Reverb/Spatial Effects

Spatial Effects

The final set of effects we look at are effects that change to spatial localisation of sound.

There a many examples of this type of processing we will study two briefly:

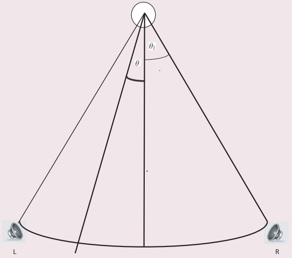
- **Panning:** in stereo audio
- **Reverb:** a small selection of reverb algorithms

Panning

What is Panning?

Mapping a **monophonic** sound source across a stereo audio image such that the sound starts in one speaker (**R**) and is moved to the other speaker (**L**) in n time steps.

- We assume that we listening in a central position so that the angle between two speakers is the same, i.e. we subtend an angle $2\theta_l$ between **2** speakers.
  We assume for simplicity, in this case that $\theta_l = 45^\circ$.



Panning Geometry

Simple applications of basic rotation geometry:

- We seek to obtain to signals one for each Left (**L**) and Right (**R**) channel, the gains of which, g_L and g_R, are applied to steer the sound across the stereo audio image.
- This can be achieved by simple 2D rotation, where the angle we sweep is θ:

$$\mathbf{A}_\theta = \begin{bmatrix} \cos\theta & \sin\theta \\ -\sin\theta & \cos\theta \end{bmatrix}$$

and

$$\begin{bmatrix} g_L \\ g_R \end{bmatrix} = \mathbf{A}_\theta . \mathbf{x}$$

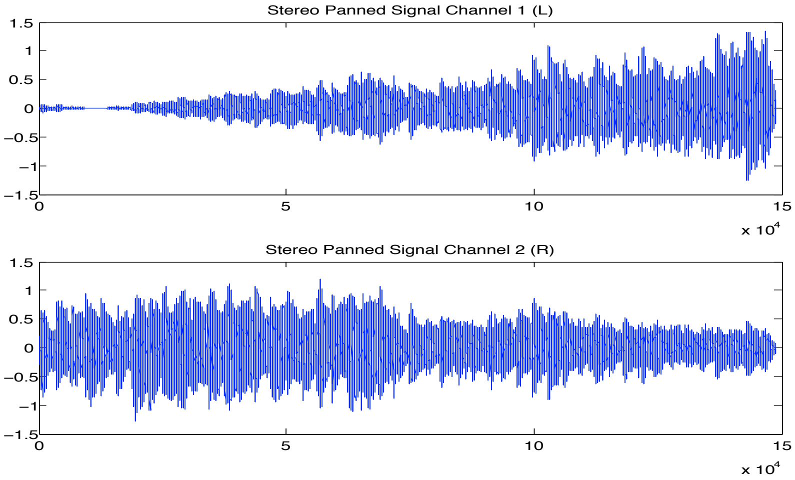
where $\mathbf{x}$ is a segment of mono audio

MATLAB Panning Example

matpan.m:

```
% read the sample waveform
filename='acoustic.wav';
[monox,Fs] = audioread(filename);

initial_angle = -40; %in degrees
final_angle = 40;    %in degrees
segments = 32;
angle_increment = (initial_angle - final_angle)/segments * pi / 180;
lenseg = floor(length(monox)/segments) - 1;
pointer = 1;
angle = initial_angle * pi / 180; %in radians

y=[[];[]]; % Preallocate
for i=1:segments
 A =[cos(angle), sin(angle); -sin(angle), cos(angle)];
 stereox =
  [monox(pointer:pointer+lenseg)'; monox(pointer:pointer+lenseg)'];
 y = [y, A * stereox];
 angle = angle + angle_increment; pointer = pointer + lenseg;
end;
% write output .......
```

MATLAB Panning Example Output



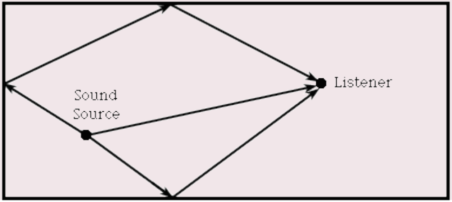
Click image or here to hear: original audio,
stereo panned audio.

