CM2202: Scientific Computing and Multimedia Applications
Fourier Transform 2: Digital Signal and Image Processing Applications and Examples

Prof. David Marshall

School of Computer Science & Informatics

February 25, 2014

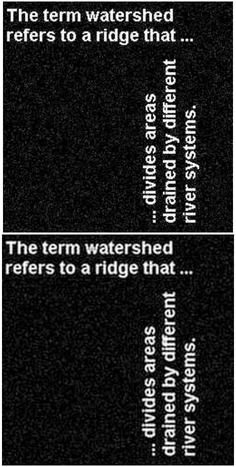
Filtering in the Frequency Domain

Low Pass Filter

***Example**: Audio Hiss, 'Salt and Pepper' noise in images,*

Noise:

- The idea with **noise Filtering** is to reduce various spurious effects of a **local nature** in the image, caused perhaps by
  - **noise** in the acquisition system,
  - arising as a result of **transmission** of the data, for example from a space probe utilising a low-power transmitter.



Frequency Space Filtering Methods

Low Pass Filtering — Remove Noise

Noise = High Frequencies:

- In audio data many spurious peaks in over a short timescale.
- In an image means there are many rapid transitions (over a short distance) in intensity from high to low and back again or vice versa, as faulty pixels are encountered.
- **Not all high frequency data noise though!**

Therefore **noise** will contribute heavily to the **high frequency** components of the signal when it is **analysed** in **Fourier space**.

Thus if we **reduce** the **high frequency** components — **Low-Pass Filter** should (if tuned properly) **reduce** the amount of noise in the data.

(Low-pass) Filtering in the Fourier Space

Low Pass Filtering with the Fourier Transform

We **filter** in Fourier space by computing

$$G(u, v) = H(u, v)F(u, v)$$

where:

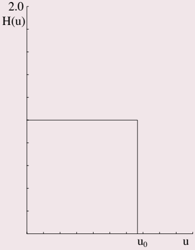
- $F(u, v)$ is the **Fourier transform** of the **original** image,
- $H(u, v)$ is a filter function, designed to reduce high frequencies, and
- $G(u, v)$ is the **Fourier transform of the improved image**.
- **Inverse Fourier transform** $G(u, v)$ to get $g(x, y)$ our **improved image**

Ideal Low-Pass Filter

We need to design or compute $H(u, v)$

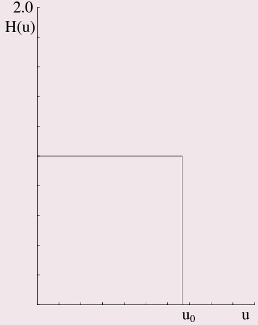
- If we know $h(x, y)$ or have a discrete sample of $h(x, y)$ can compute its Fourier Transform
- Can simply design simple filters in Frequency Space

The simplest sort of filter to use is an *ideal low-pass filter*, which in one dimension appears as :



Ideal Low-Pass Filter (2)

How the Low Pass Filter Works with Frequencies



This is a $h(x, y)$ function which is **1** for u between 0 and u_0, the *cut-off frequency*, and **zero** elsewhere.

- So all frequency space information **above** u_0 is **discarded**, and all information **below** u_0 is **kept**.
- A **very simple** computational process.

Ideal 2D Low-Pass Filter

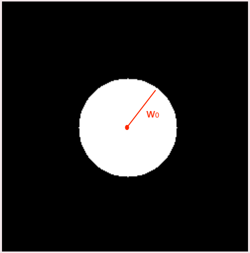
Ideal 2D Low-Pass Filter

The two dimensional version of this is the Low-Pass Filter:

$$H(u,v) = \begin{cases} 1 & \textbf{if } \sqrt{u^2+v^2} \le w_0 \\ 0 & \textbf{otherwise,} \end{cases}$$

where w_0 is now the **cut-off frequency** for **both** dimensions.

- Thus, **all** frequencies **inside** a **radius** w_0 are **kept**, and **all** others **discarded**.



