OPTIMIZATION OF CONVOLUTION REVERBERATION

Sadjad Siddiq

Advanced Technology Division, Square Enix Co., Ltd.
Tokyo, Japan
`siddsadj@square-enix.com`

ABSTRACT

A novel algorithm for fast convolution reverberation is proposed. The convolution is implemented as partitioned convolution in the frequency domain. We show that computational cost can be reduced when multiplying the spectra of the impulse response with the spectra of the input signal by using only a fraction of the bins of the original spectra and by discarding phase information. Reordering the bins of the spectra allows to avoid overhead incurred by randomly accessing bins in the spectrum. The proposed algorithm is considerably faster than conventional partitioned convolution and perceptual convolution, where bins with low amplitudes are discarded. Speed increases depend on the impulse response used. For an impulse response of around 3 s length at 48 kHz sampling rate execution took only about 40 % of the time necessary for conventional partitioned convolution and 61 % of the time needed for perceptual convolution. A listening test showed that there is only a very slight degradation in quality, which can probably be neglected for implementations where speed is crucial. Sound samples are provided.

1. INTRODUCTION

Artificial reverberation is an important audio effect in video games and music production. Especially in video games execution speed is critical and fast methods are needed to apply reverberation to game sounds at runtime, contributing to the immersive experience of players by making them feel they are in the environment of the game.

An overview of different algorithms to implement artificial reverberation and their history is given in [1]. The authors divide the algorithms in four categories: delay networks, convolution algorithms, computational acoustics (based on the simulation of wave propagation) and virtual analog models (based on the simulation of analog devices used to make reverb).

Delay networks are very fast and require only little memory, but finding configurations that deliver natural sounding results is not trivial. Another disadvantage is that they are abstract representations of the acoustic environment and it is challenging to implement reverberation with specific acoustic properties that does not exhibit any artifacts. Recent papers propose modulating the feedback matrix within the delay networks to reduce such artifacts [2]. There are also systems that can be configured automatically by taking the properties of the environment that is being modeled into account [3, 4, 5].

While delay networks can be used to implement fast reverberation, convolutional reverberation, in which a "dry" input sound is convolved with the impulse response (IR) of an acoustic environment, still has the big advantage that it is easy to use and understand. Additionally every environment can be recreated in the way that it is captured by the IR. The output is - depending on the quality of the IR - virtually the same as if the input sound had been recorded in the environment at the same positions where the IR was taken. For details on capturing and using IRs please refer to [6].

The big disadvantage of convolutional reverberation is its computational complexity. Instead of convolving a signal with an IR in the time domain, the two signals can be transferred to the frequency domain since multiplying the complex spectrum of the signal and the IR in the frequency domain leads to the same product as convolving both signals in the time domain. For long signals, executing the convolution in the frequency domain is much faster than in the time domain, but it is still a computationally intensive procedure.

Velvet noise can be used to create an artificial IR that allows us to lower the computational cost of convolutional reverberation [7]. Updating the velvet noise regularly makes it possible to use short buffers as IR [8, 9]. Hybrid models where the IR or some parts of it are modeled using velvet noise and low-pass filters or coloring filters allow to recreate realistic sounding output [10]. The use of cascading filters makes it possible to reduce the complexity of the single filters [11].

The aforementioned techniques reduce the complexity of the calculations involved by replacing the IR with abstract noise, but there are also techniques that allow to reduce the computational load without replacing the IR. Using only those parts of the IR that will make audible contributions to the output has a dramatic effect on the computation speed without affecting the quality of the output [12]. Additionally, on recent hardware considerable speedups can also be obtained by leveraging special instruction sets of the CPU, like single-instruction-multiple-data (SIMD) commands, parallel computing on the GPU or other low level optimizations [13].

Applying parallel computing or SIMD is less trivial in feedback delay networks, because each output sample depends on the feedback of past output samples. Also the performance of velvet noise reverberation, which is already highly optimized by reducing the amount of necessary floating point operations to a fraction of what is needed in frequency domain convolution, is limited by the overhead of random memory access. Convolution by multiplication in the frequency domain, however, is an ideal candidate for optimization based on parallel computing and SIMD commands. Most GPUs come with libraries that implement fast discrete Fourier transforms based on parallel computing out of the box, making it very easy to implement convolutional reverberation.

Keeping such new developments in mind we investigated algorithmic optimizations of convolutional reverberation that can work in conjunction with low-level hardware optimization. In this paper we propose a new method to optimize convolutional reverberation by simplifying and restructuring the IR in the frequency domain. Since the proposed method is based on convolving IRs with a dry signal, it is stable and straight-forward to use. The output only depends on the IR and some parameters controlling the trade-off between runtime speed and quality. A listening test revealed that the quality of the output is only slightly inferior to reverberation generated with conventional convolution. In applications where speed is paramount this might be an acceptable trade-off. To gain even more speed the quality level can easily be scaled down further. This easy trade-off between speed and quality is a useful feature in game audio. For less important sounds in a video game, for example, some degradation of output quality might be acceptable if execution speed can be reduced.

2. ALGORITHM OF PARTITIONED OVERLAPPING CONVOLUTION

To implement convolution in the frequency domain, we need to transform the input signal and the IR to complex frequency spectra. We do so using the fast Fourier transform (FFT). The number of FFT coefficients needs to be high enough to hold the entire convolution product. For a signal of L_x samples and an IR of L_h samples, the length of the convolution product is $L_x + L_h - 1$ samples. The number of FFT coefficients needs to be greater or equal to this number.

The complexity of the required FFT operations makes an implementation where a high order FFT is used to transfer the entire input signal to the frequency domain impracticable, even if the IR is short. Additionally, when applying reverb to continuous input whose future samples are unknown, an FFT of the entire input signal is not possible.

