Creating reverb effects using granular synthesis

Author Kim Ervik AT kimer@stud.ntnu.no

And Øyvind Brandsegg AT oyvind.brandtsegg@ntnu.no

Abstract

In this article we will explain how we have used Csound to create a reverb effect with granular synthesis. We will look at some commercial varieties of granular reverb, then we will show how to create the same effects using Csound. Due to the flexibility of Csound it is possible to take this idea even further and expand the concept of a granular reverb effect.

Introduction

Granular synthesis has long been available as a sound manipulating and sound generating technique, and currently has seen increasing use in commercial music software. Explained briefly, granular synthesis generates sound based on the additive combination of many very short sonic grains into larger acoustical events. This technique has vast expressive possibilities with its parametric control tied to the time and frequency domain. Some basic examples of granular synthesis parameters are grain rate, grain envelope, grain duration, grain pitch and the waveform inside each grain.

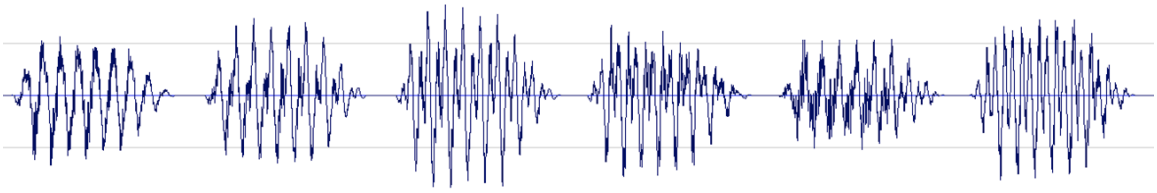

Fig 1: *Granular synthesis*

For a long time it has been common to use pre-sampled sound or complex waveforms as source for granular synthesis. In recent times, due to faster computers, granular synthesis has additionally become available as an audio effect in real time, for instance in a live performance, or as a plug-in in a DAW. A common us of this technique is the “grain delay” effect, found for example in Ableton Live[1]. This effect is similar to a classic delay effect in that it delays the incoming signal, with the parameters delay time, feedback amount and dry/wet amount. It differs from the classical delay effect in the way that it’s possible to chop the delayed signal into grains and use granular synthesis

parameters like grain pitch, grain rate and grain duration. It is also possible to scatter the grains in time with the "spray" parameter, which creates a lush cloud of the repeated signal. If the parameters are tweaked right, this delay effect can sound like a reverb effect.

With Line 6s new stompbox "M" series (M5, M9, and M13) [2] two interesting reverb effects can be found. They are called "particle verb" and "octoverb" and generate a lush cloud output, from the audio input signal. Both these effects contains pitch shift as an integrated part of the algorithm. Knowing this and taking the name "particle verb" in account we can assume that granular synthesis constitute a significant part of the reverb algorithm. In "particle verb" one can also hear a kind of grain scattering, which can also indicate that this is a variant of grain delay.

Classic reverb algorithms

In the physical world, reverb is generated by reflection from surfaces around the sound source giving the *sound object*[1] a short tail. In the early days of recorded audio, sounds were played back into an echo chamber and recorded to produce such effects. Later, tape-delay machines could produce the illusion of early reflections by delaying the incoming signal. A combination of allpass and comb filters has been used to create artificial reverb (e.g. Moorer reverb, Gardner reverb, Freeverb) [3]. Feedback delay networks constitute another well-known technique (e.g. reverbsc in Csound). Currently, also convolution [4] has been used to artificially recreate the response of a physical room or hall.

Granular reverb algorithms

Granular synthesized reverb is similar to the delay effect in the way that it simulates sound reflections by the use of delay, while it resembles reverb in that it also creates reverb-like, lush tails. One could easily term granular reverb more of a special effect than a "realistic" effect.

A granular reverb effect in Csound can be created as a grain delay process, with a delayed signal that is chopped up to smaller grains and treated with granular processing. Scattering of the grains in time creates a cloudy textural tail to the input sound. To facilitate this, audio input is recorded into a table and this table is used as source for granular synthesis. The granular synthesis output is also fed back into the table together with the audio input. This is shown in *PartikkelDly.csd* [5]. To emulate the Line 6 "octaverb", we can use the same example as with the grain delay, but with grain pitch one octave above.

[1] Sound object is described as one isolated sound event, for example a note or a drum hit or the sound of a train passing by (Roads, 2001 [8])

Another way to create a long tail on a sound object is to time stretch it. This is easily done using granular synthesis. In the first track of figure 2, we can see a small sound object. One can "chop" the sound object into grains and stretch the duration of the entire sound object like an accordion, leaving empty space between each grain. This is done in the second and third track of the figure. In track four we can se that by filling the empty spaces between the grains with sound from nearby regions, we get a longer sound object with the same structure as the shorter one, still retaining the original pitch.

