Moving into the Frequency Domain

Frequency domains can be obtained through the transformation from one (**Time** or **Spatial**) domain to the other (**Frequency**) via

- **Discrete Cosine Transform (DCT)**— Heart of **JPEG** and **MPEG Video**, (alt.) MPEG Audio. (**New**)
- **Fourier Transform (FT)** — **MPEG Audio** (**See Tutorial 2 —Recall From CM0268 and**)

Note: We mention some image (and video) examples in this section with DCT (in particular) but also the FT is commonly applied to filter multimedia data.

Recap: What do frequencies mean in an image?

- Large values at **high** frequency components then the data is changing rapidly on a short distance scale.

  e.g. a page of text

- Large **low** frequency components then the large scale features of the picture are more important.

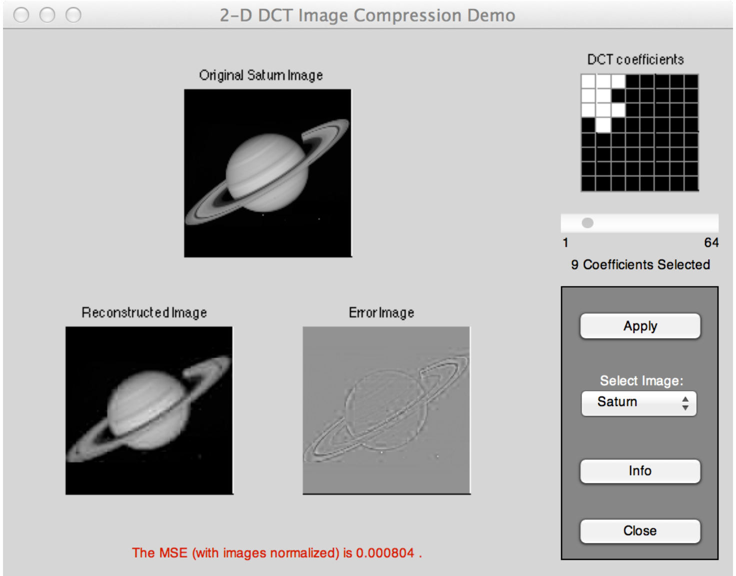
  e.g. a single fairly simple object which occupies most of the image.

The Road to Compression

How do we achieve compression?

- Low pass filter — ignore high frequency noise components
  - Only store lower frequency components
- High Pass Filter — Spot Gradual Changes
  - If changes to low Eye does not respond so ignore?

Low Pass Image Compression Example: dctdemo.m



MATLAB demo, dctdemo.m, (uses DCT (**see very soon**)) to

- Load an image
- **Low Pass Filter** in frequency (DCT) space
- *Tune* compression via a single slider value to select n coefficients
- **Inverse DCT**, **subtract input and filtered image** to see **compression artefacts**.

Recap: Fourier Transform

The tool which converts a spatial (real space) description of audio/image data into one in terms of its frequency components is called the **Fourier transform**

The new version is usually referred to as the **Fourier space description** of the data.
We then essentially process the data:

- *E.g.* for **filtering** basically this means attenuating or setting certain frequencies to zero

We then need to convert data back to real audio/imagery to use in our applications.

The corresponding **inverse** transformation which turns a Fourier space description back into a real space one is called the **inverse Fourier transform**.

The Discrete Cosine Transform (DCT)

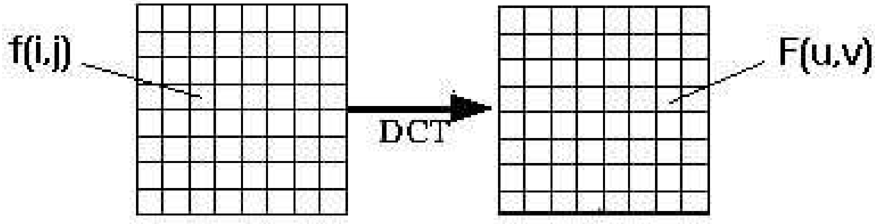
Relationship between DCT and FFT

DCT (Discrete Cosine Transform) is actually a *cut-down* version of the Fourier Transform or the Fast Fourier Transform (FFT):

- Only the **real** part of FFT
- Computationally simpler than FFT
- DCT — Effective for Multimedia Compression
- DCT **MUCH** more commonly used (than FFT) in Multimedia Image/Video Compression — **more later**
- Cheap MPEG Audio Variant — **more later**

Applying The DCT

- Similar to the discrete Fourier transform:
  - it transforms a signal or image from the spatial domain to the frequency domain
  - DCT can approximate lines well with fewer coefficients



- Helps separate the image into parts (or spectral sub-bands) of differing importance (with respect to the image's visual quality).