Reverb

Reverberation

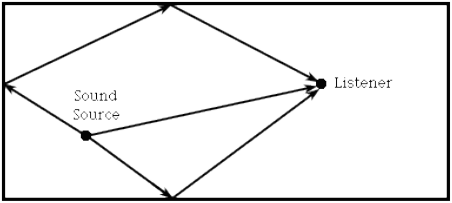
Reverb (for short) is probably one of the most heavily used effects in audio.

- *Reverberation* is the result of the many reflections of a sound that occur in a room.
  - From any sound source, say a speaker of your stereo, there is a direct path that the sounds covers to reach our ears.
  - Sound waves can also take a slightly longer path by reflecting off a wall or the ceiling, before arriving at your ears.



The Spaciousness of a Room

- A reflected sound wave like this will arrive **a little later** than the direct sound, since it travels a longer distance, and is generally a little weaker, as the walls and other surfaces in the room will absorb some of the sound energy.
- Reflected waves can again bounce off another wall before arriving at your ears, and so on.
- This series of delayed and attenuated sound waves is what we call **reverb**, and this is what creates the *spaciousness* sound of a room.
- Clearly large rooms such as concert halls/cathedrals will have a much more spaciousness reverb than a living room or bathroom.



Reverb v. Echo

Is reverb just a series of echoes?

Echo — implies a distinct, delayed version of a sound,

- *E.g.* as you would hear with a delay more than one or two-tenths of a second.

Reverb — each delayed sound wave arrives in such a short period of time that we do not perceive each reflection as a copy of the original sound.

- Even though we can't discern every reflection, we still hear the effect that the entire series of reflections has.

Reverb v. Delay

Can a simple delay device with feedback produce reverberation?

Delay: can produce a similar effect **but** there is one very important feature that a simple delay unit will **not** produce:

- The rate of arriving reflections **changes** over **time**.
- Delay can only simulate reflections with a fixed time interval.

Reverb: for a short period after the direct sound, there is generally a set of well defined directional reflections that are directly related to the shape and size of the room, and the position of the source and listener in the room.

- These are the *early reflections*
- After the early reflections, the rate of the arriving reflections increases greatly are more random and difficult to relate to the physical characteristics of the room.
  This is called the *diffuse reverberation*, or the *late reflections*.
- Diffuse reverberation is the **primary factor** establishing a room's 'spaciousness' — it decays exponentially in good concert halls.

Reverb Simulation

There are many ways to simulate reverb.

Two Broad classes of approach studied here (there are others):

- **Filter Bank**/**Delay Line** methods
- **Convolution**/**Impulse Response** methods

Schroeder's Reverberator

Schroeder's model of Reverb (1961)

- Early digital reverberation algorithms tried to mimic the a rooms reverberation by using primarily consisted of **two types** of infinite impulse response (IIR) filters with the **aim** to make the output **gradually decay**.

  Comb filter: usually in parallel banks

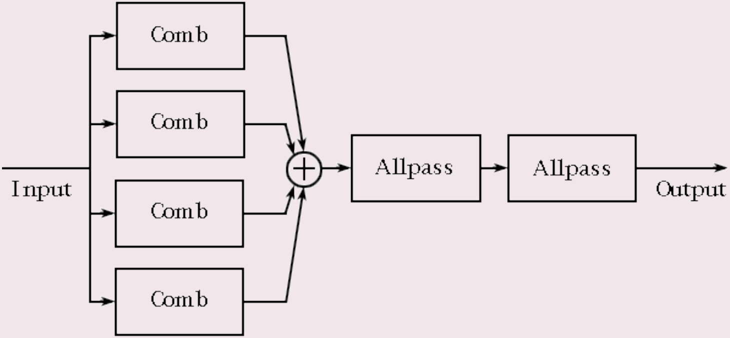
  Allpass filter: usually sequentially after comb filter banks

Much of the early work on digital reverb was performed by Schroeder.