A solution to this problem is partitioned convolution, which we will briefly describe in this subsection as it is used in this paper. A more detailed description can be found in [14].

In partitioned convolution the signal and the IR are cut into blocks and convolution is implemented by summing up the convolution products of all blocks. In this paper block n of the IR h' will be written as $h[n]$. Each block contains M samples. Sample m of block n is defined as

$$h[n][m] = w_1[m]h'[ns+m-M/2], \text{ where } m = 0..M-1. \quad (1)$$

Here $w_1[m]$ is the periodic Hann window function of length M and s the distance between the starting samples of consecutive blocks in the original signal. We assume that $h'[t] = 0$ where $t < 0$ or $t >= L_h$. In case $s = M$, blocks do not overlap. For reasons that will be explained later, we use overlapping blocks with $s = M/2$. The bias $-M/2$ ensures that the first samples of the IR are in the center of the first block. The blocks $x[n]$ of the input signal x' are defined similarly as

$$x[n][m] = w_1[m]x'[ns + m - M/2], \quad (2)$$

again defining that $x'[t] = 0$ where $t < 0$ or $t >= L_x$. Note that in the following text $h[n]$ and $x[n]$, as well as the output $y[n]$ stand for the entire block n of M samples of the IR, input signal or output signal respectively, and not just a single sample.

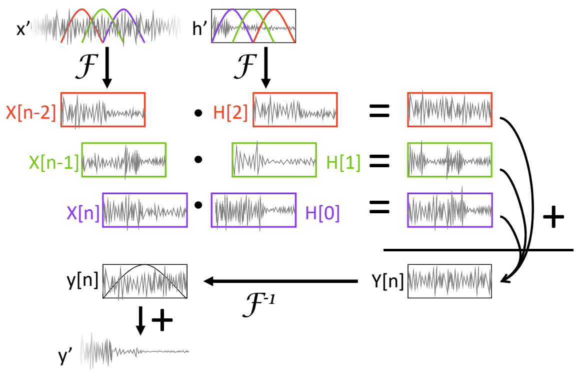

Figure 1: *Summary of partitioned overlapping convolution. In this example the IR is divided into $N_h = 3$ overlapping blocks of length M which are converted to the complex spectra $H[0..2]$ of length $2M$. These complex spectra are multiplied with the complex spectra $X[n-(0..N_h-1)]$, which are extracted from the input signal x'. The products are summed up, converted back to the time domain and overlap-added to the output signal y'.*

To calculate the output block $y[n]$, we need to sum up the convolution products of all blocks of the IR with the current input block and past input blocks according to the formula

$$y[n] = \sum_{i=0}^{N_h-1} h[i] * x[n-i], \quad (3)$$

which, as it turns out, looks like the formula for convolution applied to h and x. But please be reminded that $h[i]$ and $x[n-i]$ stand for blocks of M samples and the operator $*$ stands for the convolution of these blocks. The formula above shows which convolution products need to calculated and summed up for the output block $y[n]$.

Calculating these convolution products is more efficient in the frequency domain if M is longer than a few samples. To transfer the blocks to the frequency domain we use the FFT, making sure that there are enough coefficients to hold the convolution product, as mentioned before. Since the convolution product will be $M + M - 1$ samples long, we use an FFT with $n_{\text{FFT}} = 2M$ coefficients, which yields complex spectra that are $M_S = M + 1$ bins long. In the frequency domain equation 3 is rewritten as

$$Y[n] = \sum_{i=0}^{N_h-1} H[i] \cdot X[n-i] \quad (4)$$

where n is again the index of the output block to be calculated, $Y[n]$ is the complex spectrum of this output block and $H[i]$ and $X[n-i]$ are the complex spectra of the IR and input signal respectively. Note that we need to replace the convolution operation of the time domain with a multiplication, applied to each bin of the complex spectra.

Figure 1 summarizes the implementation of partitioned overlapping convolution. To calculate one output block $y[n]$, we need one FFT operation to calculate the complex spectrum $X[n]$ of the input block $x[n]$ and one inverse fast Fourier transform (iFFT) to transfer the complex spectrum of the output $Y[n]$ back to the time domain. The complex spectra $H[0..N_h-1]$, where N_h is their number, can be calculated offline.

