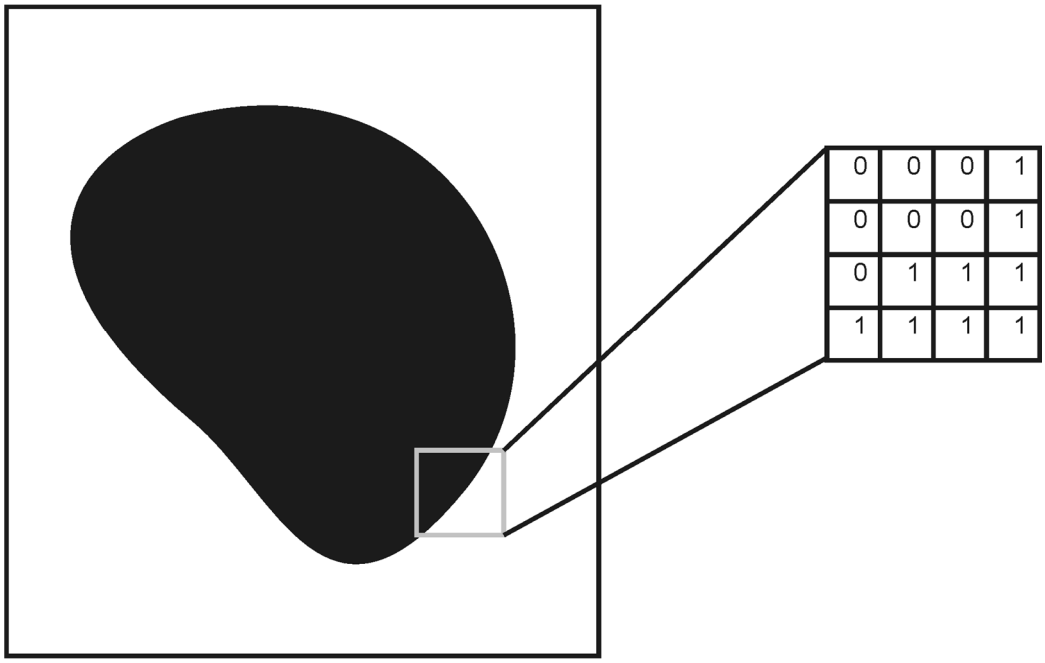
4 Edge detection

4.1 Edge Detection

Edge detection is applied on an image to obtain the edges in that image. A typical example of an application of edge detection is the measurement of the thickness of arteries in a coronary angiogram (see figure). Edge detection is also of importance in computer vision, for example to isolate specific objects from a scene.

Edge detection can be done in several ways, we shall discuss three possibilities. Points that are part of an edge are characterized by a sudden jump in grey value. You can see this clearly in the picture below:



4.1.1 Edge detection with the use of Sobel and Prewitt operators

An obvious way to detect sudden grey value changes is the gradient. The gradient has two components in each point: $\left(\frac{\partial f}{\partial x}, \frac{\partial f}{\partial y}\right)$. The gradient magnitude is a good indication of the existence of an edge. The gradient magnitude is given by $\sqrt{\left(\frac{\partial f}{\partial x}\right)^2 + \left(\frac{\partial f}{\partial y}\right)^2}$. Another good measure for the presence of an edge is $\left|\frac{\partial f}{\partial x}\right| + \left|\frac{\partial f}{\partial y}\right|$. This operation is less complex and is therefore often used in hardware implementations. $\max\left(\left|\frac{\partial f}{\partial x}\right|, \left|\frac{\partial f}{\partial y}\right|\right)$ can also be used. The only problem now is the discretization of the gradient. Two frequently used operators are the Prewitt and the Sobel operators:

$$\begin{pmatrix} -1 & 0 & 1 \\ -1 & 0 & 1 \\ -1 & 0 & 1 \end{pmatrix}$$

The Prewitt operator for vertical edges

$$\begin{pmatrix} -1 & -1 & -1 \\ 0 & 0 & 0 \\ 1 & 1 & 1 \end{pmatrix}$$

Prewitt operator for horizontal edges

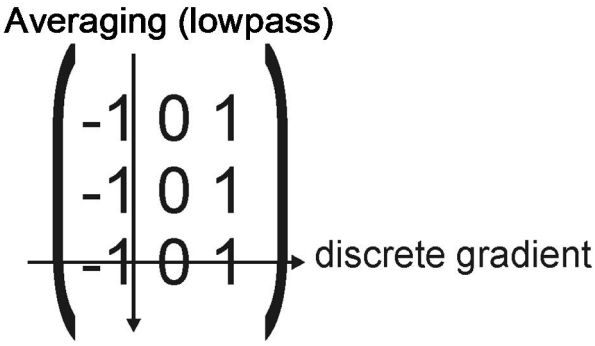
$$\begin{pmatrix} -1 & 0 & 1 \\ -2 & 0 & 2 \\ -1 & 0 & 1 \end{pmatrix}$$

Sobel operator for vertical edges

$$\begin{pmatrix} -1 & -2 & -1 \\ 0 & 0 & 0 \\ 1 & 2 & 1 \end{pmatrix}$$

Sobel operator for horizontal edges

After the application of these operators a tresholding has to be applied on the resulting image to get the strongest (real) edges. The choice of the threshold is heuristic. The Prewitt and Sobel operators are a combination of an averaging parallel to the edge followed by the discrete gradient of the result:



4.1.2 Exercise 16

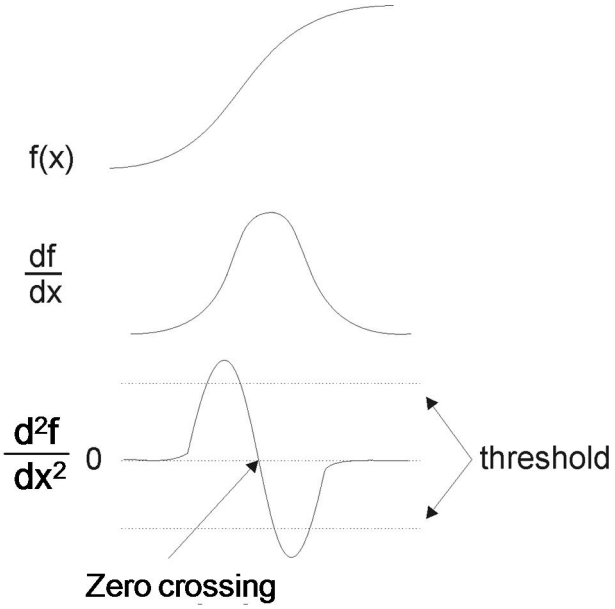
Edge detection. Read the greyscale image *gertrude.tif* and convert it to double format. Use the `fspecial` function to construct the horizontal Sobel operator (the sign in Matlab differs from the commonly accepted definition. Correct this. Calculate the vertical Sobel operator by use of the ‘ operator (transposition of a matrix). What is the conceptual difference with the Prewitt operator? Apply the horizontal and the vertical Sobel operator and view the result. Combine the results into an edge image with the use of $\sqrt{\left(\frac{\partial f}{\partial x}\right)^2+\left(\frac{\partial f}{\partial y}\right)^2}$. View the edge image. What do you notice (left and right cheek of Gertrude)? Why is this? Use `improfile` to view the progress of the grey values on the left and the right cheek in the original. Use `im2bw` to apply a threshold. Choose the threshold equal to 0.3. View the binary edge image. Is the entire operation linear?

4.1.3 Exercise 17

Edge detection. Edge detection in the presence of noise. Add gaussian noise to *gertrude.tif* (don’t forget the conversion to double) with the use of `imnoise` (mean 0, sigma=0.05). Then use the regular Sobel operator and afterwards the Sobel operator combined with averaging. Compare the results in the edge image and the binary edge image (use threshold 0.3 and 0.4).

