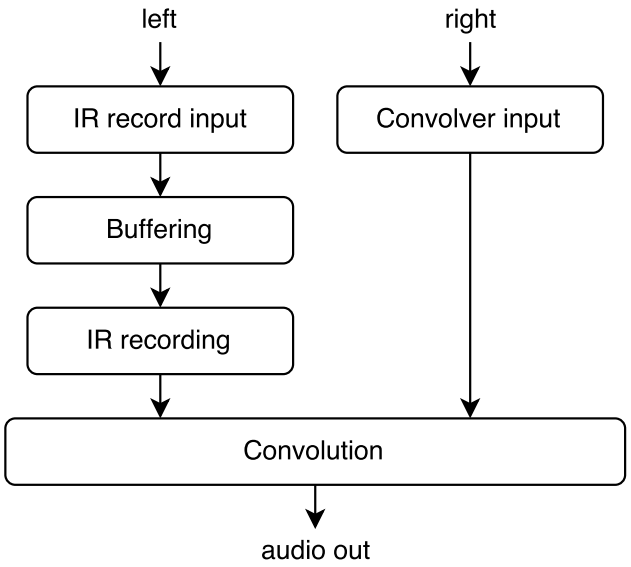

Figure 8: *Plugin signal flow overview*

The signal on the *IR record input* channel is continuously written to a circular buffer. When we want to replace the current IR, we read from this buffer and replace the IR partition by partition. The main reason for the circular buffer is to enable transformation (for example time reversal) of the audio before making the IR. As an attempt to visualize *when* the IR is taken from, we use a circular coulouring scheme to display the circular input buffer. We also represent the IR using the same colours. Time (of the input buffer) is thus represented by colour. As the color of *now* continuously and gradually changes from red to green to blue, it is possible to identify *time lag* as an azimuthal distance on the color circle [4]. Figure 9 shows the plugin gui, with a representation of the coloured input buffer and a live sampled IR. Similarly, a time reversed IR is shown in figure 10. Note the direction of color change (green to red) of the reversed IR as compared with the color change in the normal (forward) IR (blue to red).

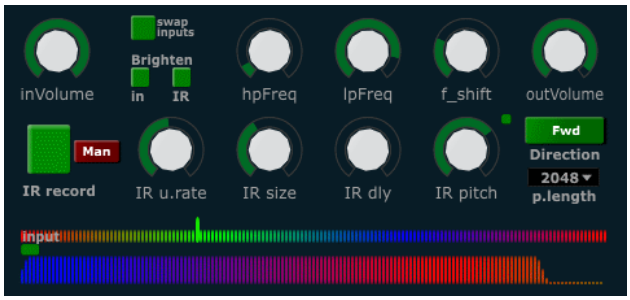

Figure 9: *Liveconvolver plugin GUI*

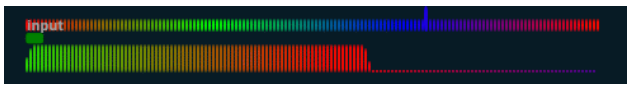

Figure 10: *Visualization of time reversed IR*

The plugin has controls for manual triggering of the IR recording, or the IR can be automatically updated by a periodic trigger (using *IR update rate* and *IR size controls*). Methods for triggering IR update via transient detection has also been implemented. Pitch modification and time reversal is available by dedicated gui controls. In addition, we have simple lowpass and highpass filtering, as this can be very useful for quickly fixing spectral problems of using convolution in a live setting. Since convolution can lead to a certain loss of high frequency content, we also have the option of "brightening" each input by applying a low-q high shelf filter. Finally, there is a significant potential for audio feedback using convolution in a live setting, when the IR is sampled in the same room where the convolver output is played back. To alleviate this risk, we have implemented a frequency shifter with relatively small amounts of shift as suggested by [18]. The amount of frequency shift is also controllable from the gui. By shifting the convolver output with just a few Hz, the feedback potential is significantly reduced.

8. USE CASE: LIVE PERFORMANCES

The liveconvolver plugin has been used for several studio sessions and performances from late 2016 onwards. Reports and sound examples from some of these experimental sessions can be found at [19] and [20]. These reports also contain reflections on the performative difference experienced in the roles of *recording the IR*

[4] https://en.wikipedia.org/wiki/Color_wheel

and *playing through it*. It is notable that a difference is perceived, since in theory the mathematical output of the convolution process is identical regardless of which one signal is used as the IR. The buffering process allows continuous updates to the IR with minimal latency, and several methods for triggering the IR update has been explored. This should ideally facilitate a seamless merging of the two input signals. However, the IR update still needs to be triggered, and the IR will be invariant between triggered updates. In this respect, the instrument providing the source for the IR will be subject to recording, and the live input to the convolution process is allowed a continuous and seamless flow. It is natural for a performer to be acutely aware of the distinction between being recorded and playing live. Similar distinctions can assumably be made in the context of all forms of live sampling performance, and in many cases of live processing as an instrumental practice. The direct temporal control of the musical energy resides primarily with the instrument *playing through the effects processing*, and to a lesser degree with the instrumental process *creating the effects process* (recording the IR in our case here).