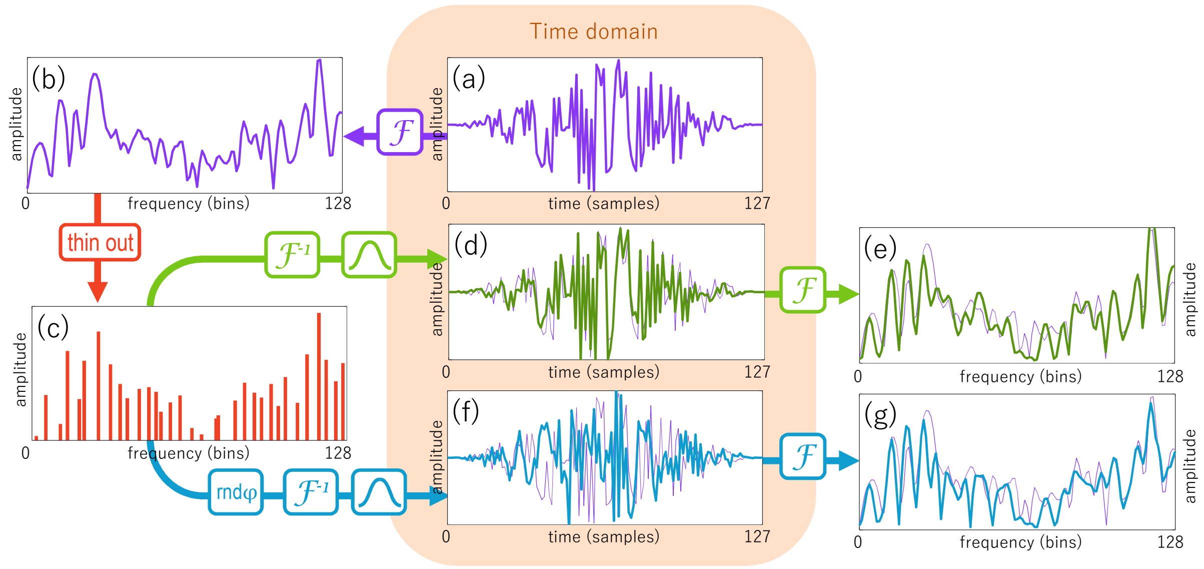

Figure 2: *A Hann window is applied to white noise in (a), it is then converted to the frequency domain to obtain the amplitude spectrum in (b). We create a sparse spectrum in (c), convert it back to the time domain, apply a Hann window and compare it to the original signal (thin, violet line) in (d). (e) shows the amplitude spectra of both signals. After applying the Hann window in the time domain the frequency content of the signal based on the sparse spectrum is very similar to the spectrum of the original signal. After randomizing the phase of the sparse spectrum in (c) as described in section 3.4, it is converted to the time domain and again compared to the original signal in (f). While the waveform looks very different, the frequency content is similar, as can be seen in (g), which shows the amplitude spectra of both signals.*

As written in equations 1 and 2, we apply a windowing function to the input blocks $x[n]$ and the IR blocks $h[n]$ before they are converted to the frequency domain. We also apply a windowing function to the output $y[n]$ before it is added to the output signal, overlapping it with previously calculated parts of the signal as follows.

$$y'[ns+m-\frac{M}{2}] \mathrel{+}= w_2[m]\cdot y[n][m], \text{ for } m = 0..2M-1, \quad (5)$$

where $w_2[m]$ is the periodic Hann window function of length $2M$, which is the length of each output block $y[n]$, since the FFT and iFFT were carried out with this many coefficients. The operator $+=$ means that the value on the right hand side is added to the value of the left hand side and the variable on the left hand side is updated to hold the new value. We use the periodic Hann window since shifted versions of that window add up to a constant if the length of the window is an integer multiple of the shift.

The most expensive part of the algorithm regarding execution speed is the multiplication of the single spectra, because each input block has to be convolved with each block of the IR. To create one single output block, we need to carry out N_h convolutions. Optimization should therefore concentrate on this part of the algorithm. In the following sections we will refer to this part of the computation as the "core loop".

3. OPTIMIZATIONS

3.1. Perceptual convolution

As reported in [12], not all bins of the complex spectra of the IR and input signal need to be used. Bins whose amplitude is below a certain threshold can be discarded without compromising on output quality, since their contribution to the output signal is barely audible. Ideally this threshold should be based on the hearing threshold, hence the name of this algorithm.

To project the hearing threshold to a certain amplitude level in the processed spectra we need to take several other values into account, for example the frequency, the playback system and the playback volume. But even if all these values are known, it is not trivial to calculate an appropriate value if masking and other perceptual factors are also taken into account. Additionally we want to discard bins before multiplying them and summing them up to an output spectrum, i.e. before we know what the output amplitude is going to be. For the experiments in this paper we set the threshold to -50 dB, where 0 dB correspond to 1, the maximum amplitude of the waveforms we process. This choice is based on informal listening tests, in which we tried to find a value where the degradation in quality was not perceivable for the IRs and test sounds we used in our experiments.

The amplitude of typical room IRs tends to drop faster in the high frequencies than in the low frequencies, so especially the last spectra of the IR will have low energy in the high frequencies regions. When multiplying the spectra of the IR with the spectra of the input signal we iterate the bins of each spectrum over frequency and stop the multiplication of bins at the last frequency in which both spectra have an amplitude that is greater than the chosen threshold. Note that the frequency of the bin at which the multiplication is stopped is independent of the preceeding and succeeding spectra. So even IR spectra with very low energy between sparse early reflections in which the multiplication stops at a low frequency do not mean that the multiplication of the succeeding spectra will also terminate at this low frequency.

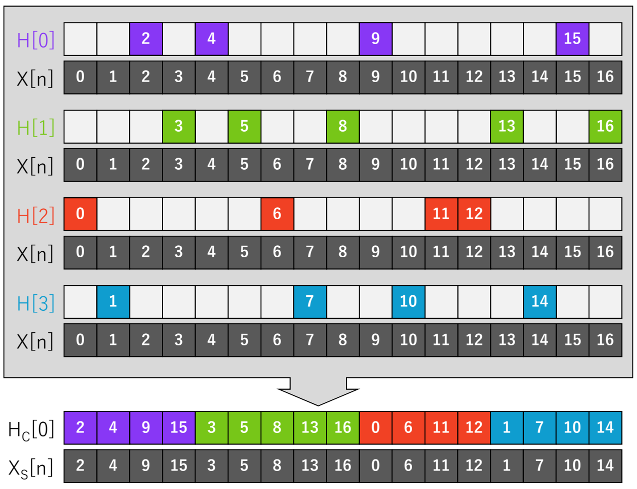

Figure 3: *Compressing sparse spectra $H[0..3]$ into $H_S[0]$. Instead of multiplying all sparse spectra with the complex input spectrum $X[n]$ it suffices to multiply the compressed spectrum $H_C[0]$ with $X_S[n]$, the reordered version of $X[n]$. Note that for illustration purpose we only show a short spectrum. The actual spectra are much longer.*

3.2. Using sparse spectra

Since the FFT is only processing a finite part of the signal, the frequency resolution in the resulting spectrum is limited and sinosoids are not represented by narrow peaks but by lobes corresponding to the Fourier transform of the window function.