4.1.4 Edge detection through zero crossings of the laplacian of the gaussian.

Edge detection. Next to the Sobel and Prewitt operators which are based on the gradient, there is another way to detect edges, namely by detecting zero crossings of the laplacian of the Gaussian of the image. We can realize that the laplacian can be used for edge detection by looking at the following illustration:



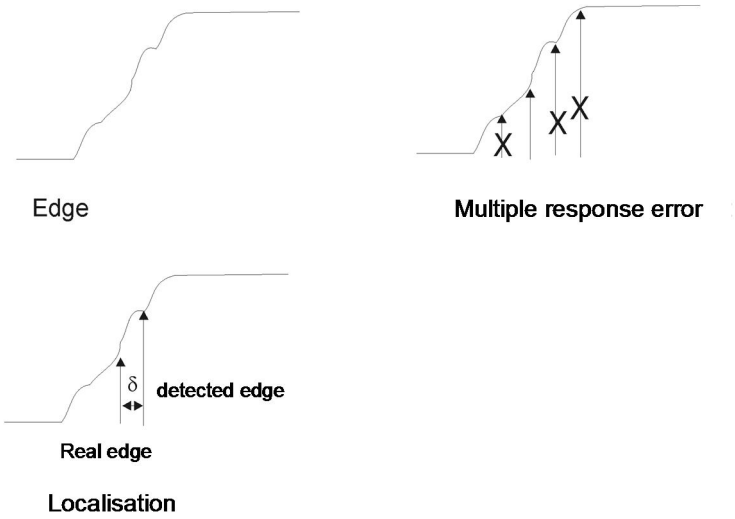
Because we are dealing with the second derivative, this operator is much more sensitive to noise than the gradient operators. That's why the laplacian of the Gaussian is used. Applying this operator delivers a result that is equivalent with the use of a Gaussian filter (lowpass to suppress noise) followed by the application of the laplacian. After filtering, we have to determine the zero crossings of the result. In two dimensions there is a zero crossing if there is at least a zero crossing in one direction. The filter has two different parameters that have an influence on its operation, σ which defines the width of the Gaussian (width on half the height, see statistics) and N, the dimensions of the filter mask. Standard (MATLAB) N equals 5 so that the filter has 5x5 coefficients.

4.1.5 Exercise 18

Edge detection. The "laplacian of Gaussian" operator. Build a "laplacian of Gaussian" operator with the use of `fspecial (option 'log')`. Take a filter of 11x11 coefficients and choose sigma equal to 0.7. View the filter (use `mesh`,`surf`,`imagesc`). Apply the filter on the greyscale image *gertrude.tif* (first convert it to double). Examine the result (use `imagesc` (see help)). Search the zero crossings of the result by using `zerocros.m`. We can see that we have false edges (spurious edges and phantom edges) in quasi uniform areas (jaw). How is this possible? Repeat the procedure with sigma 1.4, 2.1, 2.8 and compare. What is the meaning of sigma? What is the effect of sigma on the false edges? What is the advantage of this method in comparison with the previous?

4.1.6 The Canny edge detector.

The canny edge detector is designed as the optimal edge detector under three conditions: maximum SNR of the result, optimal localization and minimal number of multiple response errors (the same edge is detected several times). These notions are illustrated in the next figure.



The Canny edge detector can, with only slight loss in optimality, be approximated with the first derivative of a Gaussian. This is often used for practical implementation. Next to defining the optimal edge detection filter, canny edge detection features several post-processing steps that drastically improve edge detection performance. The complete canny edge detection procedure can be summarized in de following steps:
1. Filter the image with a Gaussian filter
2. Define the gradient magnitude and gradient direction of the result
3. Non-maxima suppression: its goal is to get edges that are maximum one pixel wide. Therefore we search for each pixel in the direction of the gradient if the pixel is a local maximum. If so, the value is kept and otherwise it is put on zero
4. Thresholding with hysteresis: We go through each pixel in the gradient magnitude image. If the magnitude is larger than T1 we continue perpendicular to the gradient direction. Each pixel we come across is saved into the edge map, as long as the magnitude is larger than T2.