9. OTHER USE CASES, FUTURE WORK

The flexibility gained by our implementation allows parametric control of convolution also in more traditional effects processing applications. One could easily envision a parametric convolution reverb with continuous pitch and filter modulation for example. Parametric modulation of the IR can be done either by applying transformations on the audio being recorded to the IR, or directly on the spectral data. Such changes to the IR could have been done with traditional convolution techniques too, by preparing a set of transformed IR's and crossfading between them. The possibilities inherent in the incremental IR update as we have described allows direct parametric experimentation, and thus is much more immediate. It also allows for automated time-variant modulations (using LFO's and random modulators), all without introducing artifacts due to IR updates.

10. CONCLUSIONS

We have shown a technique for incremental update of the impulse response for convolution purposes. The technique provides time-variant filtering by doing a continuous update of the IR from a live input signal. It also opens up possibilities for direct parametric control of the convolution process and as such enhancing the general flexibility of the technique. We have implemented a simple VST plugin as proof of concept of the live streaming convolution, and documented some practical musical exploration of its use. Further applications within more traditional uses of convolution has also been suggested.

11. ACKNOWLEDGMENTS

We would like to thank the Norwegian Artistic Research Programme for support of the research project "Cross-adaptive audio processing as musical intervention", within which the research presented here has been done. Thanks also to the University of California, and performers Kyle Motl and Jordan Morton for involvement in the practical experimentation sessions.

12. REFERENCES

[1] Mark Dolson, "Recent advances in musique concrète at CARL," in *Proceedings of the 1985 International Computer Music Conference, ICMC 1985, Burnaby, BC, Canada, August 19-22, 1985*, 1985.

[2] Curtis Roads, "Musical sound transformation by convolution," in *Opening a New Horizon: Proceedings of the 1993 International Computer Music Conference, ICMC 1993, Tokio, Japan, September 10-15, 1993*, 1993.

[3] Trond Engum, "Real-time control and creative convolution," in *11th International Conference on New Interfaces for Musical Expression, NIME 2011, Oslo, Norway, May 30 - June 1, 2011*, 2011, pp. 519–522.

[4] William G. Gardner, "Efficient convolution without input-output delay," *Journal of the Audio Engineering Society*, vol. 43, no. 3, pp. 127, 1995.

[5] Øyvind Brandtsegg, "A toolkit for experimentation with signal interaction," in *Proceedings of the 18th International Conference on Digital Audio Effects (DAFx-15)*, 2015, pp. 42–48.

[6] Øyvind Brandtsegg, Trond Engum, Andreas Bergsland, Tone Aase, Carl Haakon Waadeland, Bernt Isak Wærstad, and Sigurd Saue, "T-emp communication and interplay in an electronically based ensemble," https://www.researchcatalogue.net/view/48123/48124/10/10, 2013.

[7] Øyvind Brandtsegg, Trond Engum, Tone Aase, Carl Haakon Waadeland, and Bernt Isak Wærstad, "Evil stone circle," https://www.cdbaby.com/cd/temptrondheimelectroacou, 2015.

[8] Lars Eri Myhre, Antoine H Bardoz, Sigurd Saue, Øyvind Brandtsegg, and Jan Tro, "Cross convolution of live audio signals for musical applications," in *International Symposium on Computer Music Multidisciplinary Research*, 2013, pp. 878–885.

[9] Øyvind Brandtsegg and Sigurd Saue, "Experiments with dynamic convolution techniques in live performance," in *Linux Audio Conference*, 2013.

[10] Chris Donahue, Tome Erbe, and Miller Puckette, "Extended convolution techniques for cross-synthesis," in *Proceedings of the International Computer Music Conference 2016*, 2016, pp. 249–252.

[11] Jonathan S Abel, Sean Coffin, and Kyle S Spratt, "A modal architecture for artificial reverberation," *Journal of the Acoustical Society of America*, vol. 134, no. 5, pp. 4220–4220, 2013.

[12] Jonathan S Abel and Kurt James Werner, "Distortion and pitch processing using a modal reverberator architecture," in *International Conference on Digital Audio Effects (DAFx-15)*, Trondheim, Norway, November/2015 2015, .

[13] H.J. Nussbaumer, *Fast Fourier Transform and convolution algorithms*, Springer, 1982.

[14] Frank Wefers, *Partitioned convolution algorithms for real-time auralization*, Logos Verlag Berlin GmbH, 2015.

[15] Thomas G. Stockham, "High-speed convolution and correlation," in *Proceedings of the April 26-28, 1966, Spring joint computer conference*, 1966, pp. 229–233.

[16] Frank Wefers and Michael Vorländer, "Efficient time-varying fir filtering using crossfading implemented in the dft domain," in *Forum Acousticum. European Acoustics Association*, 2014.

[17] Victor Lazzarini, "Extensions to the csound language: from user-defined to plugin opcodes and beyond," in *Proceedings of the 3rd Linux Audio Conference*, 2005.

[18] Carlos Vila, "Digital frequency shifting for electroacoustic feedback suppression," in *Audio Engineering Society Convention 118*, May 2005.

[19] Øyvind Brandtsegg and Kyle Motl, "Session ucsd 14. februar 2017," http://crossadaptive.hf.ntnu.no/index.php/2017/02/15/session-ucsd-14-februar-2017/, 2017.

[20] Øyvind Brandtsegg and Jordan Morton, "Convolution experiments with jordan morton," http://crossadaptive.hf.ntnu.no/index.php/2017/03/01/convolution-experiments-with-jordan-morton/, 2017.