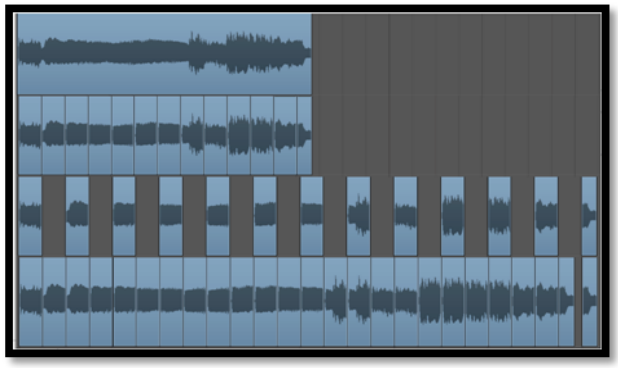

Fig 2: *Granular synthesis time stretch*

When doing time stretch using granular synthesis, we urge to avoid artefacts to the extent possible. The most common artefacts are smearing of transients (due to overlapping long grains), and amplitude modulation sidebands (due to high grain rate and periodic grains). With low grain rates (less than 30 Hz), the pitch of the sound object is perceived through transposition of the source audio, as the original source waveform is not distorted by granular amplitude modulation. We can to some extent avoid the AM effect at higher grain rates too, by using a small random deviation on the sample position for reading grains. One way to create a smooth time stretch effect is by using a grain rate of about 30 – 40 grains per second, overlapping grains (3 overlaps seems to be OK) and a tiny random offset on the sample position. To achieve a broad stereo image one can double the grain rate and mask every other grain to the left and right stereo channels.

Time stretching real time audio

There are obviously some conceptual problems with the idea of time stretching in real time. When a sound is played in its original tempo, at the same time as the same sound is played back in a lower tempo, the lower tempo playback will lag behind the original, and increasingly so over time. In real time processing this is a problem because shortly after the effect instrument is started, the time between the playback position of the original signal and the playback position of the stretched signal is far to large for it to sound like a reverb effect. Letting the stretching process "skip" some parts of the incoming sound can solve this problem, as this will let us align the two playback positions on a periodic basis. This is not ideal either, because it will be unpredictable which part of the incoming signal which will be skipped and which part will be stretched.

A better approach to real time time stretch is to use several buffers and several instances of the time stretch instrument simultaneously (*TimeStretchReverb.csd [6]*). Using the "schedkwhen" Csound opcode we can trigger a Csound instrument to record incoming sound to a buffer (a Csound table). The same opcode can also be used to start a

granulation process of the recorded sound. A metronome and a counter can keep track of which table to write to, and when to start and stop the instances of the recording and playback instruments. In this manner, we can allow several overlapping time stretch instances simultaneously. This can also be seen as a granular process, so we have in effect several crossfading layers of granular synthesis running on top of each other, or *nested granular synthesis*, if you wish. What we mean with nested granular synthesis here is that we have an inner and an outer granular processing loop. The inner loop does the audio time stretching, while the outer loop performs the skipping and crossfading of time stretch layers, "catching up" with real time. It can also be preferable to let the recording instances overlap each other to allow for the random offset on the sample position.

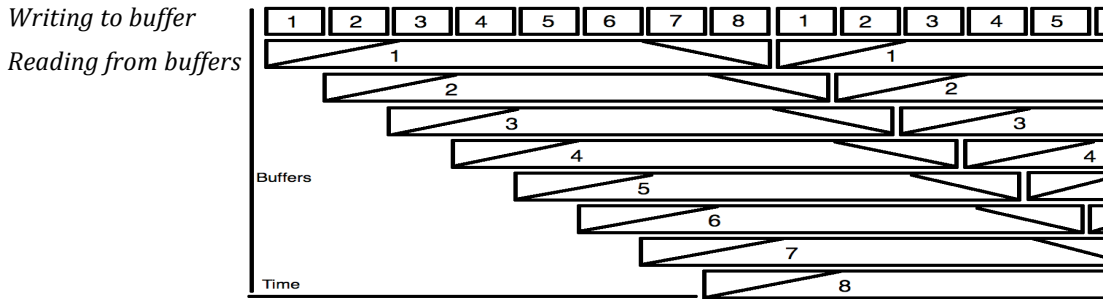

Fig. 3 *Writing audio to 8 different buffers, at the same time as they are played back.*

