Design of a Convolution Engine optimised for Reverb

Fons ADRIAENSEN
fons.adriaensen@skynet.be

Abstract

Real-time convolution has become a practical tool for general audio processing and music production. This is reflected by the availability to the Linux audio user of several high quality convolution engines. But none of these programs is really designed to be used easily as a reverberation processor. This paper introduces a Linux application using fast convolution that was designed and optimised for this task. Some of the most relevant design and implementation issues are discussed.

Keywords

Convolution, reverb.

1 Introduction

The processing power of today's personal computers enables the use convolution with relatively long signals (up to several seconds), as a practical audio tool. One of its applications is to generate reverberation — either to recreate the 'acoustics' of a real space by using captured impulse responses, or as an effect by convolving a signal with synthesised waveforms or virtual impulse responses.

Several 'convolution engines' are available to the Linux audio user. The *BruteFir* package[1] by Anders Torger has been well known for some years. More recent offerings are Florian Schmidt's *jack_convolve*[2] and *JACE*[3] by Fons Adriaensen.

While one can obtain excellent results with these programs, none of them is really designed to be an easy-to-use reverb application. Another problem is that all of them use partitioned convolution with uniform partition size, which means there is a tradeoff to be made between processing delay and CPU load (JACE allows the use of a period size smaller than the partition size, but this does not decrease the latency). While in e.g. a pure mixdown session a delay of say 100 ms could be acceptable, anything involving live interaction with performers requires much smaller latency.

The *Aella* package written by the author is a first attempt to create a practical convolution based 'reverb engine'. An alpha release[4] will be available at the time this paper is presented at the 4th Linux Audio Conference. In the following sections, some of the design and implementation issues of this software will be discussed.

2 Anatomy of natural reverb

If reverb is added as an effect then everything that 'sounds right' can be used. If on the other hand the object is to recreate a real acoustical space or to create a virtual one, then we need to observe how a natural reverb is built up, and how it is perceived by our hearing mechanisms.

Imagine you are in a concert hall listening to some instrument being played on stage. Provided the sound source is not hidden, the first thing you hear is the direct sound (DS). This will be followed after some milliseconds by the first reflections from the walls and ceiling. The sound will continue to bounce around the room and a complex pattern of reflections will build up, with increasing density and decreasing amplitude. This is shown in fig.1. Traditionally a reverb pattern is divided into an initial period of *early reflections* (ER) lasting approximately 50 to 80 ms, and a *reverb tail* (RT) that shows a quadratic increase in density and decays exponentially. From an acoustical point of view there is no clear border between the two regimes — the distinction is the result of psychoacoustic effects.

The early reflections, while being discrete, are not heard as a separate sound, rather they 'merge' with the direct sound. They provide our hearing mechanisms with important clues as to the direction and distance of the sound

[1] http://www.ludd.luth.se/ torger/brutefir.html
[2] http://www.affenbande.org/ tapas/jack_convolve
[3] http://users.skynet.be/solaris/linuxaudio
[4] http://users.skynet.be/solaris/linuxaudio/aella

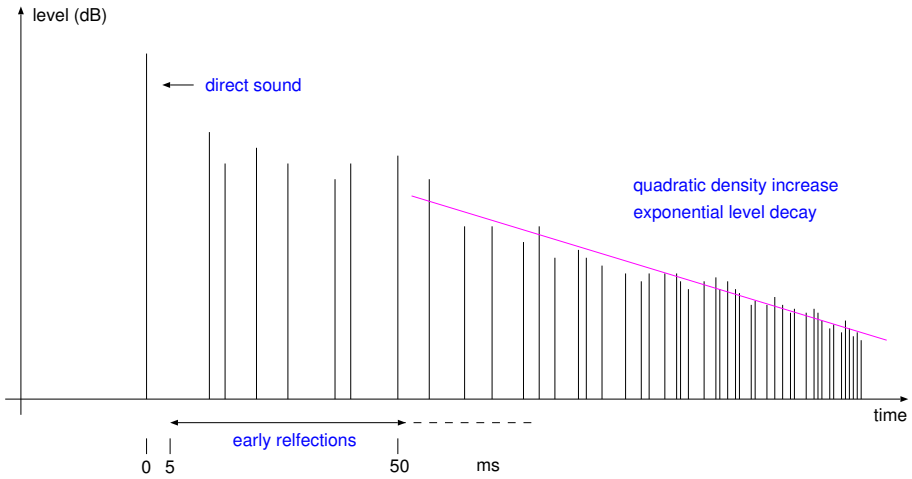

Figure 1: A typical natural reverb pattern

source, and the size and nature of the acoustical environment. The pattern of ER can either reinforce or contradict the results of other mechanisms used by our brain to detect the location of a sound source. For this reason it is important, when recreating an acoustical space, that the ER are consistent with the positioning of sound sources. For a traditional setting with musicians on a stage in front of the audience, at least three (left, centre, right) and preferably more sets of ER should be available to achieve a convincing result. More would be necessary in case the sound sources are all around the listeners.