Thanks to this leaking of peaks over the bins of several frequencies a lot of information in the amplitude spectrum is redundant if we do not want to recreate the exact same waveform in the time domain. For typical room IRs, which usually do not have strong, narrow peaks in the frequency spectrum and which consist mostly of decaying colored noise, we do not need to recreate the exact same shape as the input waveform as long as the overall shape of the frequency content of the signal is more or less left intact. Since we are dealing with noise, we also do not need to care about the exact pitches of the sinosoids constituting the signal.

We can therefore create an approximation of the original IR by using only a fraction of the bins of its original complex spectra and setting all other bins to zero. Figure 2 illustrates how noise can be represented using sparse spectra.

To be able to retain the approximate frequency content, the non-zero bins should be equally distributed over the whole spectrum. A constant distance between non-zero bins will lead to comb filtering artifacts in the output signal, so while their distance should be equal on average, there should also be some variation in the placement of the non-zero bins.

We define a constant d specifying that, on average, every dth bin of every spectrum of the IR should be non-zero. The upper part of Figure 3 shows how non-zero bins are chosen when $d = 4$. Every spectrum is divided into sections of d bins. Within each section only one bin is randomly chosen to be non-zero, the other bins are set to zero. As will be explained later, it is advantageous if in a group of d succeeding spectra a different bin in each section is non-zero in each spectrum, so that in one group of d spectra every bin $m = 0..M_S - 1$ is non-zero in one spectrum. To decide which bins should be non-zero in the spectra of the IR H, we start by looking at the first d bins $m = 0..d - 1$ in the first d spectra $n = 0..d - 1$. For each bin m we randomly pick one spectrum $H[0..d-1]$ in which it is non-zero. In all other spectra $H[0..d-1]$ the bin m is set to zero. After that we look at the next section of bins $m = d..2d - 1$ and continue to do so until all bins of the spectra $H[0..d - 1]$ have been processed in this way. We then repeat the algorithm for the next group of spectra $H[d..2d - 1]$ until all groups of spectra have been processed. Note that the last section of bins may contain less then d bins, as is also the case in figure 3, but the bins are still processed in the same way and set to zero in all spectra of the group expect in one randomly chosen spectrum per bin. If the last group of spectra contains less then d spectra, we process this group as if there were d spectra, assuming the non-existing spectra are all zero.

In the example shown in the upper part of Figure 3, 75 % of all bins are set to zero. When these spectra are multiplied with the input spectrum $X[n]$, the number of necessary complex multiplications is reduced by the same amount.

3.3. Reordering of the bins

Even though the amount of necessary complex multiplications when multiplying a spectrum of the input signal with a sparse spectrum of the IR has been reduced, iterating over the whole spectrum to find non-zero bins incurs a heavy overhead. Keeping a list of non-zero bins in a separate array does not solve this problem, since most time is spent when fetching the non-zero bins from their random locations in memory or storing them there.

This problem can be avoided by reordering the bins of all spectra of the IR and the input signal. Figure 3 shows how this can be done for four spectra of the IR $H[0..d]$, when $d = 4$. The bins of the spectrum are reordered in a way that the four non-zero bins of $H[0]$ come first, then the non-zero bins of spectrum $H[1]$ and so on. This way the four spectra can be summarized in one array of length M_S. As can be seen in the figure, the first four elements of the array are the bins 2, 4, 9 and 15 and represent the spectrum $H[0]$. We will refer to these elements as the first split, to the next five elements representing $H[1]$ as the second split and so on.

After reordering the input spectrum $X[n]$ in the same way to $X_S[n]$, we can multiply all bins of $X[n]$ with $H_S[0]$ in this shuffled order and then place the splits into the corresponding output spectra $Y_S[n..n + d - 1]$ in the same shuffled order.

The sparse spectra of the IR $H[0..d-1]$ have been compressed into $H_C[0]$. Similarly the succeeding sparse spectra $H[d..2d - 1]$ are compressed into $H_C[1]$. We use the exact same pattern so that the bins are ordered in the same way as in $H_C[0]$. We can therefore bundle all spectra of the IR into groups of d spectra and compress each group to one spectrum following the exact same order of bins, yielding $H_C[k]$, the compressed reordered spectra of the IR, where $H_C[k]$ contains the spectra $H[kd..kd+d-1]$. The number of H_C is $N_k = \text{ceil}(N_H/d)$

The spectra of the IR can be reordered in the initialization phase of the convolution algorithm. The spectra of the input signal need to be reordered during the execution of the algorithm, but since all spectra follow the same order, we need to reorder every spectrum $X[n]$ only once and can thus avoid expensive reordering in the core loop.

The optimization based on discarding bins with small amplitudes as described in 3.1 works best when the bins are still ordered from smallest to largest frequency within each split. This allows

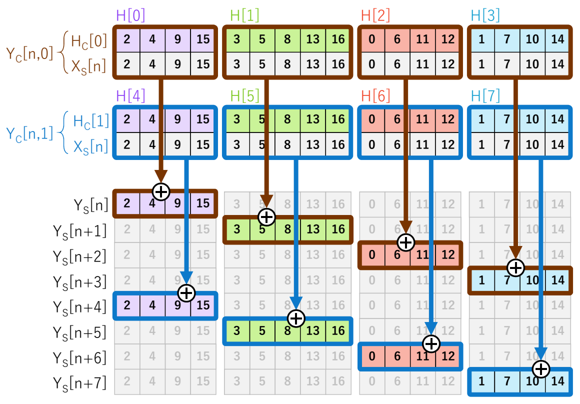

Figure 4: *After multiplying the input spectrum $X_S[n]$ with the spectra of the IR compressed in $H_C[0]$ and $H_C[1]$, the splits containing the products are written to their corresponding output spectra.*

us to stop the calculation within the split at the frequency where the amplitude drops below the threshold.

