Reverb Algorithm [closed]

Asked 13 years, 9 months ago Modified 5 years, 6 months ago

Viewed 45k times

32

Closed. This question is seeking recommendations for software libraries, tutorials, tools, books, or other off-site resources. It does not meet Stack Overflow guidelines. It is not currently accepting answers.

We don't allow questions seeking recommendations for software libraries, tutorials, tools, books, or other off-site resources. You can edit the question so it can be answered with facts and citations.

Closed 5 years ago.

Improve this question

I'm looking for a simple or commented reverb algorithm, even in pseudocode would help a lot.

I've found a couple, but the code tends to be rather esoteric and hard to follow.

c++ signal-processing

Share Follow

edited Jul 16, 2014 at 11:45

Sam
7,398 ● 16 ● 47 ● 68

asked Mar 15, 2011 at 22:45

Reu
1,287 ● 3 ● 15 ● 31

4 Is this for creating a real-time reverb effect, or applying a reverb effect to an existing recording (like a WAV file)? Real-time effects are much more complicated to implement, yet they make up most of the samples I've seen out there.
– MusiGenesis Mar 15, 2011 at 22:49

@MusiGenesis A real-time one ideally, but one to modify an existing recording might help. – Reu Mar 15, 2011 at 22:51

7 DSP isn't really the kind of thing you can learn by reading the code. Code is the application of mathematical models, and it will be more productive to understand those models than to read the implementations. Unless you just want to drop-in some code to create reverb, but then you wouldn't be here, right? – tenfour Mar 15, 2011 at 23:15

1 Nice run down of reverb in layperson's terms here soundonsound.com/sos/Oct01/articles/advancedreverb1.asp
– Matthew Lock May 19, 2014 at 5:45

1 This is a good example showing that programming languages themselves only solve the easy problems.
– Samuel Danielson Aug 28, 2015 at 15:07

4 Answers

Sorted by: Highest score (default)

Here is a very simple implementation of a "delay line" which will produce a reverb effect in an existing array (C#,

`buffer` is `short[]`):

```
int delayMilliseconds = 500; // half a second
int delaySamples =
    (int)((float)delayMilliseconds * 44.1f); // assume
float decay = 0.5f;
for (int i = 0; i < buffer.length - delaySamples; i++)
{
    // WARNING: overflow potential
    buffer[i + delaySamples] += (short)((float)buffer[
}
```

Basically, you take the value of each sample, multiply it by the decay parameter and add the result to the value in the buffer `delaySamples` away.

This will produce a true "reverb" effect, as each sound will be heard multiple times with declining amplitude. To get a simpler echo effect (where each sound is repeated only once) you use basically the same code, only run the `for` loop in reverse.

Update: the word "reverb" in this context has two common usages. My code sample above produces a classic reverb effect common in cartoons, whereas in a musical application the term is used to mean reverberation, or more generally the creation of artificial spatial effects.

A big reason the literature on reverberation is so difficult to understand is that creating a good spatial effect requires much more complicated algorithms than my sample method here. However, most electronic spatial effects are built up using multiple delay lines, so this

sample hopefully illustrates the basics of what's going on. To produce a really good effect, you can (or should) also muddy the reverb's output using FFT or even simple blurring.

Update 2: Here are a few tips for multiple-delay-line reverb design:

- Choose delay values that won't positively interfere with each other (in the wave sense). For example, if you have one delay at 500ms and a second at 250ms, there will be many spots that have echoes from both lines, producing an unrealistic effect. It's common to multiply a base delay by different prime numbers in order to help ensure that this overlap doesn't happen.
- In a large room (in the real world), when you make a noise you will tend to hear a few immediate (a few milliseconds) sharp echoes that are relatively undistorted, followed by a larger, fainter "cloud" of echoes. You can achieve this effect cheaply by using a few backwards-running delay lines to create the initial echoes and a few full reverb lines plus some blurring to create the "cloud".
- The absolute *best* trick (and I almost feel like I don't want to give this one up, but what the hell) only works if your audio is stereo. If you slightly vary the parameters of your delay lines between the left and right channels (e.g. 490ms for the left channel and 513ms for the right, or .273 decay for the left and

.2631 for the right), you'll produce a much more realistic-sounding reverb.

