

Bay Area Csound Users Group

Meeting Topic: Reverbs

Steven Yi
11.11.2003

I. Csound's Builtin Reverbs

Table 1 - Csound Opcodes for Reverb

alpass	Reverberates an input signal with a flat frequency response.
babo	A physical model reverberator (ball within a box)
comb	Reverberates an input signal with a "colored" frequency response.
nestedap	Three different nested all-pass filters.
nreverb (reverb2)	A reverberator consisting of 6 parallel comb-lowpass filters.
reverb	Reverberates an input signal with a "natural room" frequency response.
valpass	Variably reverberates an input signal with a flat frequency response.
vcomb	Variably reverberates an input signal with a "colored" frequency response.

Alpass, babo, comb, valpass, vcomb are generally used as building blocks in physical modeling instruments or as building blocks to reverb designs.

nreverb, reverb, and babo are generally used as reverberating units in themselves, though certainly can be used within larger designs.

Also used often are delay lines in conjunction with filters, often feeding back into the system.

II. Josep M Comajuncosas's Reverb Pack

Josep M Comajuncosas has collected and organized various reverb implementations together into a reverb macro pack, available at:

<http://csounds.com/jmc/Processors/Processors.htm>

Not only has he done a fantastic job of organizing the reverbs and making them easy to use, he's also provided a great demo orc/sco that runs through each reverb macro with the same audio input to make it easy to compare!

The pack can be considered in two parts: reverbs using opcode and IR files for convolution. For the reverbs using opcodes, Josep has collected the reverbs into an include file, `rvb.h`, and organized them all as `#define`'s. The pack comes fully documented with usage tips and hints, and the example orc/sco show how these reverb macros are to be used. Generally, it's as simple as:

```
instr 100
    $xrvb2s(ga3'aoutL'aoutR'.2'1'1000)
    outs aoutL, aoutR
endin
```

III. Convolution Reverb

Table 2 - Csound Opcodes For Convolution

convolve(convle)	Convolve a signal and an impulse response.
dconv	A direct convolution opcode

“In [mathematics](#) and in particular, [functional analysis](#), the **convolution** (German: *Faltung*) is a mathematical [operator](#) which takes two [functions](#) f and g and produces a third function that in a sense represents the amount of overlap between f and a reversed and translated version of g . “ (<http://en2.wikipedia.org/wiki/Convolution>).

A space has characteristics which define it audibly, such that when a sound is generated in that space, the response of the space will affect the perceived sound, which can be described with words like “the sound reverberates for 2 seconds with an emphasis on low frequencies”, etc. One way to attempt to create this characteristic of a space is to try to synthesize it with delay lines and filters, trying to approximate that space. Another method known as convolution can be used to directly capture characteristics of a space and allow you to imprint that character on other signals.

The process of reverberating with convolution starts off by taking a sound imprint of a space (or generating a synthetic imprint). This imprint is called an **impulse response**. The process usually involves using a starter pistol or electronically generated noise (the impulse) and recording the sound that results from it (the response).

After the impulse response is generated, it can be convolved with an signal to give that signal the character of the space that the impulse response was taken in.

Impulse responses can be found all over the internet and are generally available as wav files. To use these in Csound, you need to use the cvanal utility (I recommend using the one built into csound by calling “csound -U cvanal for reasons of insuring your getting the right number of bits for the version of csound you're using) to generate a .cv file, which will then be used by convolution opcodes in Csound (convolve).

Some things to note when using convolution in Csound:

1)The analyzed .cv files are dependent on two things: the endian of your processor (either little(intel, amd) or big endian(motorola, ibm), and -bit version of Csound you use (32-bit csound or 64-bit csound64). So, if you create your .cv file on a Mac, it won't work on a Windows or Linux machine. If you create a .cv file on Windows it will work on Linux (but not LinuxPPC, which runs on Mac hardware). If you create a .cv file with 32-bit csound, it won't work with csound64, and vice versa.

2)Using the convolve opcode introduces a delay in the output signal equal to the length of the impulse response, i.e. if you're reverb is 3 seconds long, expect the sound going into

the reverb to start 3 seconds later and reverberate for 3 seconds more. If you're expecting to mix wet and dry signals, you should delay the dry signal for the same amount of time that the wet signal will be delayed by the convolve opcode.

[convolve vs. dconv]

convolve, I believe, uses FFT convolution, while **dconv** uses direct convolution. FFT convolution is faster to process than when the IR is longer than 64 points (IR's for reverb are generally MUCH larger than 64 points), but delays the signal, thus making it unusable for realtime use (unless you want that delay...). **dconv** uses data stored in a table for convolution which can be altered in realtime and introduces no delay in signal.

(Information for the above consulted from <http://www.dspguide.com/>, chapters 6, 7, and 18.)

The impulse responses for this presentation have come from:

1. JMC's Reverb Pack
2. Noise Vault (<http://www.noisevault.com>)
3. Voxengo (<http://www.voxengo.com/>)