In concert halls used for classical music, the lateral early reflections (from the side walls) seem to play an important part in how the 'acoustics' are appreciated by the listeners. This is often said to explain why 'shoe-box' concert halls such as the ones at the Amsterdam Concertgebouw or the Musikverein in Vienna are preferred over 'auditorium' shaped ones.

Early reflections very close to the direct sound (less than a few milliseconds) will often result in a 'comb filter' effect, and should be avoided. Discrete reflections that are too far behind the DS or too loud will be heard as a separate 'echo'. These echos occur in many acoustical spaces but not in a good concert hall.

In contrast to the ER, the reverb tail is clearly perceived as a separate sound. A natural reverb tail corresponds to a 'diffuse' sound field with no clear source direction. This doesn't mean that it has no spacial distribution — it has, and this should be reproduced correctly. Of course, the reverb tail will be different for each source (and listener) position, but in general we can not hear this difference — for the RT, only its statistical properties seem to matter. As a result, provided the early reflections are correct, it is possible to use a single reverb tail for all sources.

In most rooms, the RT will show an exponential decay over most of the time. This is not always the case: some spaces with more complex shapes and subspaces (e.g. churches) can produce a significantly different pattern.

3 Requirements for a convolution reverb engine

Taking the observations from the previous section into account it is now possible to define the requirements for a practical convolution based reverb engine.

3.1 Flexibility

In order to be as flexible and general-purpose as possible, the following is needed:

- The engine should allow to combine a number of ER patterns with one or more reverb tails. The number of each should be under control of the user. Separate inputs are required for each ER pattern.
- The relative levels of ER and RT, and the shape of the RT must be controllable. The latter can be used for effects such as e.g. 'gated reverb' that is cut off in the middle of the tail.
- The engine must be able to use a number of formats, from mono to full 3D Ambisonics. It should also support the use of different sample rates.

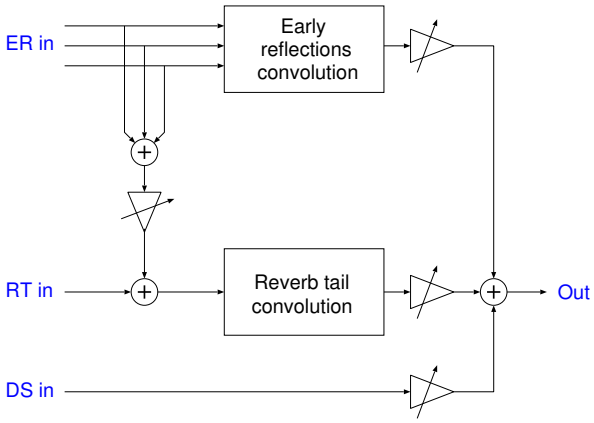

Figure 2: Aella audio structure.

This leads to the general structure shown in fig.2. The ER and RT inputs are always mono, while the DS input and the output will have the number of channels required for the selected format.

In most cases a reverb unit like Aella will be driven from post-fader auxiliary sends from a mixer application and in that case the DS input is normally not used. It is provided for cases where the reverb is used as in insert, e.g. for single channel effects, and to overcome some potential latency problems discussed in the next section.

3.2 Minimal processing delay

The reverb engine should ideally operate with no processing delay, and be usable with all period sizes when running as a JACK client. This turned out to be the most difficult requirement to satisfy. This is discussed in more detail in section 4.

There is even a requirement for *negative processing delay*. This occurs when the output of the reverb is sent back into the same JACK client that is driving it, creating a loop with one period time delay on the returned signal. It *is* possible to compensate for this: remember that the first few (5 to 10) milliseconds after the direct sound should not contain any ER in order to avoid coloration. So provided the period time is small enough, this 'idle time' can be absorbed into the processing delay, provided the DS path is not used. To enable this, Aella provides the option to take the feedback delay into account when loading a reverb pattern.