Not So Ideal Low-Pass Filter? (1)

In practice, the ideal Low-Pass Filter is no so ideal

The **problem** with this filter is that as well as noise there may be **useful** high frequency content:

- In **audio**: plenty of other high frequency content: high pitches, rustles, scrapes, wind, mechanical noises, cymbal crashes etc.
- In **images**: **edges** (places of rapid transition from light to dark) also significantly contribute to the high frequency components.

Choosing the **most appropriate** cut-off frequency is not so easy

- Similar problem to choosing a threshold in **image thresholding**.

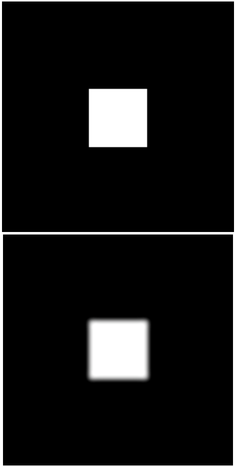
Not So Ideal Low-Pass Filter? (2)

What if you set the wrong value for the cut-off frequency?

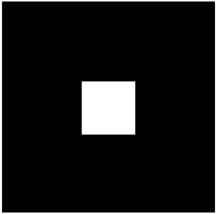
If you **choose the wrong cut-off frequency** an ideal low-pass filter will tend to *blur* the data:

- High audio frequencies become muffled
- Edges in images become blurred.

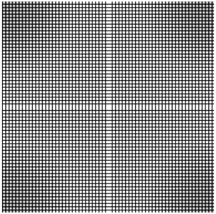
The lower the cut-off frequency is made, the more pronounced this effect becomes in *useful data content*



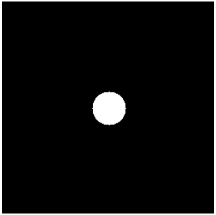
Ideal Low Pass Filter Example 1



(a) Input Image



(b) Image Spectra



(c) Ideal Low Pass Filter

(d) Filtered Image

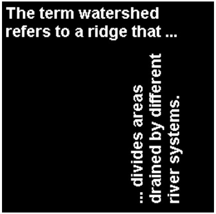
Ideal Low-Pass Filter Example 1 MATLAB Code

low pass.m:

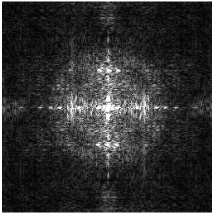
```
% Create a white box on a
% black background image
M = 256; N = 256;
image = zeros(M,N)
box = ones(64,64);
%box at centre
image(97:160,97:160) = box;

% Show Image

figure(1);
imshow(image);

% compute fft and display its spectra

F=fft2(double(image));
figure(2);
imshow(abs(fftshift(F)));
```

```
% Compute Ideal Low Pass Filter
u0 = 20; % set cut off frequency

u=0:(M-1);
v=0:(N-1);
idx=find(u>M/2);
u(idx)=u(idx)-M;
idy=find(v>N/2);
v(idy)=v(idy)-N;
[V,U]=meshgrid(v,u);
D=sqrt(U.^2+V.^2);
H=double(D<=u0);

% display
figure(3);
imshow(fftshift(H));

% Apply filter and do inverse FFT
G=H.*F;
g=real(ifft2(double(G)));

% Show Result
figure(4);
imshow(g);
```

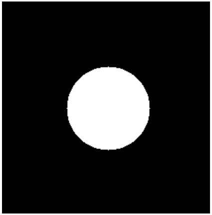
Ideal Low Pass Filter Example 2



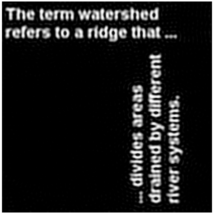
(a) Input Image



(b) Image Spectra



(c) Ideal Low-Pass Filter



(d) Filtered Image

Ideal Low-Pass Filter Example 2 MATLAB Code

lowpass2.m:

```
% read in MATLAB demo text image
image = imread('text.png');
[M N] = size(image)


% Show Image

figure(1);
imshow(image);

% compute fft and display its spectra

F=fft2(double(image));
figure(2);
imshow(abs(fftshift(F))/256);
```

```
% Compute Ideal Low Pass Filter
u0 = 50; % set cut off frequency

u=0:(M-1);
v=0:(N-1);
idx=find(u>M/2);
u(idx)=u(idx)-M;
idy=find(v>N/2);
v(idy)=v(idy)-N;
[V,U]=meshgrid(v,u);
D=sqrt(U.^2+V.^2);
H=double(D<=u0);

% display
figure(3);
imshow(fftshift(H));

% Apply filter and do inverse FFT
G=H.*F;
g=real(ifft2(double(G)));

% Show Result
figure(4);
imshow(g);
```