Figure 4 shows how the output spectra $Y_S[n]$ are generated from the products of the current input spectrum $X_S[n]$ with the compressed spectra of the IR $H_C[k]$. As was shown by formula 3, we need to multiply the current input spectrum and past input spectra $X[n..n - N_H + 1]$ with all spectra of the IR $H[0..N_H - 1]$ and sum up the products to calculate the current output spectrum $Y_S[n]$. Alternatively, we can also multiply the current input spectrum $X[n]$ with all spectra of the IR $H[0..N_H - 1]$ and add the products to an output cache that consists of the current output spectrum $Y_S[n]$ and future output spectra $Y_S[n+1..N_H-1]$. This is of course still true when the spectra of the IR have been compressed into $H_C[k]$, but the compressed spectra need to be decompressed and the single splits written to their corresponding output spectra. Let us first multiply the input spectra $X[n]$ by $H_C[k]$ and store the temporary product in $Y_C[n, k]$:

$$Y_C[n, k] = H_C[k] \cdot X[n] \tag{6}$$

Now the splits in $Y_C[n, k]$ need to distributed to the corresponding output spectra. The first split of $Y_C[n, k]$ is added to $Y_S[n]$, the second split to $Y_S[n + 1]$ and so on for all splits. The output spectrum $Y_S[n]$ is thus the sum of the first split $s = 0$ of $H_C[0] \cdot X_S[n]$, the second split $s = 1$ of $H_C[0] \cdot X_S[n - 1]$ and so on. The following formula summarizes this relation:

$$Y_S[n+s][a(s)..b(s)] \mathrel{+}= Y_C[n, k][a(s)..b(s)], \text{for } s = 0..d \tag{7}$$

Here $a(s)$ and $b(s)$ are, respectively, the first and last index of split s and again the operator $+=$ means that the value on the right hand side is added to the value on the left hand side, updating the variable on the left hand side.

We can restore the original order of bins in the output spectra just before converting the signal back to the time domain using the iFFT, after all necessary spectra have been added to the sum.

3.4. Discarding phase

To implement the multiplication of the complex spectra, six floating point operations are needed (four multiplications and two additions) for each pair of complex numbers, i.e. for each pair of bins. Since these floating point operations are carried out in the core loop, reducing their number has a big impact on performance.

Sparse spectra, while retaining enough information about the frequency content of the signal, have the disadvantage that they do not allow us to generate signals that resemble the shape of the original waveform exactly. Output blocks need to be overlapped after applying a windowing function in order to generate a signal without artifacts caused by discontinuities between blocks in the time domain.

Since we are forced to overlap the output buffers anyway, we can introduce another simplification and dispose of the phase in the complex compressed spectra of the IR H_C. Setting all phases to 0 would introduce undesired periodicity in the output signal, so we set the phase to random values. The lower part of figure 2 shows the effect of randomizing the phase in the same way as described below. While the shape of the waveform in the time domain bares no resemblance to the original anymore, the frequency content is still close to the original.

We set the phases of the non-zero bins to values like

$$\phi = e^{\frac{1}{2}\pi r j}, \tag{8}$$

where r is a random integer number in the range $0 \leq r \leq 3$. If we randomize the phases this way, either Re ϕ or Im ϕ is zero and the implementation of the complex multiplication of one bin in H_C and X_S can be realized with only two multiplications and one addition.

Verifying if the real part or the imaginary part of the complex number is zero to determine how the multiplication should be carried out takes time in the core loop. Instead, since the phase is random anyway, it is much faster to hard code the order of phases.

If we assume that the order of the hard coded phases is

$$\begin{aligned}
\phi_1 &= e^{\frac{1}{2}\pi j} = 1, \\
\phi_2 &= e^{\pi j} = j, \\
\phi_3 &= e^{\frac{3}{2}\pi j} = -1 \text{ and} \\
\phi_4 &= e^{2\pi j} = -j,
\end{aligned}$$

then the first bins of the multiplication of $X_S[n]$ and $H_C[0]$ will be carried out in this way:

$$\begin{aligned}
Y_S[n][0] &= \mathrm{Re}X_S[n][0] \cdot H_C[0][0] + j\mathrm{Im}X_C[n][0] \cdot H_C[0][0] \\
Y_S[n][1] &= j\mathrm{Re}X_S[n][1] \cdot H_C[0][1] - \mathrm{Im}X_C[n][1] \cdot H_C[0][1] \\
Y_S[n][2] &= -\mathrm{Re}X_S[n][2] \cdot H_C[0][2] - j\mathrm{Im}X_C[n][2] \cdot H_C[0][2] \\
Y_S[n][3] &= -j\mathrm{Re}X_S[n][3] \cdot H_C[0][3] + \mathrm{Im}X_C[n][3] \cdot H_C[0][3]
\end{aligned}$$

The same pattern is repeated for the following bins, until all bins have been processed. Informal listening tests have shown that this kind of hard coded randomization to four different phase values is enough to avoid periodic artifacts in the output.

Discarding the phase information also has the advantage that the only value that needs to be stored for every bin of the IR spectra is the amplitude, which lets us save memory.