In our example we've used 8 buffers, recording 0.5 seconds to each buffer. This fraction of sound is time stretched by a factor of 8 so playback of the sound segment lasts 4 seconds (see figure 1). We've also applied an envelope with a slow attack and a slow release to get all the instances to overlap each other smoothly. The parameter values has been tweaked by empirical adjustments, so that the reverb tail is long enough, and at the same time it is not too unpredictable when an incoming sound "returns" from the audio processing. There will be a small periodically changing pre-delay on the reverb effect due to the moving read position on the time stretched playback. Using more layers of overlapping time stretch would make the effect smoother and the variations in pre delay smaller. The reverberation time can be adjusted by controlling the time stretch ratio. The reverb time can be calculated with the formula:

$$T = \frac{t}{2} - 0.5$$

where *T* is the reverb time in seconds and *t* is the time stretch ratio.

Implementation considerations

Our granular reverb implementation uses the partikkel opcode [7]. This opcode was inspired by Curtis Roads book "Microsound" [8] and is capable of generating all the

varieties of granular synthesis described in his book. It is extremely flexible and offers a large set of control parameters.

The “partikkel” opcode is a good choice for implementing granular reverbs, both for its flexibility, and also because one can use up to four sound sources in each opcode instance. A suggestion for optimisation of the reverb algorithm proposed here is to use all the four sound sources of the opcode. This way one could perform the same processing with only two instances of the partikkel opcode instead of eight as we’ve used here. For the purpose of this paper it was thought that the computationally inefficient implementation might provide a clearer separation of the processing stages involved.

With “partikkel” one has “per grain control” over some of the parameters, meaning that one can, for example, specify the pitch and the amp of each grain. This way one can spectrally compose the “reverb cloud”. A neat suggestion is to let every other grain be one octave above the original grains. One could also imagine more exotic variations like pitch sweeping reverbs, and partikkel’s “per grain control” can provide a method for spatial scattering of reverb components.

```
giRecDur    init 0.5    ; LENGTH OF CHUNK TO BE RECORDED FOR STRETCHING
giPlayDur   init 4      ; PLAYBACK TIME OF THE RECORDED CHUNK
                        ; (0.5 SECONDS STRETCHED TO 4 SECONDS)

instr 1
kmetrotime = 2
ktrig metro kmetronome
; ------- COUNTING TABLE NUMBERS FOR TIME STRETCH INSTRUMENT -------
gktablenr = gktablenr + ktrig
if gktablenr > 8 then
gktablenr = 1
endif
schedkwhen ktrig , 0, 3, 2, 0, giRecDur + 0.4, gktablenr ;REC TRIG
if gkplaytablenr > 0 then
schedkwhen ktrig , 0, 8, 3, 0, giPlayDur, gktablenr ;STRETCH TRIG
endif
endin
```

Fig 4 :*Time stretch trigger instrument.*

Conclusion

In this paper we have shown how to create reverb-like effects using granular synthesis. We have also explained the concept of a time stretching reverb using nested granular techniques. We have seen how some commercial grain reverb works, and we have shown how to do create a grain reverb in csound. We have also shown that with the flexibility of csound it is possible to take the idea of a granular reverb even further, creating highly animated and modulated reverb algorithms. Finally, we have suggested some possible optimizations on the provided example implementations.

References

[1] Ableton – http://www.ableton.com/live-8

[2] Line 6 – http://line6.com/m9/models.html

[3] Smith, Julius O. (2006) *Physical Audio Signal Processing*, web resource

http://www.dsprelated.com/dspbooks/pasp/Comb_Filters.html

http://www.dsprelated.com/dspbooks/pasp/Schroeder_Allpas_Sections.html

http://www.dsprelated.com/dspbooks/pasp/Freeverb.html

http://www.dsprelated.com/dspbooks/pasp/FDN_Reverberation.html

[4] Boulanger, Richard (2000) *The Csound book,* page 507. MIT Press, Cambridge, Massachusetts.

[5] Ervik, Kim csound exsample nr1: PartikkelDly.csd

http://folk.ntnu.no/kimer/csoundconference2011/PartikkelDly.csd

[6] Ervik, Kim csound exsample nr2: TimeStretchReverb.csd

http://folk.ntnu.no/kimer/csoundconference2011/TimeStretchReverb.csd

[7] Brandtsegg, Ø. Saue, S. and Johansen, T. (2011) *Particle synthesis, a unified model for granular synthesis.* Linux Audio Conference 2011.

http://lac.linuxaudio.org/2011/papers/39.pdf

[8] Roads, Curtis (2001) *Microsound*. MIT Press, Cambridge, Massachusetts.