Low-Pass Butterworth Filter (1)

We introduced the **Butterworth Filter** with **IIR/FIR Filters** (**Temporal Domain Filtering**). Let's now study it in more detail.

- Much easier to visualise in Frequency space

2D Low-Pass Butterworth Filter

Another popular (and general) filter is the **Butterworth low pass filter**.

In the 2D case, $H(u, v)$ takes the form

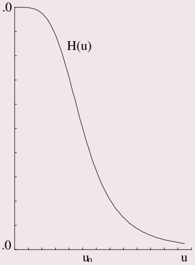
$$H(u, v) = \frac{1}{1 + \left[(u^2 + v^2)/w_0^2\right]^n},$$

where n is called the **order** of the filter.

Low-Pass Butterworth Filter (2)

Visualising the 1D Low-Pass Butterworth Filter

This keeps some of the high frequency information, as illustrated by the second order **one dimensional** Butterworth filter:



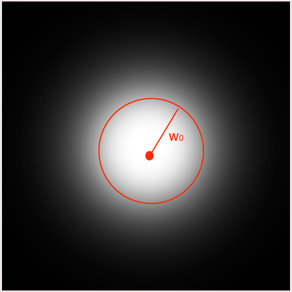
Consequently **reduces** the **blurring**.

- **Blurring** the **filter** — Butterworth is essentially a **smoothed** top hat functions — **reduces blurring by** the filter.

Low-Pass Butterworth Filter (3)

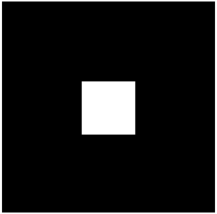
Visualising the 2D Low-Pass Butterworth Filter

The **2D second order** Butterworth filter looks like this:

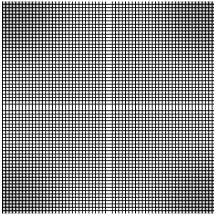


- Note this is **blurred circle** — blurring of the ideal 2D Low-Pass Filter.

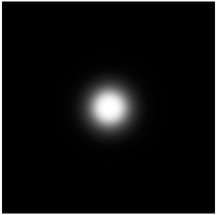
Butterworth Low Pass Filter Example 1 (1)



(a) Input Image



(b) Image Spectra



(c) Butterworth Low-Pass Filter

(d) Filtered Image

Butterworth Low-Pass Filter Example 1 (2)

Comparison of Ideal and Butterworth Low Pass Filter:

Ideal Low-Pass

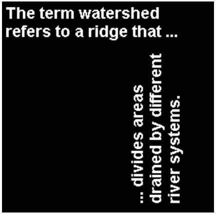
Butterworth Low-Pass

Butterworth Low-Pass Filter Example 1 (3)

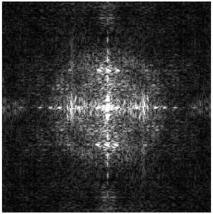
butterworth.m:

```
% Load Image and Compute FFT as
%  in Ideal Low Pass Filter Example 1
 .......
% Compute Butterworth Low Pass Filter
u0 = 20; % set cut off frequency

u=0:(M-1);
v=0:(N-1);
idx=find(u>M/2);
u(idx)=u(idx)-M;
idy=find(v>N/2);
v(idy)=v(idy)-N;
[V,U]=meshgrid(v,u);

for i = 1: M
    for j = 1:N
      %Apply a 2nd order Butterworth
      UVw = double((U(i,j)*U(i,j) + V(i,j)*V(i,j))/(u0*u0));
      H(i,j) = 1/(1 + UVw*UVw);
    end
end
% Display Filter and Filtered Image as before
```