1D DCT

For N data items 1D DCT is defined by:

$$F(u) = \left(\frac{2}{N}\right)^{\frac{1}{2}} \sum_{i=0}^{N-1} \Lambda(u).cos\left[\frac{\pi.u}{2.N}(2i+1)\right] f(i)$$

and the corresponding **inverse** 1D DCT transform is simple $F^{-1}(u)$, i.e.:

$$\begin{aligned} f(i) &= F^{-1}(u) \\ &= \left(\frac{2}{N}\right)^{\frac{1}{2}} \sum_{u=0}^{N-1} \Lambda(u).cos\left[\frac{\pi.u}{2.N}(2i+1)\right] F(u) \end{aligned}$$

where

$$\Lambda(\xi) = \begin{cases} \frac{1}{\sqrt{2}} & \text{for} \xi = 0 \\ 1 & \text{otherwise} \end{cases}$$

2D DCT

For a 2D N by M image 2D DCT is defined :

$$
\begin{aligned}
F(u,v) &= \left(\frac{2}{N}\right)^{\frac{1}{2}}\left(\frac{2}{M}\right)^{\frac{1}{2}}\sum_{i=0}^{N-1}\sum_{j=0}^{M-1}\Lambda(u)\cdot\Lambda(v). \\
&\quad cos\left[\frac{\pi\cdot u}{2\cdot N}(2i+1)\right] cos\left[\frac{\pi\cdot v}{2\cdot M}(2j+1)\right]\cdot f(i,j)
\end{aligned}
$$

and the corresponding **inverse** 2D DCT transform is simple $F^{-1}(u,v)$, i.e.:

$$
\begin{aligned}
f(i,j) &= F^{-1}(u,v) \\
&= \left(\frac{2}{N}\right)^{\frac{1}{2}}\sum_{u=0}^{N-1}\sum_{v=0}^{M-1}\Lambda(u)\cdot\Lambda(v). \\
&\quad cos\left[\frac{\pi.u}{2.N}(2i+1)\right]\cdot cos\left[\frac{\pi\cdot v}{2\cdot M}(2j+1)\right]\cdot F(u,v)
\end{aligned}
$$

where

$$
\Lambda(\xi) = \begin{cases} \frac{1}{\sqrt{2}} & \text{for}\xi = 0 \\ 1 & \text{otherwise} \end{cases}
$$

Performing DCT Computations

The basic operation of the DCT is as follows:

- The input image is N by M;
- f(i,j) is the intensity of the pixel in row i and column j;
- F(u,v) is the DCT coefficient in row u_i and column v_j of the DCT matrix.
- For JPEG image (and MPEG video), the DCT input is usually an 8 by 8 (or 16 by 16) array of integers.
  This array contains each image window's respective colour band pixel levels;

Compression with DCT

- For most images, much of the signal energy lies at low frequencies;
  - These appear in the upper left corner of the DCT.
- Compression is achieved since the lower right values represent higher frequencies, and are often small
  - Small enough to be neglected with little visible distortion.

Computational Issues (1)

- Image is partitioned into 8 x 8 regions — The DCT input is an 8 x 8 array of integers. Why 8 x 8?
- An 8 point DCT would be:

$$
\begin{aligned}
F(u,v) &= \frac{1}{4}\sum_{i,j}\Lambda(u).\Lambda(v).cos\left[\frac{\pi.u}{16}(2i+1)\right]. \\
&\quad cos\left[\frac{\pi.v}{16}(2j+1)\right] f(i,j)
\end{aligned}
$$

where

$$
\Lambda(\xi) = \begin{cases} \frac{1}{\sqrt{2}} & \text{for} \xi = 0 \\ 1 & \text{otherwise} \end{cases}
$$