Schroeder's Reverberator (Cont.)

Schroeder's reverberator example (one of a few variations):

This particular design uses **four comb filters** and **two allpass filters**:



Note:This design does not create the increasing arrival rate of reflections, and is rather primitive when compared to current algorithms.

MATLAB Schroeder Reverb Example

Simple Schroeder: schroeder1.m:

- *n* **allpass** filters in series. (**No Comb Filters yet**)

```
function [y,b,a]=schroeder1(x,n,g,d,k)
% This is a reverberator based on Schroeder's design which consists of n
% allpass filters in series.
%
% The structure is:  [y,b,a] = schroeder1(x,n,g,d,k)
%
% where x = the input signal
%       n = the number of allpass filters
%       g = the gain of the allpass filters
%           (should be less than 1 for stability)
%       d = a vector which contains the delay length of each allpass filter
%       k = the gain factor of the direct signal
%       y = the output signal
%       b = the numerator coefficients of the transfer function
%       a = the denominator coefficients of the transfer function
%
% note: Make sure that d is the same length as n.
%
```

MATLAB Schroeder Reverb Example (Cont.)

schroeder1.m (Cont.):

```
% send the input signal through the first allpass filter
[y,b,a] = allpass(x,g,d(1));

% send the output of each allpass filter to the input of
% the next allpass filter
for i = 2:n,
   [y,b1,a1] = allpass(y,g,d(i));
   [b,a] = seriescoefficients(b1,a1,b,a);
end

% add the scaled direct signal
y = y + k*x;

% normalize the output signal
y = y/max(y);
```

MATLAB Schroeder Reverb Example (Cont.)

The support files to do the filtering (for following reverb methods also) are here:

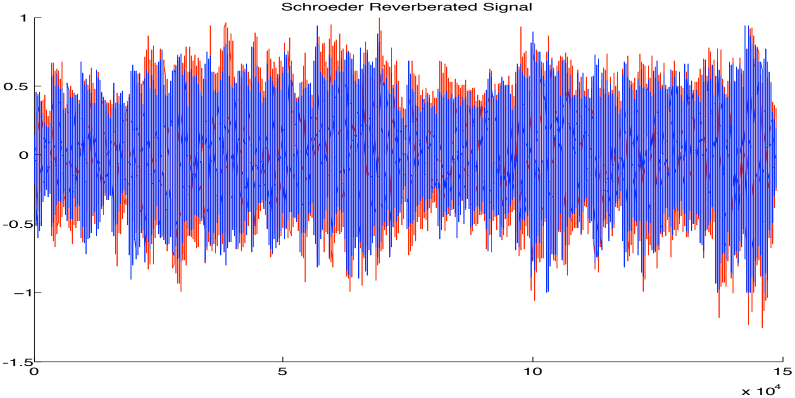
- delay.m,
- seriescoefficients.m,
- parallelcoefficients.m,
- fbcomb.m,
- ffcomb.m,
- allpass.m

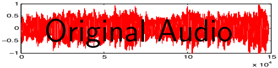
MATLAB Schroeder Reverb (Cont.)

Example call, reverb_schroeder_eg.m:

```
filename='acoustic.wav'; % Read the waveform
[x,Fs] = audioread(filename);

% Set the number of allpass filters
n = 6;
% Set the gain of the allpass filters
g = 0.9;
% Set delay of each allpass filter in number of samples
% Compute a random set of milliseconds and use sample rate
rand('state',sum(100*clock))
d = floor(0.05*rand([1,n])*Fs);
%set gain of direct signal
k= 0.2;

[y b a] = schroeder1(x,n,g,d,k);

% write output
audiowrite('out_schroederreverb.wav', y,Fs);
```

The input signal (blue) and reverberated signal (red):





Click images or here to hear: original audio, reverberated audio.