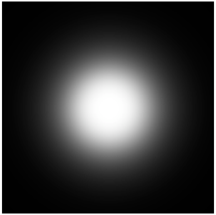
Butterworth Low-Pass Butterworth Filter Example 2 (1)



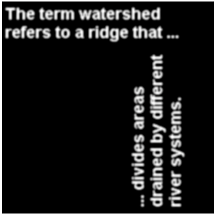
(a) Input Image



(b) Image Spectra



(c) Butterworth Low-Pass Filter

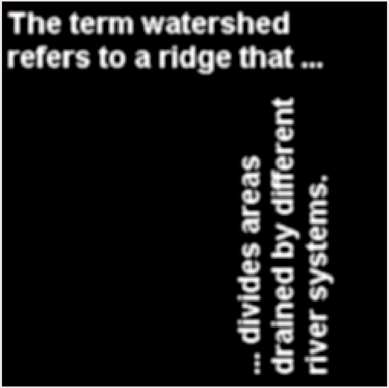


(d) Filtered Image

Butterworth Low-Pass Filter Example 2 (2)

Comparison of Ideal and Butterworth Low-Pass Filter:

Ideal Low Pass



Butterworth Low-Pass

Butterworth Low Pass Filter Example 2 MATLAB (3)

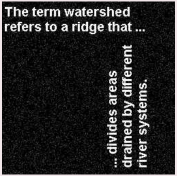
butterworth2.m:

```
% Load Image and Compute FFT as in Ideal Low Pass Filter
% Example 2
.......
% Compute Butterworth Low Pass Filter
u0 = 50; % set cut off frequency

u=0:(M-1);
v=0:(N-1);
idx=find(u>M/2);
u(idx)=u(idx)-M;
idy=find(v>N/2);
v(idy)=v(idy)-N;
[V,U]=meshgrid(v,u);

for i = 1: M
    for j = 1:N
      %Apply a 2nd order Butterworth
      UVw = double((U(i,j)*U(i,j) + V(i,j)*V(i,j))/(u0*u0));
      H(i,j) = 1/(1 + UVw*UVw);
    end
end
% Display Filter and Filtered Image as before
```

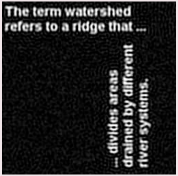
Low Pass Filtering Noisy Images

How to create noise and results of Low Pass Filtering

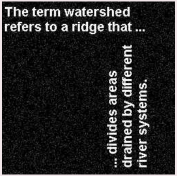
Use Matlab function, `imnoise()` to add noise to image (lowpass.m, lowpass2.m):



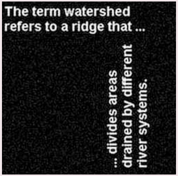
(a) Input Noisy Image



(b) Deconvolved Noisy Image (Low Cut Off)



(c) Input Noisy Image



(d) Deconvolved Noisy Image (Higher Cut Off)

Other Filters

Other Filters

High-Pass Filters — opposite of low-pass, select high frequencies, attenuate those **below** u_0

Band-pass — allow frequencies in a range $u_0 \ldots u_1$ attenuate those outside this range

Band-reject — opposite of band-pass, attenuate frequencies within $u_0 \ldots u_1$ **select** those **outside** this range

Notch — attenuate frequencies in a narrow bandwidth around cut-off frequency, u_0

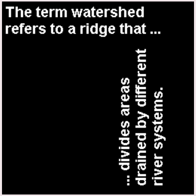
Resonator — amplify frequencies in a narrow bandwidth around cut-off frequency, u_0

Other filters exist that essentially are a combination/variation of the above

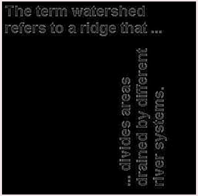
High Pass Filtering

Easy to Implement from the above Low Pass Filter

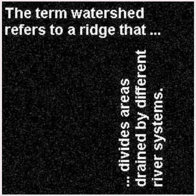
A High Pass Filter is usually defined as **1 - low pass** $= 1 - H$:



(a) Input Image



(b) High Pass Filtered Image



(c) Input Noisy Image

(d) High Pass Filtered Noisy Image

Convolution

Many Useful Applications of Convolution

Several important audio and optical effects can be described in terms of convolutions.