4. IMPLEMENTATION DETAILS

4.1. Avoiding temporal smearing

The disadvantage of destroying the phase information as described in section 3.4 is that sharp impulses in the IR will also be destroyed. This can be heard especially in the beginning of the output signal where the direct sound and the early reflections are standing out. In typical IRs we can get rid of the phase in most parts and implement convolution using the optimizations described above. For those few parts of the IR where we want to keep the phase information intact, for example the first block or the first few blocks, we can use conventional convolution.

4.2. Consistent IR amplitude

Convolving the input signal with sparse spectra in which most bins are set to zero has an effect on the amplitude of the output. Since every sparse spectrum in the IR has nearly the same number of non-zero bins, the effect on the amplitude of the output is approximately the same in all output blocks and results in attenuating or amplifying the whole signal by a constant. This is important to note if we use the original spectra for some blocks of the IR and sparse spectra for the rest. In this case we need to ensure that the amplitude level of those parts of the input signal that are convolved with the sparse spectra does not change when compared to convolving them with the original spectra.

The change in amplitude when setting random bins to zero is difficult to predict. We therefore measure the output amplitude of our reverberator by using a unit impulse as an input and calculating its IR. We then compare the root mean square (RMS) of those parts of the output that where generated using sparse spectra to the RMS of the corresponding parts of the original IR to find out by which value α we need to multiply the output of our reverberator to generate output with the same amplitude as the original IR. The amplitudes in the sparse spectra of the IR are then all multiplied by α.

4.3. Split order

Figure 3 showed how d sparse spectra of the IR are compressed into one spectrum. While the possible combinations of bins that are non-zero in one sparse spectrum always remain the same, for example [2, 4, 9, 15] or [3, 5, 8, 13, 16], the order in which we pick one such combination for one spectrum of the IR can be random. In the figure spectrum $H[0]$ is represented by the combination of bins [2, 4, 9, 15], which corresponds to the first split in the compressed spectrum $H_C[0]$. The second spectrum $H[1]$ is represented by the second split, and so on. We could, however, also represent $H[0]$ by the second split, $H[1]$ by the first split, and $H[2]$ and $H[3]$ randomly by the two other splits. Randomly deciding which one of the d splits to use to represent spectrum $H[n]$ avoids artifacts based on repetition of the same pattern in the output. Note, however, that there is one restriction: Within a group of d succeding spectra, each must be represented by a different split, so that all d spectra can be squeezed into one spectrum H_C without having bins that are non-zero in more than one sparse spectrum within the group. Informal listening tests have shown that it also improves quality if two succeeding sparse spectra of the IR are never represented by the same split. When assigning all spectra within a group to a different split each, care should be taken that the first split is not the same as the last split of the previous group. After multiplying the compressed spectrum H_C with an input spectrum, we only need to take care that we know which split of the compressed output spectrum Y_C needs to go to which output spectrum Y_S.

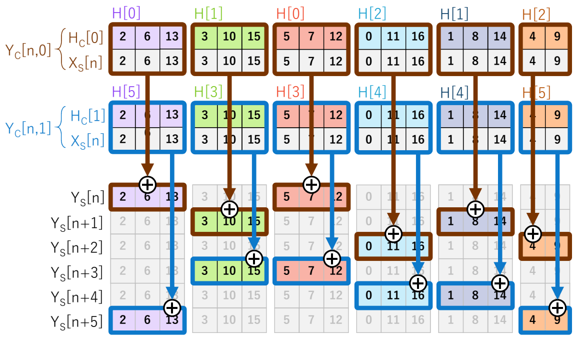

Figure 5: *After multiplying the input spectra $X_S[n-4..n]$ with the spectra of the IR compressed in $H_C[0]$, the products of the splits are written to their corresponding output spectra. This process is shown for two succeeding output spectra $Y_S[n-1]$ and $Y_S[n]$ when $d = 6$ and $c = 2$.*

4.4. Multiple splits

To increase randomization further we can increase the number of splits. Unfortunately this also reduces the number of non-zero bins in the sparse spectra, which results in deteriorated output quality. To avoid this we can represent one spectrum of the IR by a random combination of c splits, as shown in figure 5. This also has the advantage that the number of patterns of the distribution of non-zero bins is increased further. When we represent each spectrum by a combination of c splits and we have d splits in total, the number of possible unique combinations can be calculated as the binomial coefficient

$$\binom{d}{c} = \frac{d!}{c!(d-c)!}. \tag{9}$$

When writing the output signal we need to write all c splits that represented one spectrum of the IR to the corresponding output spectrum, as is shown in the figure. We can do so by updating equation 7 as follows:

$$Y_S[n+s_k[k,s]][a(s)..b(s)] += Y_C[n,k][a(s)..b(s)], \text{ for } s=0..d \tag{10}$$

So, instead of writing the splits in a constant order, we check the table $s_k[k, s]$ to find out to which output spectrum split s of the kth compressed spectrum needs to be written to.

4.5. Multichannel input and output

Stereo or multichannel input and output can essentially be processed in the same way as mono signals, applying the same calculations to each channel. The content of the different channels in typical multichannel signals does not only differ in amplitude, but also in phase. Since the phase information is discarded in the proposed algorithm, the same difference can not be reconstructed in the output signal. Informal listening tests have shown that it suffices to ensure there is some difference between the output channels - even if that difference is not the same as in the original signal. This can be done by making sure that the order of splits used for each channel is different. In the case of stereo input signals or stereo IRs this will lead to output that sounds slightly different from sounds generated with conventional convolution but the listening tests detailed in section 5.4 have shown that listeners had no strong preference for one or the other.

5. PERFORMANCE

5.1. Overview

We compared the speed and memory requirements of the proposed algorithm to two other algorithms: partitioned convolution without any optimizations (also called "non-optimized" below) and perceptual convolution where bins of the complex spectra of the IR are discarded if they are below a certain threshold. A listening test to compare the quality of reverberation generated by the proposed algorithm and by the non-optimized algorithm was also conducted.