MATLAB *Classic* Schroeder Reverb Example

Classic Schroeder Reverb, schroeder2.m:

- **4 comb** and **2 allpass** filters.

```
function [y,b,a]=schroeder2(x,cg,cd,ag,ad,k)
% This is a reverberator based on Schroeder's design which consists of 4
% parallel feedback comb filters in series with 2  allpass filters.
%
% The structure is:  [y,b,a] = schroeder2(x,cg,cd,ag,ad,k)
% where x = the input signal
%      cg = a vector of length 4 which contains the gain of each of the
%              comb filters (should be less than 1)
%      cd = a vector of length 4 which contains the delay of each of the
%             comb filters
%      ag = the gain of the allpass filters (should be less than 1)
%      ad = a vector of length 2 which contains the delay of each of the
%              allpass filters
%      k = the gain factor of the direct signal
%      y = the output signal
%      b = the numerator coefficients of the transfer function
%      a = the denominator coefficients of the transfer function
```

MATLAB *Classic* Schroeder Reverb Example (Cont.)

Classic Schroeder Reverb, schroeder2.m (Cont.):

```
% send the input to each of the 4 comb filters separately
[outcomb1,b1,a1] = fbcomb(x,cg(1),cd(1));
[outcomb2,b2,a2] = fbcomb(x,cg(2),cd(2));
[outcomb3,b3,a3] = fbcomb(x,cg(3),cd(3));
[outcomb4,b4,a4] = fbcomb(x,cg(4),cd(4));


% sum the ouptut of the 4 comb filters
apinput = outcomb1 + outcomb2 + outcomb3 + outcomb4;

%find the combined filter coefficients of the the comb filters
[b,a]=parallelcoefficients(b1,a1,b2,a2);
[b,a]=parallelcoefficients(b,a,b3,a3);
[b,a]=parallelcoefficients(b,a,b4,a4);
```

MATLAB *Classic* Schroeder Reverb Example (Cont.)

Classic Schroeder Reverb, schroeder2.m (Cont.):

```
% send the output of the comb filters to the allpass filters
[y,b5,a5] = allpass(apinput,ag,ad(1));
[y,b6,a6] = allpass(y,ag,ad(2));

%find the combined filter coefficients of the the comb filters in
% series with the allpass filters
[b,a]=seriescoefficients(b,a,b5,a5);
[b,a]=seriescoefficients(b,a,b6,a6);

% add the scaled direct signal
y = y + k*x;

% normalize the output signal
y = y/max(y);
```

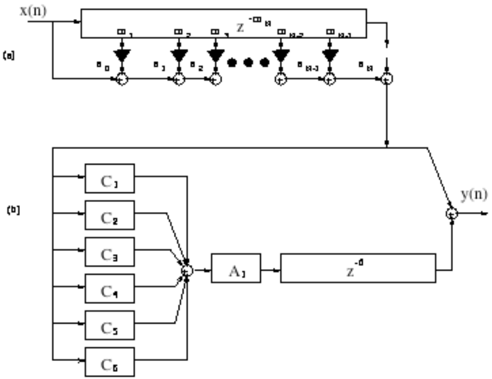
See forthcoming Lab Class for examples of this effect and extensions.

Moorer's Reverberator

Moorer's reverberator (1976): Build's on Schroeder

- **Parallel comb filters** with **different** delay lengths are used to simulate modes of a room, and sound reflecting between parallel walls
- **Allpass** filters to **increase** the reflection density (diffusion).
- **Lowpass** filters **inserted** in the feedback loops to alter the reverberation time as a **function of frequency**
  - **Shorter** reverberation time at **higher** frequencies is caused by air absorption and reflectivity characteristics of wall).
  - Implement a **DC-attenuation**, and a frequency dependent attenuation.
  - Encode a difference in each comb filter because their coefficients **depend** on the **delay line length**.