Share Follow

edited Mar 15, 2011 at 23:26

answered Mar 15, 2011 at 22:59

MusiGenesis

75.3k ● 41 ● 197 ● 338

Right, so I do that, how do I calculate how many delaylines to add and where to put them? Thanks for your answer – Reu Mar 15, 2011 at 23:04

1 You just do what sounds nice, honestly. These two url's taught me everything I've learned on the subject: ccrma.stanford.edu/~jos/pasp/Artificial_Reverberation.html earlevel.com/main/1997/01/19/a-bit-about-reverb – Samuel Danielson Aug 28, 2015 at 15:16

27

Digital reverbs generally come in two flavors.

- **Convolution Reverbs** convolve an impulse response and a input signal. The impulse response is often a recording of a real room or other reverberation source. The character of the reverb is defined by the impulse response. As such, convolution reverbs usually provide limited means of adjusting the reverb character.
- **Algorithmic Reverbs** mimic reverb with a network of delays, filters and feedback. Different schemes will

combine these basic building blocks in different ways. Much of the art is in knowing how to tune the network. Algorithmic reverbs usually expose several parameters to the end user so the reverb character can be adjusted to suit.

The A Bit About Reverb post at EarLevel is a great introduction to the subject. It explains the differences between convolution and algorithmic reverbs and shows some details on how each might be implemented.

Physical Audio Signal Processing by Julius O. Smith has a chapter on reverb algorithms, including a section dedicated to the Freeverb algorithm. Skimming over that might help when searching for some source code examples.

Sean Costello's Valhalla blog is full of interesting reverb tidbits.

Share Follow

edited Jun 19, 2019 at 2:18

answered Mar 16, 2011 at 1:24

Shannon Matthews

10.3k ● 7 ● 46 ● 77

6

What you need is the impulse response of the room or reverb chamber which you want to model or simulate. The full impulse response will include all the multiple and multi-path echos. The length of the impulse response will

be roughly equal to the length of time (in samples) it takes for an impulse sound to completely decay below audible threshold or given noise floor.

Given an impulse vector of length N, you could produce an audio output sample by vector multiplication of the input vector (made up of the current audio input sample concatenated with the previous N-1 input samples) by the impulse vector, with appropriate scaling.

Some people simplify this by assuming most taps (down to all but 1) in the impulse response are zero, and just using a few scaled delay lines for the remaining echos which are then added into the output.

For even more realistic reverb, you might want to use different impulse responses for each ear, and have the response vary a bit with head position. A head movement of as little as a quarter inch might vary the position of peaks in the impulse response by 1 sample (at 44.1k rates).

Share Follow

answered Mar 16, 2011 at 0:28

hotpaw2

70.6k ● 15 ● 92 ● 155

+1 because you're right, but this explanation might be "esoteric and hard to follow". :) – MusiGenesis Mar 16, 2011 at 18:23

1 +1 because what's worth doing is worth doing well. – Eric Brotto Mar 16, 2011 at 19:03

2

You can use GVerb. Get the code from here.GVerb is a LADSPA plug-in, you can go here if you want to know something about LADSPA.

Here is the wiki for GVerb , including explaining of the parameters and some instant reverb settings.

Also we can use it directly in Objc:

```
ty_gverb        *_verb;
_verb = gverb_new(16000.f, 41.f, 40.0f, 7.0f, 0.5f, 0.
AudioSampleType *samples =
(AudioSampleType*)dataBuffer.mBuffers[0].mData;//Audio
Render or ExtAuidoFileReader
float lval,rval;
for (int i = 0; i< fileLengthFrames; i++) {
     float value = (float)samples[i] / 32768.f;//from
     gverb_do(_verb, value, &lval, &rval);
     samples[i] =  (SInt16)(lval * 32767.f);//float to
}
```

GVerb is a mono effect but if you want a stereo effect you could run each channel through the effect separately and then pan and mix the processed signals with the dry signals as required

Share Follow

edited Dec 26, 2012 at 4:35

answered Dec 26, 2012 at 4:29

Joe Qian

493 ● 6 ● 13

amazing! but it is a costly library. Could not use it real time – cs guy Jul 31, 2021 at 10:58