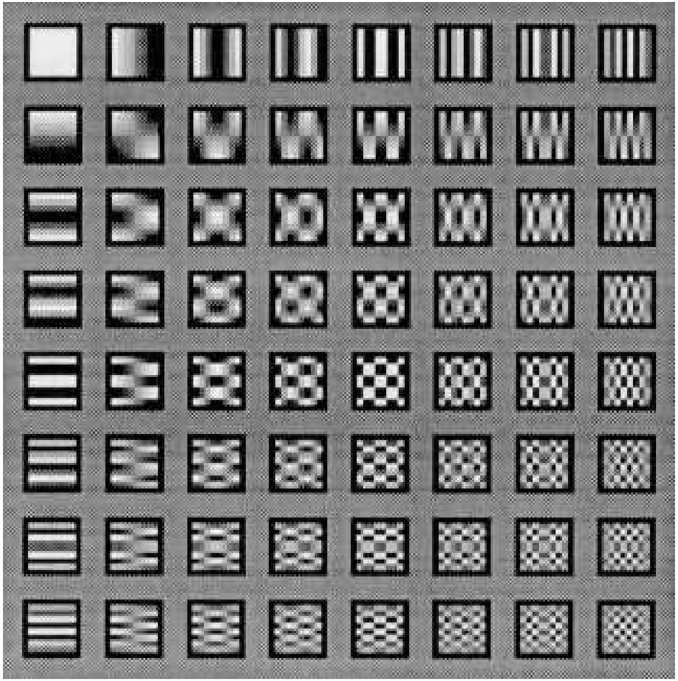
- The output array of DCT coefficients contains integers; these can range from -1024 to 1023.

Back
Close

Computational Issues (2)

- Computationally easier to implement and more efficient to regard the DCT as a set of **basis functions**
  - Given a known input array size (8 x 8) can be precomputed and stored.
  - Computing values for a convolution mask (8 x 8 window) that get applied
    * Sum values x pixel the window overlap with image apply window across all rows/columns of image
  - The values as simply calculated from DCT formula.

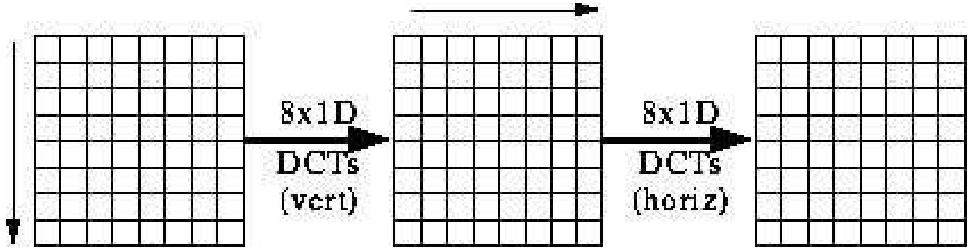
Computational Issues (3)
Visualisation of DCT basis functions



See MATLAB demo, dctbasis.m, to see how to produce these bases.

Computational Issues (4)

- Factoring reduces problem to a series of 1D DCTs (No need to apply 2D form directly):
  - apply 1D DCT (Vertically) to Columns
  - apply 1D DCT (Horizontally) to resultant Vertical DCT above.
  - or alternatively Horizontal to Vertical.



Computational Issues (5)

- The equations are given by:

$$
\begin{aligned}
G(i,v) &= \frac{1}{2}\sum_i \Lambda(v)\cdot cos\left[\frac{\pi\cdot v}{16}(2j+1)\right] f(i,j) \\
F(u,v) &= \frac{1}{2}\sum_i \Lambda(u)\cdot cos\left[\frac{\pi\cdot u}{16}(2i+1)\right] G(i,v)
\end{aligned}
$$

- Most software implementations use fixed point arithmetic. Some fast implementations approximate coefficients so all multiplies are shifts and adds.

Filtering in the Frequency Domain: Some more examples

FT and DCT methods pretty similar:

- DCT has less data overheads — no complex array part
- FT captures more frequency 'fidelity' (e.g. Phase).

Low Pass Filter

Example: Frequencies above the Nyquist Limit, Noise:

- The idea with noise smoothing 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.

Recap: Frequency Space Smoothing Methods

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 image when it is considered in Fourier space.