Moorer's Reverberator



(a) **Tapped delay lines simulate** *early reflections* — forwarded to (b)

(b) **Parallel comb filters** which are then **allpass** filtered and delayed before being added back to early reflections — simulates **diffuse reverberation**

MATLAB Moorer Reverb

moorer.m:

```
function [y,b,a]=moorer(x,cg,cg1,cd,ag,ad,k)
% This is a reverberator based on Moorer's design which consists of 6
% parallel feedback comb filters  (each with a low pass filter in the
% feedback loop) in series with an all pass filter.
%
% The structure is:  [y,b,a] = moorer(x,cg,cg1,cd,ag,ad,k)
%
% where x = the input signal
%       cg = a vector of length 6 which contains g2/(1-g1) (this should be less
%               than 1 for stability), where g2 is the feedback gain of each of the
%               comb filters and g1 is from the following parameter
%       cg1 = a vector of length 6 which contains the gain of the low pass
%               filters in the feedback loop of each of the comb filters (should be
%               less than 1 for stability)
%       cd = a vector of length 6 which contains the delay of each of comb filter
%       ag = the gain of the allpass filter (should be less than 1 for stability)
%       ad = the delay of the allpass filter
%       k = the gain factor of the direct signal
%       y = the output signal
%       b = the numerator coefficients of the transfer function
%       a = the denominator coefficients of the transfer function
%
```

MATLAB Moorer Reverb (Cont.)

moorer.m (Cont.):

```
% send the input to each of the 6 comb filters separately
[outcomb1,b1,a1] = lpcomb(x,cg(1),cg1(1),cd(1));
[outcomb2,b2,a2] = lpcomb(x,cg(2),cg1(2),cd(2));
[outcomb3,b3,a3] = lpcomb(x,cg(3),cg1(3),cd(3));
[outcomb4,b4,a4] = lpcomb(x,cg(4),cg1(4),cd(4));
[outcomb5,b5,a5] = lpcomb(x,cg(5),cg1(5),cd(5));
[outcomb6,b6,a6] = lpcomb(x,cg(6),cg1(6),cd(6));

% sum the ouptut of the 6 comb filters
apinput = outcomb1 + outcomb2 + outcomb3 + outcomb4 + outcomb5 + outcomb6;

%find the combined filter coefficients of the the comb filters
[b,a]=parallelcoefficients(b1,a1,b2,a2);
[b,a]=parallelcoefficients(b,a,b3,a3);
[b,a]=parallelcoefficients(b,a,b4,a4);
[b,a]=parallelcoefficients(b,a,b5,a5);
[b,a]=parallelcoefficients(b,a,b6,a6);
```

MATLAB Moorer Reverb (Cont.)

moorer.m (Cont.):

```
% send the output of the comb filters to the allpass filter
[y,b7,a7] = allpass(apinput,ag,ad);

%find the combined filter coefficients of the the comb filters in series
% with the allpass filters
[b,a]=seriescoefficients(b,a,b7,a7);

% add the scaled direct signal
y = y + k*x;

% normalize the output signal
y = y/max(y);
```

MATLAB Moorer Reverb (Cont.)

Example call, reverb_moorer_eg.m:

```
% reverb_moorer_eg.m
% Script to call the Moorer Reverb Algoritm

% read the sample waveform
filename='../acoustic.wav';
[x,Fs] = audioread(filename);

% Call moorer reverb
%set delay of each comb filter
%set delay of each allpass filter in number of samples
%Compute a random set of milliseconds and use sample rate
rand('state',sum(100*clock))
cd = floor(0.05*rand([1,6])*Fs);

% set gains of 6 comb pass filters
g1 = 0.5*ones(1,6);
%set feedback of each comb filter
g2 = 0.5*ones(1,6);
```

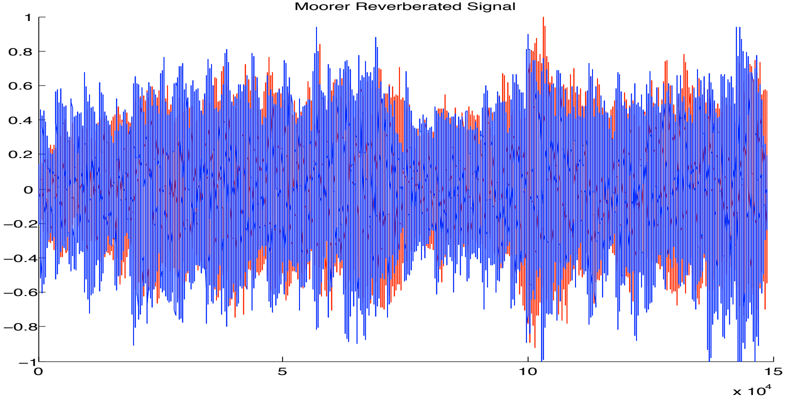
MATLAB Moorer Reverb (Cont.)