- FIltering — In fact the **above** **Fourier filtering** is applying **convolutions** of a **low pass filter** where the equations are Fourier Transforms of real space equivalents.
- Deblurring — **high pass** filtering
- Reverb — impulse response convolution (**more soon**).

Note we have seen a discrete **real domain** example of Convolution with **Edge Detection**.

Formal Definition of 1D Convolution:

Let us examine the concepts using 1D continuous functions.

The convolution of two functions $f(x)$ and $g(x)$, written $f(x) * g(x)$, is defined by the integral

$$f(x) * g(x) = \int_{-\infty}^{\infty} f(\alpha) g(x - \alpha)\, d\alpha.$$

- $*$ is the mathematical **notation** for **convolution**.

No Fourier Transform in sight here — **but wait**!

1D Convolution Real Domain Example (1)

Convolution of Two Top Hat Functions

For example, let us take two **top hat functions**:

Let $f(\alpha)$ be the top hat function shown:

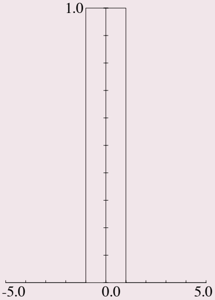
$$f(\alpha) = \begin{cases} 1 & \textbf{if } |\alpha| \leq 1 \\ 0 & \textbf{otherwise,} \end{cases}$$

and let $g(\alpha)$ be as shown in next slide, defined by

$$g(\alpha) = \begin{cases} 1/2 & \textbf{if } 0 \leq \alpha \leq 1 \\ 0 & \textbf{otherwise.} \end{cases}$$

1D Convolution Example (2)

Our Two Top Hat Functions Plots

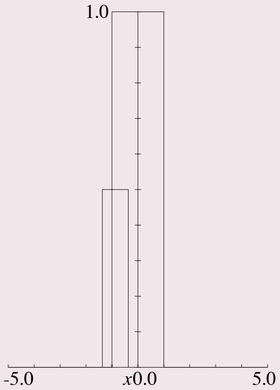


$$f(\alpha) = \begin{cases} 1 & \textbf{if } |\alpha| \leq 1 \\ 0 & \textbf{otherwise,} \end{cases} \qquad g(\alpha) = \begin{cases} 1/2 & \textbf{if } 0 \leq \alpha \leq 1 \\ 0 & \textbf{otherwise.} \end{cases}$$

1D Convolution Example (3)

The Convolution Process: Graphical Interpretation

- $g(-\alpha)$ is the **reflection** of this function in the **vertical** y-axis,
- $g(x-\alpha)$ is the **latter shifted** to the right by a **distance** x.
- Thus for a given value of x, $f(\alpha)g(x-\alpha)$ integrated over all α is the area of overlap of these two top hats, as $f(\alpha)$ has unit height.
- An example is shown for x in the range $-1 \leq x \leq 0$ opposite



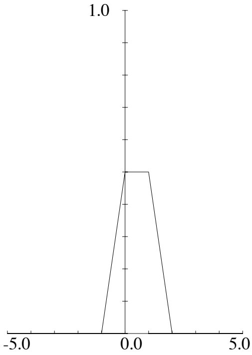
1D Convolution Example (4)

So the solution is:

If we now consider x moving from $-\infty$ to $+\infty$, we can see that

- For $x \leq -1$ or $x \geq 2$, there is **no overlap**;
- As x goes from -1 to 0 the area of overlap **steadily increases** from 0 to $1/2$;
- As x **increases** from0 to 1, the overlap area remains at $1/2$;
- Finally as x increases from 1 to 2, the overlap area steadily **decreases** again from $1/2$ to 0.
- Thus the convolution of $f(x)$ and $g(x)$, $f(x) * g(x)$, in this case has the form shown on next slide

1D Convolution Example (5)

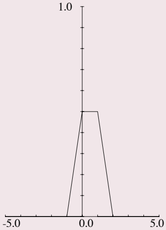


Result of $f(x) * g(x)$

1D Convolution Example (6)

Mathematically the convolution is expressed by:

$$f(x) * g(x) = \begin{cases} (x+1)/2 & \textbf{if } -1 \leq x \leq 0 \\ 1/2 & \textbf{if } 0 \leq x \leq 1 \\ 1 - x/2 & \textbf{if } 1 \leq x \leq 2 \\ 0 & \textbf{otherwise.} \end{cases}$$



Fourier Transforms and Convolution

Convolution Theorem: Convolution in Frequency Space is Easy

One **major** reason that Fourier transforms are so important in signal/image processing is 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, $F(u)G(u)$.