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

(Low-pass) Filtering in the Fourier Space

We thus create a new version of the image in Fourier space by computing

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

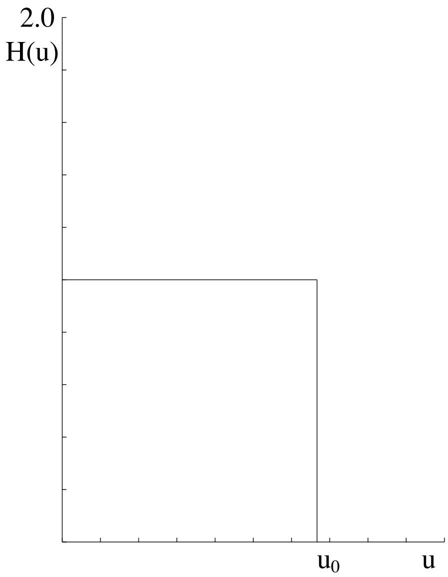
where:

- $F(u, v)$ 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**

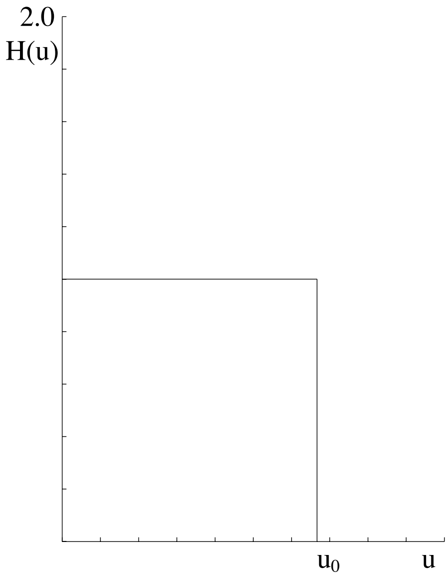
Note: Discrete Cosine Transform approach identical, sub. FT with DCT

Ideal Low-Pass Filter

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



Ideal Low-Pass Filter (Cont.)



This is a top hat 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 thrown away, and all information below u_0 is kept.
- A **very simple** computational process.

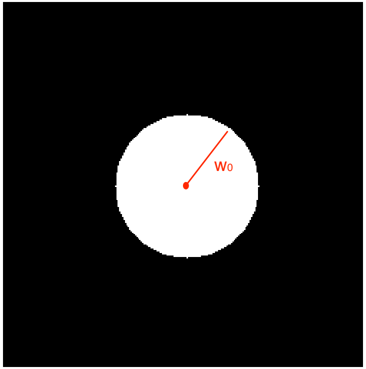
Ideal 2D Low-Pass Filter

The two dimensional analogue of this is the function

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

where w_0 is now the cut-off frequency.

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



Not So Ideal Low-Pass Filter?

The problem with this filter is that as well as the noise:

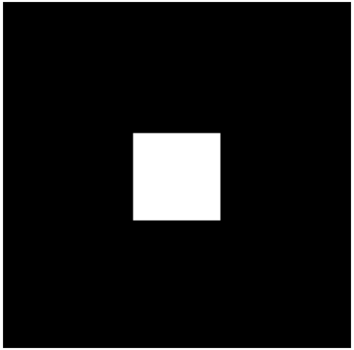
- In audio: plenty of other high frequency content
- In Images: edges (places of rapid transition from light to dark) also significantly contribute to the high frequency components.

Thus an ideal low-pass filter will tend to *blur* the data:

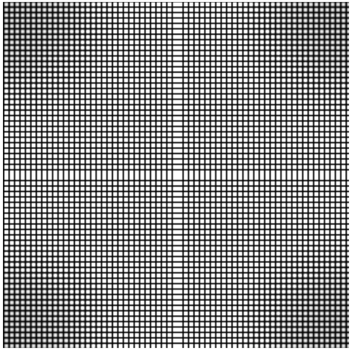
- High audio frequencies become muffled
- Edges in images become blurred.

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

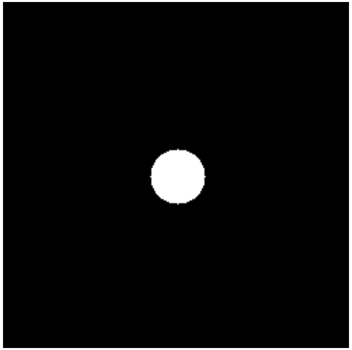
Ideal Low Pass Filter Example 1



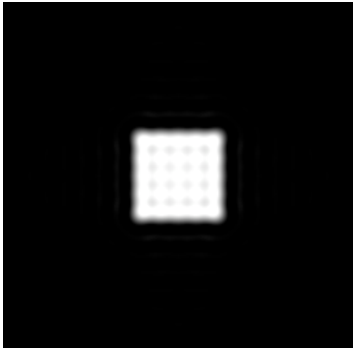
(a) Input Image



(b) Image Spectra



(c) Ideal Low Pass Filter



(d) Filtered Image

Ideal Low-Pass Filter Example 1 MATLAB Code

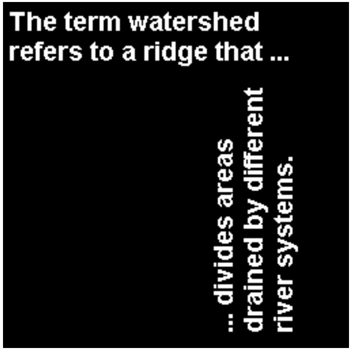
low pass.m:

```
% 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)));
```