The samples for the listening test were all generated with a block size of $M = 1024$ samples. As mentioned in 3.1, a cutoff amplitude of -50 dB was used for the perceptual and the proposed algorithm. To create the output samples of the proposed algorithm each spectrum was represented by a combination of $c = 3$ splits and each compressed spectrum of the IR contained $d = 5$ splits. We used non-optimized convolution for the first block of the IR and optimized convolution for all other blocks. The sampling rate depended on the IR and input sample used and was either 44.1 kHz or 48 kHz. Some sounds were rendered as mono and some as stereo. All sounds were processed as 32 bit floating point values.[1]

For the performance analysis a mono channel sample with a sampling rate of 48 kHz and a length of 70.37 s was chosen and an IR of 3.2 s length with the same sampling rate was used. The parameters for the algorithm were the same as mentioned above.

5.2. Computation speed

Table 1 compares the floating point operations (FLOPs) necessary to calculate the output of each algorithm. The values are normalized by the length of the input signal. FLOPs only give a very rough idea of the performance of the algorithm, since during execution a lot of time is also spent in memory access, but it is interesting to compare the three algorithms this way to see where most computational power is needed. In the non-optimized and in the perceptual algorithm the cost for the FFT is relatively low, but the cost for multiplying the spectra is high. In the proposed algorithm we need to process overlapping blocks, thus the number of FLOPs needed to calculate the FFT doubles. At the same time the number of FLOPs needed to multiply the spectra decreases. If the FFT is implemented in hardware and does not have to run on the CPU, the efficiency of the proposed algorithm will increase even more.

The speed of the algorithm was measured on a PC running on an Intel® Core™ i7-7700K CPU at 4.2 GHz. The time needed to convolve the entire input signal with the IR was about 718 ms for the non-optimized algorithm, 481 ms for the perceptual algorithm and 294 ms for the proposed algorithm at a sampling rate of 48 kHz. SIMD optimizations or GPU optimizations were not used. The algorithm was implemented in C++, executed as a single thread and only the computation time of the reverberation algorithm was measured. Data input and output etc. are therefore not part of this measurement. Note that the measurements for our algorithm include the computation time of conventional convolution for the first block. Please also note that while the length of the input signal has no impact on the performance by which the algorithms processes one block of input data, it is, however, affected by the length of the IR, since every block of the input signal needs to be processed with every block of the IR. The values given in the table are thus specific to the IR used in performance measurements.

[1]Examples can be found here: https://doi.org/10.5281/zenodo.3965342

Table 1: *Comparison of the three algorithms. Floating point operations are counted per input sample and rounded to the nearest integer. They are divided into FFT and multiplication of spectra ("mult."). The sum of the two is also shown. The computation time is given in ms per second of the input signal. Values are measured for an IR of 3.2 s at 48 kHz sampling rate and will vary depending on the IR.*

	non-optimized	perceptual	ours
FLOPs / sample (FFT)	87	87	174
FLOPs / sample (mult.)	1265	863	189
FLOPs / sample (total)	1352	951	364
computation time / s	10.2 ms	6.84 ms	4.18 ms
Memory	2476 kB	2100 kB	2573 kB

5.3. Memory requirements

All three algorithms need to store the frequency domain representation of the IR, i.e. the spectra of the IR in memory. The second big requirement is the memory needed for cache buffers to hold future output blocks. For all three algorithms the memory needed is not affected by the length of the input signal, but depends on the length of the IR.

In the non-optimized algorithm the IR is cut into non-overlapping frames of length M. Those frames are transferred to the frequency domain using an FFT with $2M$ coefficients and stored as complex spectra of length $M_S = M + 1$, adding 1 to M to keep a bin for the Nyquist frequency. Two floating point values are needed to store the real and imaginary part of each bin. So one block of M floating point samples in the time domain is represented by $2M_S = 2M + 2$ floating point values in the frequency domain. Neglecting the additional bin for the Nyquist frequency and the fact that the last samples of the IR need to be padded with zeros if its length is not a multiple of M, the memory needed for the IR is approximately $2L_h$ floating point values. The memory needed for the output cache is the same. The total amount of memory needed is therefore about $4L_h$ floating point values.

We can calculate the memory needed for the perceptual algorithm in the same way. However, some memory can be saved by discarding the last bins of each spectrum of the IR if their amplitude drops below our chosen threshold. In the case of the IR and the threshold used in the tests above about 30 % of all bins could be discarded. Please note that the number of bins that can be discarded depends on the IR and on the threshold that is being used. Informal tests with other IRs have shown that up to 50 % of all bins can be discarded without compromising on quality. The memory needed to store the IR is therefore somewhere between L_h and $2L_h$ floating point values. The memory needed for the output cache is $2L_h$, so the total amount of memory needed lies somewhere between $3L_h$ and $4L_h$.

For the proposed algorithm we need overlapping blocks where the distance s between the starting point of two successive blocks is $M/2$. The number of blocks we extract is thus twice as big as for the previous algorithms. As for the spectra of the IR this doubling is compensated by the fact that we discard phase information and only need to store the amplitude of each bin, i.e. only one value for each bin. We summarize d sparse spectra into one

Table 2: *Question and answers of the survey. The score of each answer was not shown to participants.*

Score	Question / Answers
	In which sound is the quality of the reverberation better?
3	It is much better in A.
2	It is better in A.
1	It is slightly better in A.
0	It is about the same in both sounds.
-1	It is slightly better in B.
-2	It is better in B.
-3	It is much better in B.
0	I don't know.

spectrum, which reduces the amount of memory we need by a factor of d. In the case of a combination of c sparse spectra, the factor is d/c. Additionally, we can discard those bins in each split of the compressed spectra that do not contribute in an audible way to the output signal, in the same way as we did it in the perceptual algorithm. The memory needed to store an IR where up to 50 % of all bins can be discarded based on their low amplitude is thus between $\frac{c \cdot L_h}{d}$ and $\frac{c \cdot L_h}{2d}$ floating point samples. The memory needed for the output cache is twice as high as for the other algorithms, because aoutput buffers need to overlap as well. The total amount of memory needed is therefore about $4L_h + \frac{c \cdot L_h}{d}$ and $4L_h + \frac{c \cdot L_h}{2d}$ floating point values.