Recall our Low Pass Filter Example (MATLAB CODE)

```
% Apply filter
G=H.*F;
```

Where `F` was the Fourier transform of the image, `H` the filter

Computing Convolutions with the Fourier Transform

Example Applications:

- To apply some reverb to an audio signal.
- To compensate for a less than ideal image capture system.

Deconvolution: Compensating for undesirable effects

To do this **fast convolution** we simply:

- Take the **Fourier transform** of the **audio/imperfect image**,
- Take the **Fourier transform** of the **function describing the effect** of the system,
- To **remove**/**compensate** for effect: Divide by the effect's Fourier Transform to obtain the Fourier transform of the 'ideal' audio/image.
- **Inverse** Fourier transform to **recover** the new **improved** audio/image.

This process is sometimes referred to as **deconvolution**.

Image Deblurring Deconvolution Example

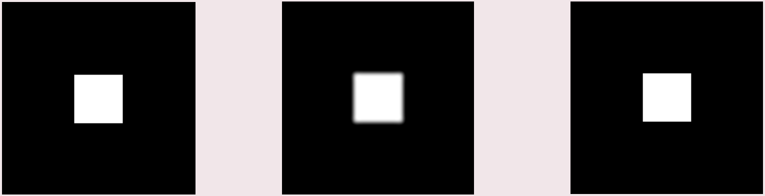
Inverting our Previous Low-Pass Filter

Recall our Low Pass (Butterworth) Filter example of a few slides ago: butterworth.m:
deconv.m and deconv2.m reuses this code and adds a deconvolution stage:

- Our computed butterworth low pass filter, H is our blurring function
- So to simply invert this we can divide (as opposed to multiply) by H with the blurred image G — effectively a **high pass filter**

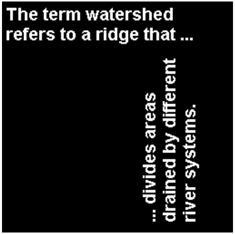
```
Ghigh = G./H;
ghigh=real(ifft2(double(Ghigh)));
figure(5)
imshow(ghigh)
```

- In this ideal example we clearly get F back and to get the image simply to inverse Fourier Transfer.
- In the real world we don't really know the **exact blurring function** H so things are not so easy.

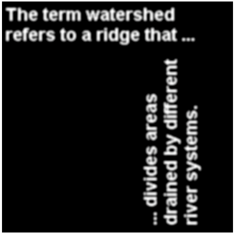


(a) Input Image (b) Blurred Low-Pass Filtered Image (c) Deconvolved Image

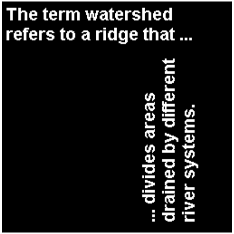
deconv2.m results



(a) Input Image

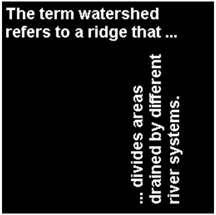


(b) Blurred Low-Pass Filtered Image

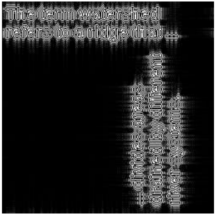


(c) Deconvolved Image

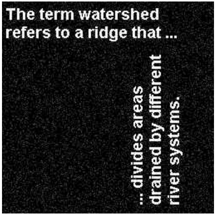
Deconvolution is not always that simple!