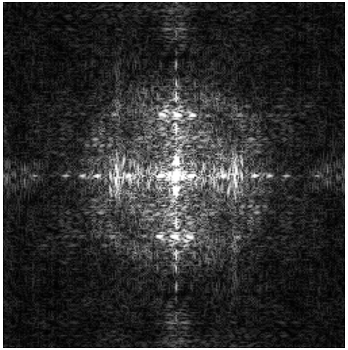
Ideal Low-Pass Filter Example 1 MATLAB Code (Cont.)

```
%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);
```

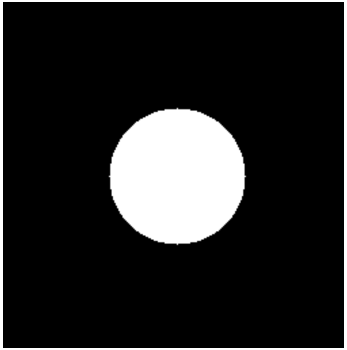
Ideal Low-Pass Filter Example 2



(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);
```

Ideal Low-Pass Filter Example 2 MATLAB Code (Cont.)

```
%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

Another filter sometimes used is the *Butterworth low pass filter*.

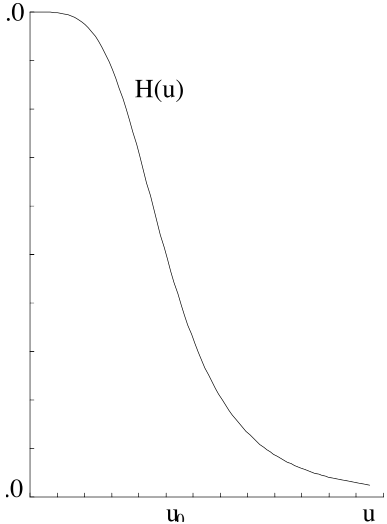
In the 2D case, $H(u,v)$ takes the form

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

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

Low-Pass Butterworth Filter (Cont.)

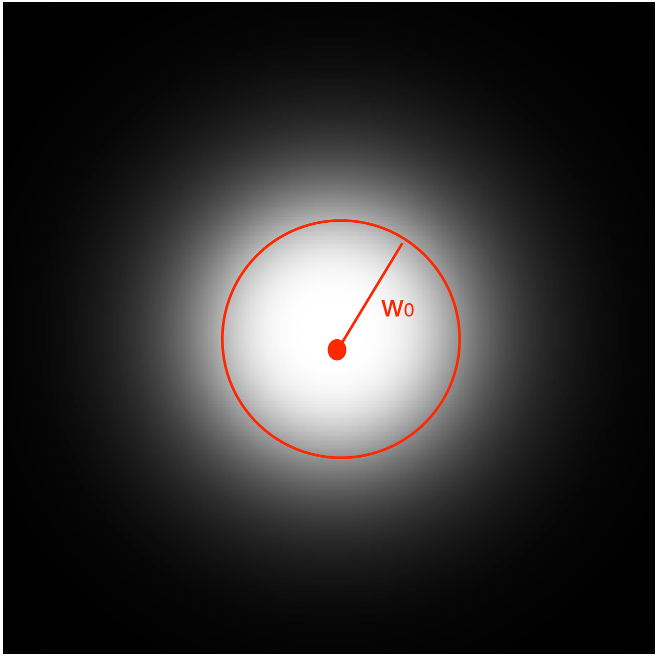
This keeps some of the high frequency information, as illustrated by the second order one dimensional Butterworth filter:



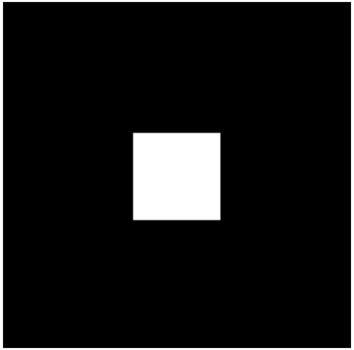
Consequently reduces the blurring.