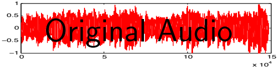
reverb_moorer_eg.m:

```
% set input cg and cg1 for moorer function see help moorer
cg = g2./(1-g1);
cg1 = g1;

%set gain of allpass filter
ag = 0.7;
%set delay of allpass filter
ad = 0.08*Fs;
%set direct signal gain
k = 0.5;

[y b a] = moorer(x,cg,cg1,cd,ag,ad,k);

% write output
audiowrite('out_moorerreverb.wav', y,Fs);
```

MATLAB Moorer Reverb (Cont.)

The input signal (blue) and reverberated signal (red):





Click here to hear: original audio, Moorer reverberated audio.

Convolution Reverb

Convolution Reverb: Basic Idea

If the impulse response of the room is known then the most faithful reverberation method would be to **convolve** it with the input signal.

- Due to the usual length of the target response it is not feasible to implement this with filters — several hundreds of taps in the filters would be required.
- However, **convolution readily implemented** using **FFT**:
  - Recall: The **discrete convolution** formula:

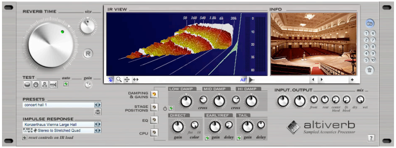
$$y(n) = \sum_{k=-\infty}^{\infty} x(k).h(n-k) = x(n) * h(n)$$

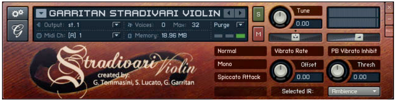
  - Recall: The **convolution theorem** which states that:
    If $f(x)$ and $g(x)$ are two functions with Fourier transforms $F(u)$ and $G(u)$, then the Fourier transform of the **convolution** $f(x) * g(x)$ *is simply the product of the Fourier transforms of the two functions, $F(u)G(u)$.*

Commercial Convolution Reverbs

Commercial Convolution Reverbs

- Altiverb — one of the first mainstream convolution reverb effects units
- Most sample based synthesisers (E.g. Kontakt, Intakt) provide some convolution reverb effect
- Dedicated sample-based software instruments such as Garritan Violin and PianoTeq Piano use convolution not only for reverb simulation but also to simulate key responses of the instruments body vibration.



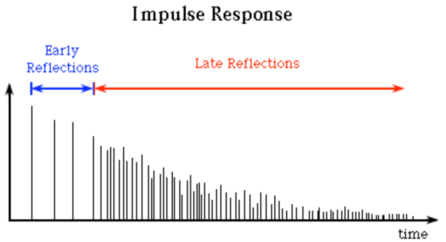


Room Impulse Responses

Record a Room Impulse

Apart from providing a high (professional) quality recording of a room's impulse response, the process of using an impulse response is quite straightforward:

- Record a short impulse (gun shot,drum hit, hand clap) in the room.
- Room impulse responses can be simulated in software also.
- The impulse encodes the rooms reverb characteristics:



MATLAB Convolution Reverb (1)

Let's develop a fast convolution routine: fconv.m

```
function [y]=fconv(x, h)
%   FCONV Fast Convolution
%   [y] = FCONV(x, h) convolves x and h,
%         and normalizes the output   to +-1.
%       x = input vector
%       h = input vector
%

Ly=length(x)+length(h)-1;  %
Ly2=pow2(nextpow2(Ly));    % Find smallest power of 2
                           % that is > Ly
X=fft(x, Ly2);                   % Fast Fourier transform
H=fft(h, Ly2);                   % Fast Fourier transform
Y=X.*H;                           % DO CONVOLUTION
y=real(ifft(Y, Ly2));      % Inverse fast Fourier transform
y=y(1:1:Ly);               % Take just the first N elements
y=y/max(abs(y));           % Normalize the output
```