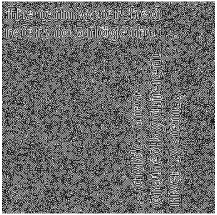
(a) Input Image



(b) Deconvolved Image



(c) Input Noisy Image



(d) Deconvolved Noisy Image

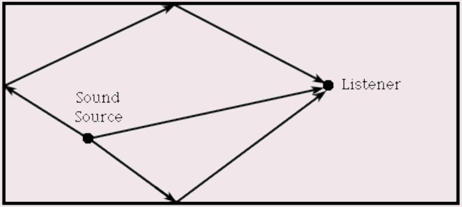
Convolution Reverb: Back to Audio

What is Reverb?

Reverberation (**reverb** for short) is probably one of the most heavily used effects in music.

Reverberation is the result of the many reflections of a sound that occur in a room.

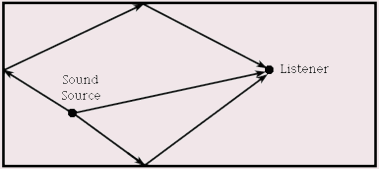
- From any sound source, say a speaker of your stereo, there is a direct path that the sounds covers to reach our ears.
- Sound waves can also take a slightly longer path by reflecting off a wall or the ceiling, before arriving at your ears.



The Spaciousness of a Room

What is Reverb? (Cont.)

- 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 Simulation

How to implement a Digital Audio Effect

There are a few ways to simulate reverb.

We will only study two reverb methods of approach here (there are others):

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

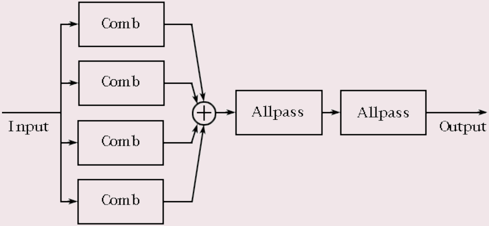
Schroeder's Reverberator

Reverb via Filters and Delays

- Early digital reverberation algorithms tried to mimic a room's reverberation by primarily using **two types** of infinite impulse response (IIR) filters.

  Comb filter — usually in parallel banks
  Allpass filter — usually sequentially after comb filter banks

- A delay is (set via the feedback loops allpass filter) aims to make the output would gradually decay.

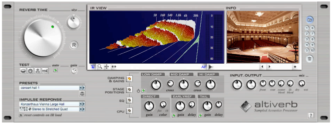


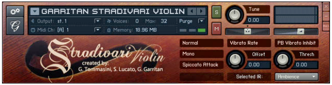
MATLAB Code: schroeder1.m

Convolution Reverb Implementation

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.



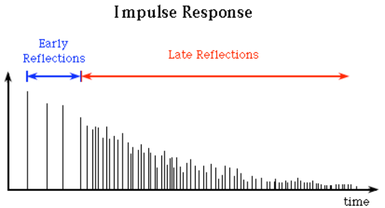


Room Impulse Responses

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)