Table 1 shows the memory values measured for the three algorithms when the IR and input sound mentioned in 5.1 are used.

5.4. Listening test

To evaluate the quality of the reverberation generated by the proposed algorithm a simple listening test based on the comparison category rating method [15] was conducted. In the test the 15 different short sound samples to which the link was given in 5.1 where used. The samples included speech in different languages, small musical phrases played on the piano or harpsichord, paper being crushed, a door being slammed and a wine glass being hit. To each sample reverberation based on a different IR was applied and in the test each sample was presented in two versions. One version was the sample with reverberation applied by our algorithm, the other one had reverberation generated by conventional partitioned convolution. These two versions were randomly presented as sound A or sound B. Subjects were asked to use earphones or headphones and were allowed to listen to the sounds as often as they like and to revisit earlier questions if they wanted. For each pair of sounds they had to decide if the quality of the reverberation in sound A or sound B was better and by how much. The answers subjects could pick from are shown in table 2. To evaluate the results of the survey, each answer was associated with the score shown in the table. For pairs where sound B was the sound generated by our algorithm, the signs of these scores were flipped.

The survey was done by 25 subjects, who had, for the most part, no particular experience in working with audio. The average over all questions and subjects was a score of -0.25 with a standard deviation of 1.48. This shows a small tendency of preference towards the sound generated by conventional convolution but the large standard deviation shows that this tendency is not very consistent. For the two percussive sounds there was a small tendency to prefer the versions generated by our algorithm (score 0.2 and 0.32), presumably because of the different stereo spread.

6. REFERENCES

[1] Vesa Välimäki, Julian Parker, Lauri Savioja, Julius O Smith, and Jonathan Abel, "More than 50 years of artificial reverberation," in *Audio engineering society conference: 60th international conference: dreams (dereverberation and reverberation of audio, music, and speech)*. Audio Engineering Society, 2016.

[2] Sebastian Jiro Schlecht, *Feedback Delay Networks in Artificial Reverberation and Reverberation Enhancement*, Ph.D. thesis, Friedrich-Alexander-Universität Erlangen-Nürnberg, 2018.

[3] Carl Schissler and Dinesh Manocha, "Interactive sound rendering on mobile devices using ray-parameterized reverberation filters," *arXiv preprint arXiv:1803.00430*, 2018.

[4] Hequn Bai, *Hybrid models for acoustic reverberation*, Ph.D. thesis, Télécom ParisTech, 2016.

[5] Enzo De Sena, Hüseyin Hacıhabiboğlu, Zoran Cvetković, and Julius O Smith, "Efficient synthesis of room acoustics via scattering delay networks," *IEEE/ACM Transactions on Audio, Speech, and Language Processing*, vol. 23, no. 9, pp. 1478–1492, 2015.

[6] Angelo Farina, "Impulse response measurements," in *23rd Nordic Sound Symposium, Bolkesjø (Norway)*, 2007, pp. 27–30.

[7] Hanna Järveläinen and Matti Karjalainen, "Reverberation modeling using velvet noise," in *Audio Engineering Society Conference: 30th International Conference: Intelligent Audio Environments*. Audio Engineering Society, 2007.

[8] Keun-Sup Lee, Jonathan S. Abel, Vesa Välimäki, Timothy Stilson, and David P. Berners, "The switched convolution reverberator," *Journal of the Audio Engineering Society*, vol. 60, no. 4, pp. 227–236, 2012.

[9] Sami Oksanen, Julian Parker, Archontis Politis, and Vesa Välimäki, "A directional diffuse reverberation model for excavated tunnels in rock," in *2013 IEEE International Conference on Acoustics, Speech and Signal Processing*. IEEE, 2013, pp. 644–648.

[10] Bo Holm-Rasmussen, Heidi-Maria Lehtonen, and Vesa Välimäki, "A new reverberator based on variable sparsity convolution," in *Proc. of the 16th Int. Conference on Digital Audio Effects (DAFx-13)*, 2013, vol. 5, pp. 7–8.

[11] Vesa Välimäki, Bo Holm-Rasmussen, Benoit Alary, and Heidi-Maria Lehtonen, "Late reverberation synthesis using filtered velvet noise," *Applied Sciences*, vol. 7, no. 5, pp. 483, 2017.

[12] Wen-Chieh Lee, Chi-Min Liu, Chung-Han Yang, Jiun-In Guo, et al., "Fast perceptual convolution for room reverberation," in *6th International Conference on Digital Audio Effects (DAFx-03), London, United Kingdom*. Citeseer, 2003.

[13] Stefan Heidtmann, "Implementation and evaluation of optimization strategies for audio signal convolution," M.A. Thesis, Hochschule für Angewandte Wissenschaften Hamburg, 2019.

[14] Frank Wefers, *Partitioned convolution algorithms for real-time auralization*, vol. 20, Logos Verlag Berlin GmbH, 2015.

[15] "Methods for subjective determination of transmission quality," *ITU-T P.800 (08/1996)*.