Low-Pass Butterworth Filter (Cont.)

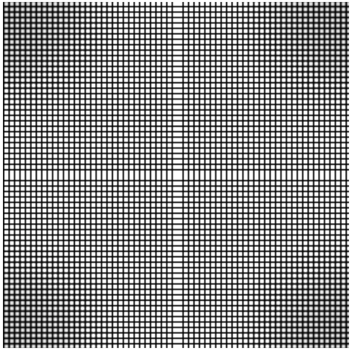
The 2D second order Butterworth filter looks like this:



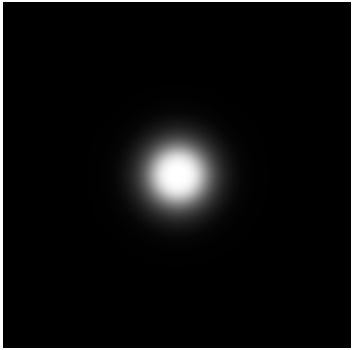
Butterworth Low Pass Filter Example 1



(a) Input Image



(b) Image Spectra



(c) Butterworth Low-Pass Filter

(d) Filtered Image

Butterworth Low-Pass Filter Example 1 (Cont.)

Comparison of Ideal and Butterworth Low Pass Filter:

Ideal Low-Pass

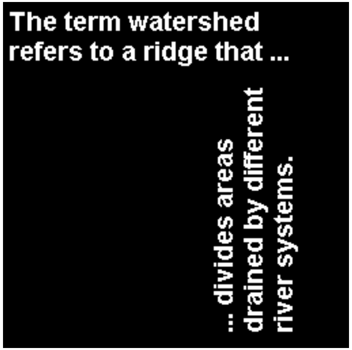
Butterworth Low Pass

Butterworth Low-Pass Filter Example 1 MATLAB Code

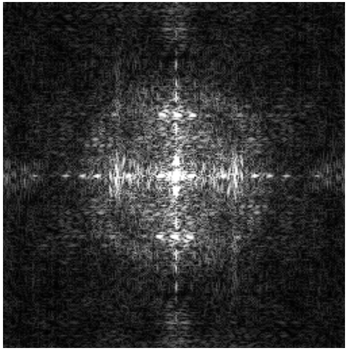
butterworth.m:

```
% 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
```

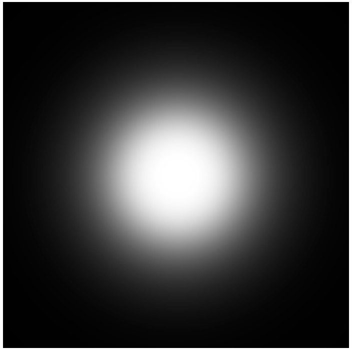
Butterworth Low-Pass Butterworth Filter Example 2