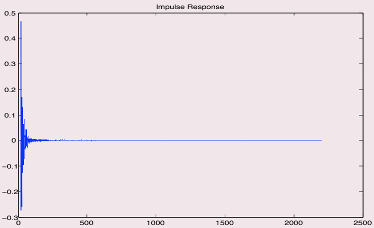
reverb_convolution_eg.m

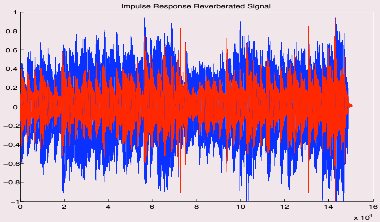
```
% reverb_convolution_eg.m
% Script to call implement Convolution Reverb

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

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

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

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

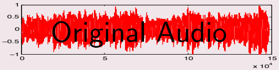
MATLAB Convolution Reverb (3)

Some example results:

Living Room Impulse Response Convolution Reverb:



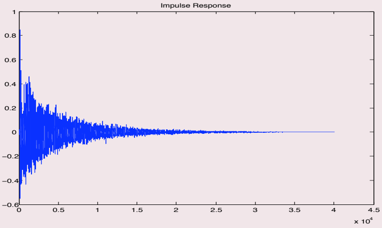


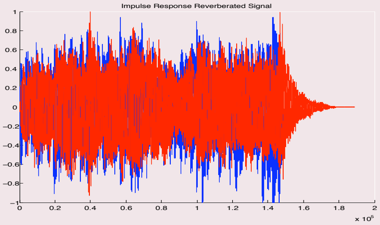


Click on above images or here to hear: original audio, room impulse response audio, room impulse reverberated audio.

MATLAB Convolution Reverb (4)

Cathedral Impulse Response Convolution Reverb:



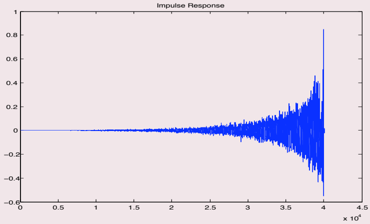


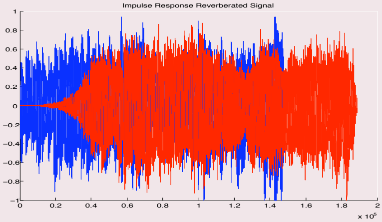
Click on above images or here to hear: original audio, cathedral impulse response audio, cathedral reverberated audio.

MATLAB Convolution Reverb (5)

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

Reverse Cathedral Impulse Response Convolution Reverb:



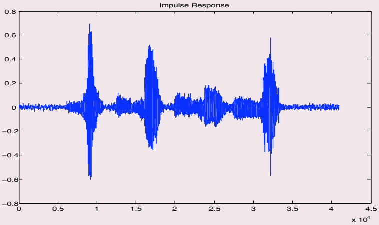


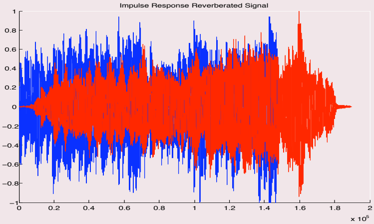
Click on above images or here to hear: original audio, reverse cathedral impulse response audio, reverse cathedral reverberated audio.

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.

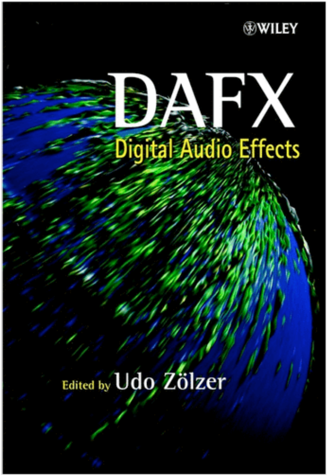
Further Reading

DAFX: Digital Audio Effects
Udo Zolzer
John Wiley and Sons Ltd , 2002
(ISBN-13: 978-0471490784)

Excellent coverage of audio signal processing effects and synthesis plus a lot more

All MATLAB examples

Copies in library



Further Reading

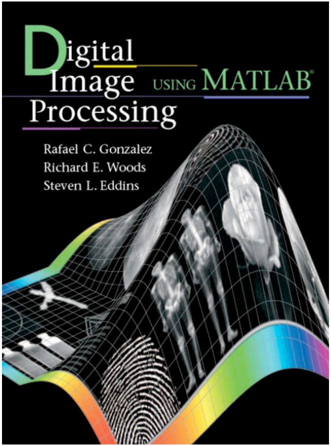
Digital Image Processing Using MATLAB
Rafael C. Gonzalez,
Richard E. Woods,
and Steven L. Eddins
Prentice Hall, 2004
(ISBN-13: 978-0130085191)

Excellent coverage of Image processing examples plus a lot more

All MATLAB examples

Useful for CM0311 Image Processing

Copies in library



Additional Examples, MATLAB Code and Reading

Additional Examples

Audio Synthesis — CM0340 Module Notes

Audio Effects — CM0340 Module Notes + (Past) CM0268 Module Notes

See also **next** **Lab Class**

CARDIFF UNIVERSITY PRIFYSGOL CAERDYD

If you like this sort of stuff

Related Year 3 Modules

- **CM3106** Multimedia
- **CM3102** Graphics, Visualisation and Computer Vision