In case the period time is too long to do this, another solution is to route the direct sound through the reverb and accept a delay on all sound. Aella will always insert the proper delay (not shown in fig.2) into the DS path, depending on its configuration and the period size. Doing this also allows operation with larger processing delay, leading to lower CPU usage.

3.3 Ease of use

It's mainly the requirements from the previous two sections that make general purpose convolution engines impractical for day-to-day work. Having to take all of this into account and keep track of all impulse files, offsets, gains, etc. when writing a configuration script will rapidly drive most less technically inclined users away.

The solution is to automate the complete convolution engine configuration, using only parameters and options that make direct sense to e.g. a musical user. This is what Aella tries to achieve.

Aella first presents a menu of available reverb responses to the user. When one is selected, more information is provided, e.g. in case of a captured real impulse response some info is given on the original space, its reverb time, how the recording was made, etc. A new menu is presented showing the available ER and RT patterns for the selected reverb. The user selects the signal format, the patterns to be used, and some options that enable him or her to trade off CPU load against latency. Immediate feedback about processing delay is provided for the latter. Finally the users clicks the 'LOAD' button, and then all the complex partitioning and scheduling discussed in the next section is performed, and the impulse responses are loaded. The reverb is then ready for use.

Aella uses a single file for each reverb program, containing all the impulse responses (even for different sample rates and formats) and all the extra information that is required. Since this file has to contain binary data (the IR, in floating point format) anyway, and given the author's known aversion to things like XML, it should not come as a surprise that this is a binary file format. It is of course completely open, and (hopefully) flexible enough to allow for future extensions.

Presently these files are generated using a command line tool. In future versions of Aella this function may be integrated into the main program.

4 Using non-uniform partition sizes

The only way to obtain zero processing delay (in the sense that at the end of the process call-

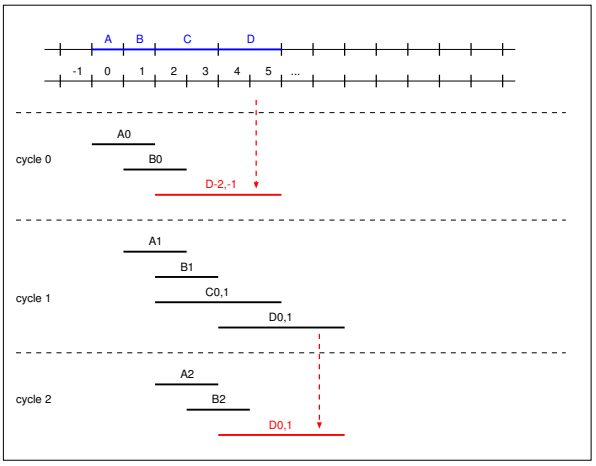

Figure 3: Equal load schema for (1,1,2,2) partitioning

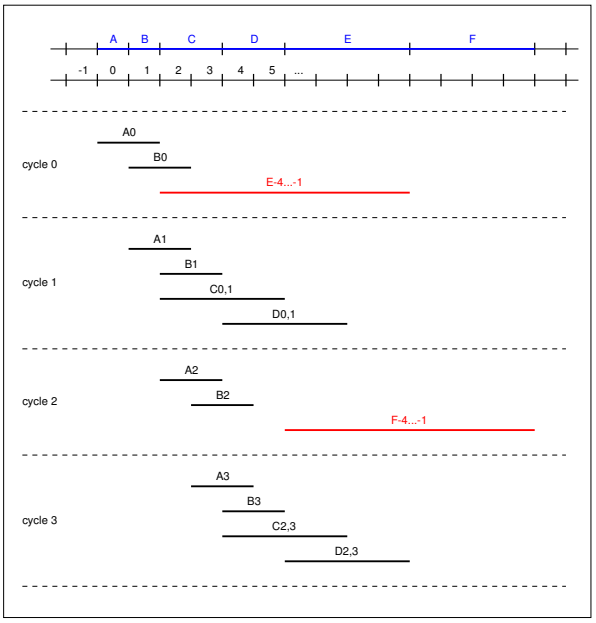

Figure 4: Equal load schema for (1,1,2,2,4,4) partitioning

back the output signal contains the input of the same callback convolved with the first samples of the impulse response) is to use a partition size equal to the period size. For small period sizes it is infeasible to compute the entire convolution using this partition size — the CPU load would be above the limit or unacceptable — so a scheme using a mix of sizes is required.