See also: MATLAB built in function `conv()`

MATLAB Convolution Reverb (2)

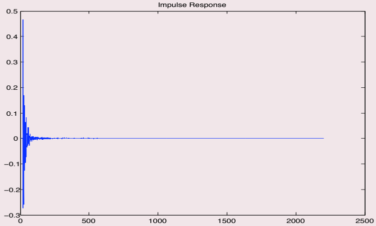
reverb_convolution_eg.m

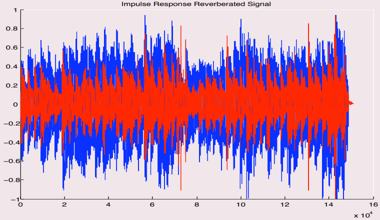
```
% reverb_convolution_eg.m
% Script to call implement Convolution Reverb

% read the sample waveform
filename='../acoustic.wav';
[x,Fs] = audioread(filename);

% read the impulse response waveform
filename='impulse_room.wav';
[imp,Fsimp] = audioread(filename);

% Do convolution with FFT
y = fconv(x,imp);

% write output
audiowrite('out_IRreverb.wav', y,Fs);
```

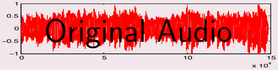
MATLAB Convolution Reverb (3)

Some example results:

Living Room Impulse Response Convolution Reverb:



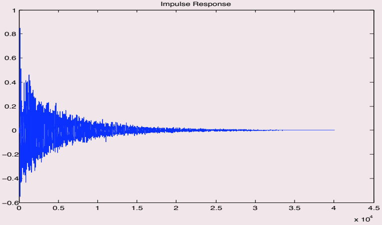


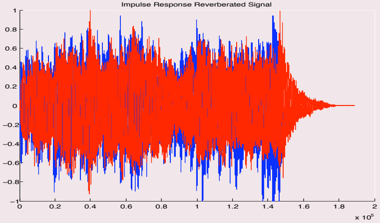


Click on above images or here to hear: original audio, room impulse response audio, room impulse reverberated audio.

MATLAB Convolution Reverb (4)

Cathedral Impulse Response Convolution Reverb:



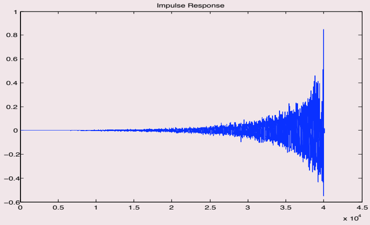


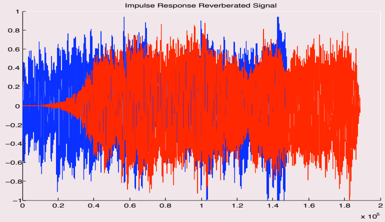
Click on above images or here to hear: original audio, cathedral impulse response audio, cathedral reverberated audio.

MATLAB Convolution Reverb (5)

It is easy to implement some **other (odd?) effects** also

Reverse Cathedral Impulse Response Convolution Reverb:



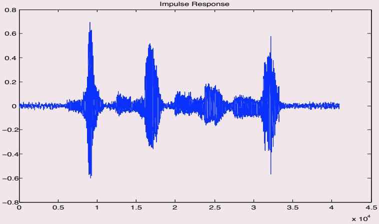


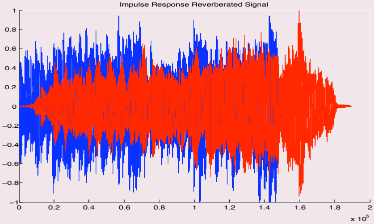
Click on above images or here to hear: original audio, reverse cathedral impulse response audio, reverse cathedral reverberated audio.

MATLAB Convolution Reverb (6)

You can basically convolve with anything.

Speech Impulse Response Convolution Reverb!:





Click on above images or here to hear: original audio, speech *'impulse response'* audio, speech impulse reverberated audio.