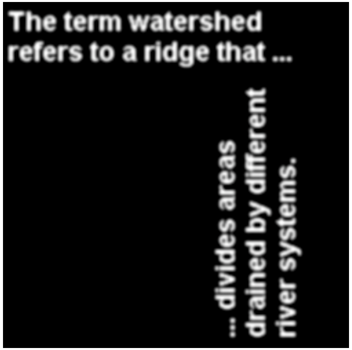
(a) Input Image



(b) Image Spectra



(c) Butterworth Low-Pass Filter

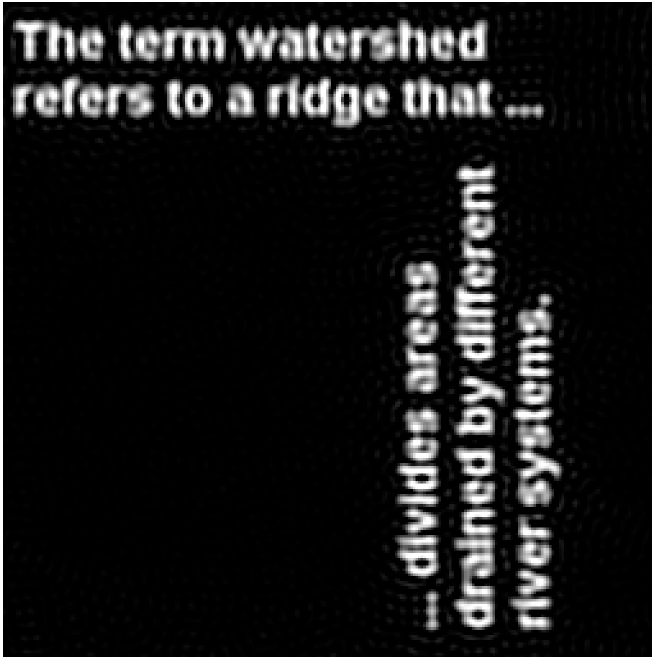


(d) Filtered Image

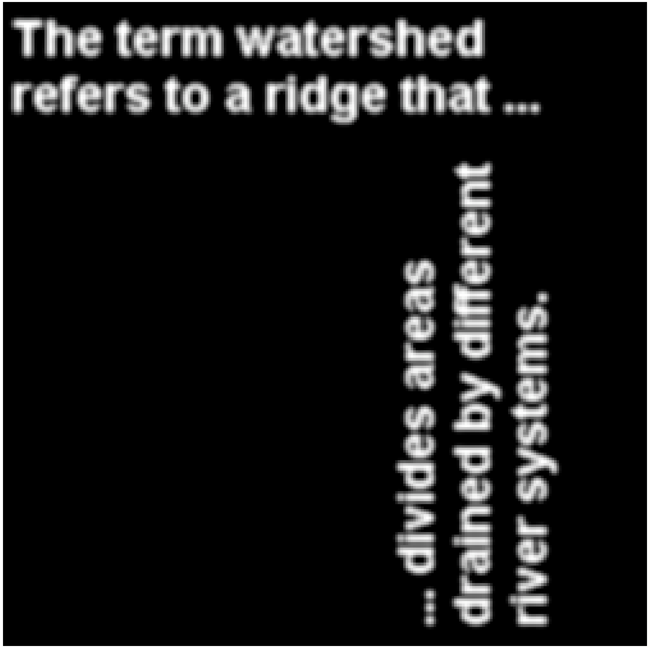
Back
Close

Butterworth Low-Pass Filter Example 2 (Cont.)

Comparison of Ideal and Butterworth Low-Pass Filter:



Ideal Low Pass



Butterworth Low Pass

Butterworth Low Pass Filter Example 2 MATLAB Code

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
```

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 are a combination of the above

Convolution

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

- In fact the above Fourier filtering is applying convolutions of low pass filter where the equations are Fourier Transforms of real space equivalents.
- deblurring — high pass filtering
- reverb — see CM0268.

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

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

1D Convolution Example

For example, let us take two top hat functions of the type described earlier.

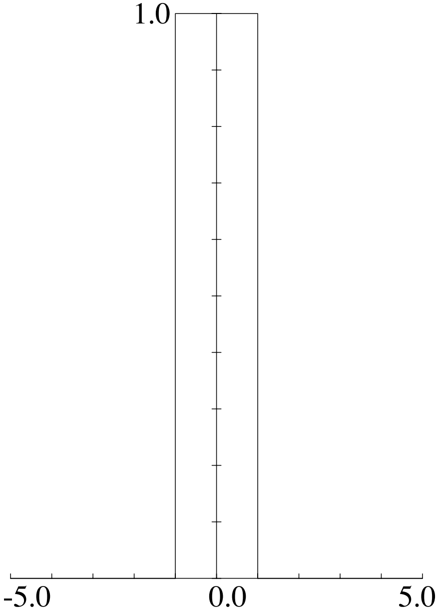
Let $f(\alpha)$ be the top hat function shown:

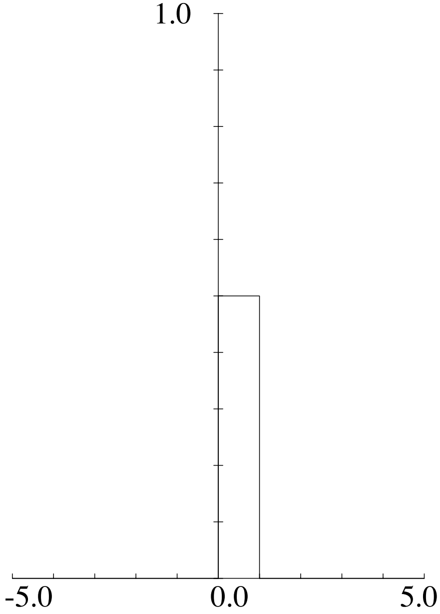
$$f(\alpha) = \begin{cases} 1 & \text{if } |\alpha| \leq 1 \\ 0 & \text{otherwise,} \end{cases}$$

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

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

1D Convolution Example (Cont.)

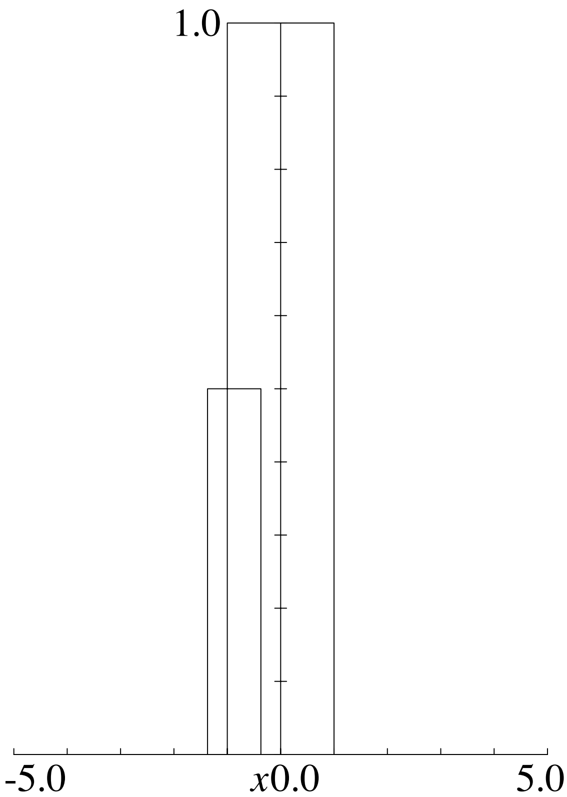




$$f(\alpha) = \begin{cases} 1 & \text{if } |\alpha| \leq 1 \\ 0 & \text{otherwise,} \end{cases} \qquad g(\alpha) = \begin{cases} 1/2 & \text{if } 0 \leq \alpha \leq 1 \\ 0 & \text{otherwise.} \end{cases}$$

1D Convolution Example (Cont.)

- $g(-\alpha)$ is the reflection of this function in the vertical 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

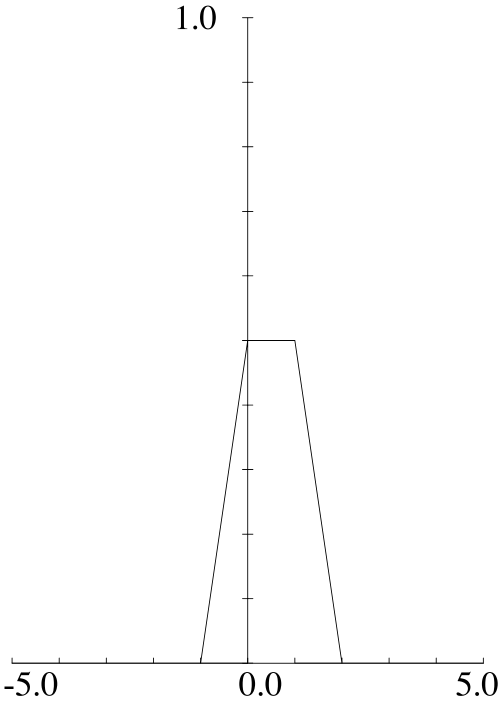


1D Convolution Example (cont.)

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 from 0 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 (cont.)

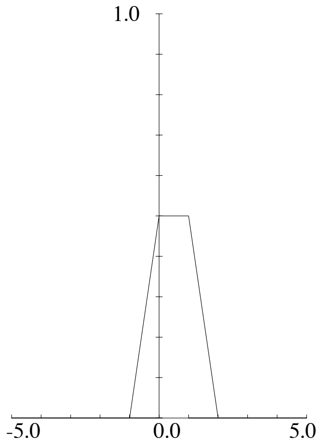


Result of $f(x) * g(x)$

1D Convolution Example (cont.)

Mathematically the convolution is expressed by:

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



Fourier Transforms and Convolutions

One major reason that Fourier transforms are so important in 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

E.g.:

- To apply some reverb to an audio signal, **example later**
- To compensate for a less than ideal image capture system:

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,
- Multiply by the effect to apply effect to audio data
- To remove/compensate for effect: Divide by the effect to obtain the Fourier transform of the ideal image.
- Inverse Fourier transform to recover the new audio/ideal image.

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