How to organise a non-uniform partition size convolution so as to obtain the same CPU load for all cycles is known to be a *hard problem*, in the sense that there is no simple algorithm, nor even a complicated one, that provides the optimal solution in all cases. It's an interesting research problem to say the least. One of the referenced papers (Garcia, 2002) will provide a good idea of the state of the art, and of the complexity of some of the proposed solutions.

The problem is further complicated if multiple inputs and outputs are taken into consideration (these can sometimes share the FFT and inverse FFT operations), and even more if the target platform is not a specialised DSP chip with predictable instruction timing but a general purpose PC, and things such as multitasking and cache effects have to be taken into account.

One of the most useful results was published by Bill Gardner as far as ten years ago (Gardner, 1995). Assuming the CPU load is dominated by the multiply-and-add (MAC) phases, and is proportional to the data size, a uniform load can be obtained if the partition sizes follow a certain pattern. Figure 3 provides the simplest example. Here we have four partitions A, B, C, and D with relative sizes 1, 1, 2, 2. The black lines depict the work that can be started in each cycle, for the indicated partition and cycle number(s). For example the line labelled 'C0,1' represents the output generated from the inputs of cycles 0 and 1 and partition C. In the odd numbered cycles we would have three times the work of the even numbered ones. But the computation related to partition D can be delayed by three cycles, so it can be moved to the next even cycle, resulting in an uniform load.

This schema can be generalised for partition sizes following the same doubling pattern. Figure 4 shows the solution for sizes proportional to 1, 1, 2, 2, 4, 4. The red lines in this figure correspond to the computations that have been moved.

There is a limit to what can be done in this way, as the underlying assumptions become invalid very for small period and large partition sizes.

Looking again at fig.4, it is clear that except when partition A is involved, the outputs are required only in the next or in even later cycles. This is even more the case for any later partitions (G, H, ... of relative size 8 or more). So part the work can be delegated to a separate thread running at at lower priority than JACK's client thread (but still in real-time mode). This will increase the complexity of the solution, as the work for later partitions needs to prioritised

in some way in order to ensure timely execution, but in practice this can be managed relatively easily.

In Aella, a mix of both techniques is employed. For the reverb tail convolution a large process delay can be accepted and compensated for by removing the zero valued samples at the start. It uses a minimum size of 1024 frames for the first partition, increasing to a maximum of 8K for later parts (due to cache trashing, nothing is gained by using larger sizes), and all work is done in a lower priority thread.

There is an unexpected potential problem related to moving work to a secondary thread. Imagine the average load is high (say 50%) and the largest partition size is 8K, i.e. there will be something like six such partitions per second. The audio processing by other applications will not suffer from the high load, as most of the work is being done at a lower priority. But the responsiveness of the system e.g. for GUI interaction, or even just for typing text in a terminal, will be severely impaired by a real-time thread executing for several tens of milliseconds at a time. So the work performed in the lower priority thread can not just be started 'in bulk' — it has to be divided into smaller chunks started by triggers from the higher frequency process callback.

For the early reflections convolution the situation is quite different, as it has to provide outputs without any delay. Depending on the period size, Aella will use a minimum partition length of 64 frames, and a schema similar to fig.4 for the first few partitions. Later ones are again moved to a lower priority thread.

The process callback will never *wait* for work done at lower priority to complete — it just assumes it will be ready in time. But it will *check* this condition, and signal a 'soft overrun' (indicated by a flashing light in Aella's GUI) if things go wrong.

For period sizes below 64 frames, Aella will buffer the inputs and outputs, and still try to get the best load distribution. The processing delay will increase due to the buffering, but any idle time before the first early reflection will be used to compensate automatically for the increased delay, in the same way as happens for the feedback delay.

The exact scheme used depends in a quite complicated way on the actual period size and on some user options related to processing delay. This was one of the more difficult to write parts of the application, much more than the real DSP code. The final result is that Aella will be able to operate without any processing delay and at a reasonable CPU load in most practical cases.

References

Guillermo Garcia. 2002. Optimal filter partition for efficient convolution with short input/output delay. *Audio Engineering Society Convention Paper 5660*. 113th AES Convention, Los Angeles October 2002.

William Gardner. 1995. Efficient convolution without input-output delay. *Journal of the Audio Engineering Society*, 43(3):127–136.