4.1.7 Exercise 19

The Canny edge detector. Apply the Canny edge detector on *gertrude.tif* (`edge` command). Also use edge to try out the other edge detection techniques on *gertrude.tif* and compare the results. Which method works best? If we look at the results of the edge detection with the Sobel filter, it is striking that the edges are only one pixel wide. This in contradiction to what we got earlier. How could this be?

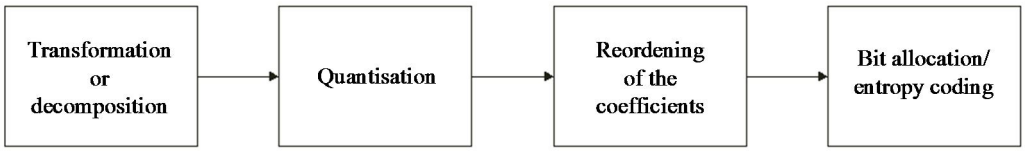
5 Compression

5.1 In general

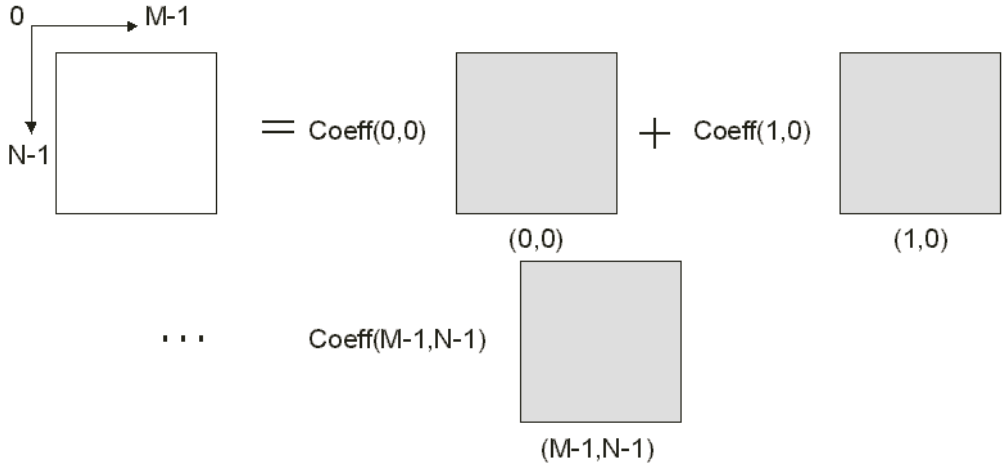
Image compression is an operation which aims to minimize the amount of bits needed to represent an image digitally. The advantages of a more compact image representation are obvious: more efficient transmission and storage. Images can be compressed by exploiting the correlation between neighbouring pixels and by using some properties of the human visual system. Image compression can be divided into two domains: compression with quality loss (*lossy*) and without any loss of quality (*lossless*). We will see examples of the lossy case.

5.2 Lossy compression

Lossy compression means that some distortion in the reconstruction of the compressed image is tolerated to be able to attain higher compression ratios. The most important group of lossy compression algorithms are based on an image transformation followed by quantization and entropy coding. The next figure shows that typical structure:



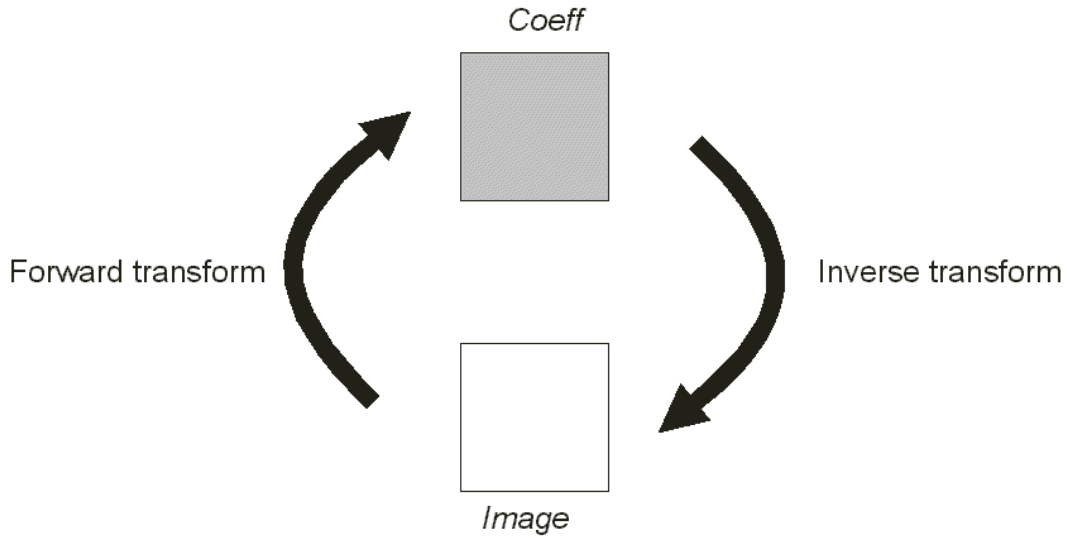
In a first step the image is transformed to another representation with the help of an image transformation.



The main idea behind an image transformation is that we decompose the image, with dimensions M x N, in a linear combination of M x N basis images and that the coefficients of that linear compression form once again an image that we shall call the transformed image. Formulated mathematically this becomes:

$$image(x,y) = \sum_{i=0}^{M-1}\sum_{j=0}^{N-1} coeff(i,j)\cdot b_{ij}(x,y) = \sum_{i=0}^{MN-1} coeff(i)\cdot b_i(x,y)$$

image(x,y) is the pixel on position (x,y) in the original image and $b_{ij}(x,y)$ the pixel on position (x,y) in basic image b_{ij}. The *coeff(i,j)* form a new image with the same dimensions as the original image. In this case there is no loss of information and we can retrieve the original image by applying the inverse transformation on the original image.



An image transformation is used for two reasons as a basis for a compression algorithm. First of all, the image may be represented with less than MxN coefficients in the transformed domain without having serious visual consequences. A second reason is that the coefficients of the transformation can have a probability distribution that is strongly non uniform, which makes efficient entropy coding impossible.
The second step is the quantization step. This is the step that introduces loss. The most important part of quantization is the representation of the coefficients (continuous variables) of the transformation by a final set of values that come as close as possible to the coefficients. Formally, quantization is defines as follows: X is a real, continuous stochastic variable. A quantizer is an operator that transforms X to $Y=Q(X)$ so that:
$a_{i-1} < X \leq a_i \Leftrightarrow Y = \gamma_i$
The coefficients a_i form the quantisation thresholds, the γ_i-values are called the reconstruction levels. The quantization is uniform if the quantization tresholds are on equal distance from each other:
$\forall i : a_i - a_{i-1} = \Delta$
The quantization module in the coder does more than convert continuous coefficients to discrete symbols. During the quantization step coefficients that aren't significant for the visual quality of the image are eliminated.
The next step in the scheme is the reordering of the quantized coefficients so they would be more efficient to encode (e.g.: a longer series of the same symbol with run-length coding), or to support a certain functionality, like spatial scalability.
The last step is bit allocation. Typical algorithms for bit allocation are arithmetic coding, Huffman encoding and run-length encoding. This step is normally lossless.

5.3 Performance of image compression techniques

To compare the performance of lossless compression codecs they are tested on a number of images where the *compression ratio* or the *bit-rate* is used as a value meter. The compression ratio (CR) is the proportion between the number of bits in the original image file and the number of bits in the compressed bitstream. The *bit-rate* is a value that states how many bits are assigned on average in the compressed bitstream per pixel in the image.

$$CR = \frac{number\ \ of\ bits\ \ in\ \ original\ \ image}{number\ of\ \ bits\ \ in\ \ compressed\ \ bitstream}$$

$$bit - rate = \frac{number\ \ of\ bits\ \ in\ \ compressed\ \ bitstream}{number\ \ of\ pixels}$$

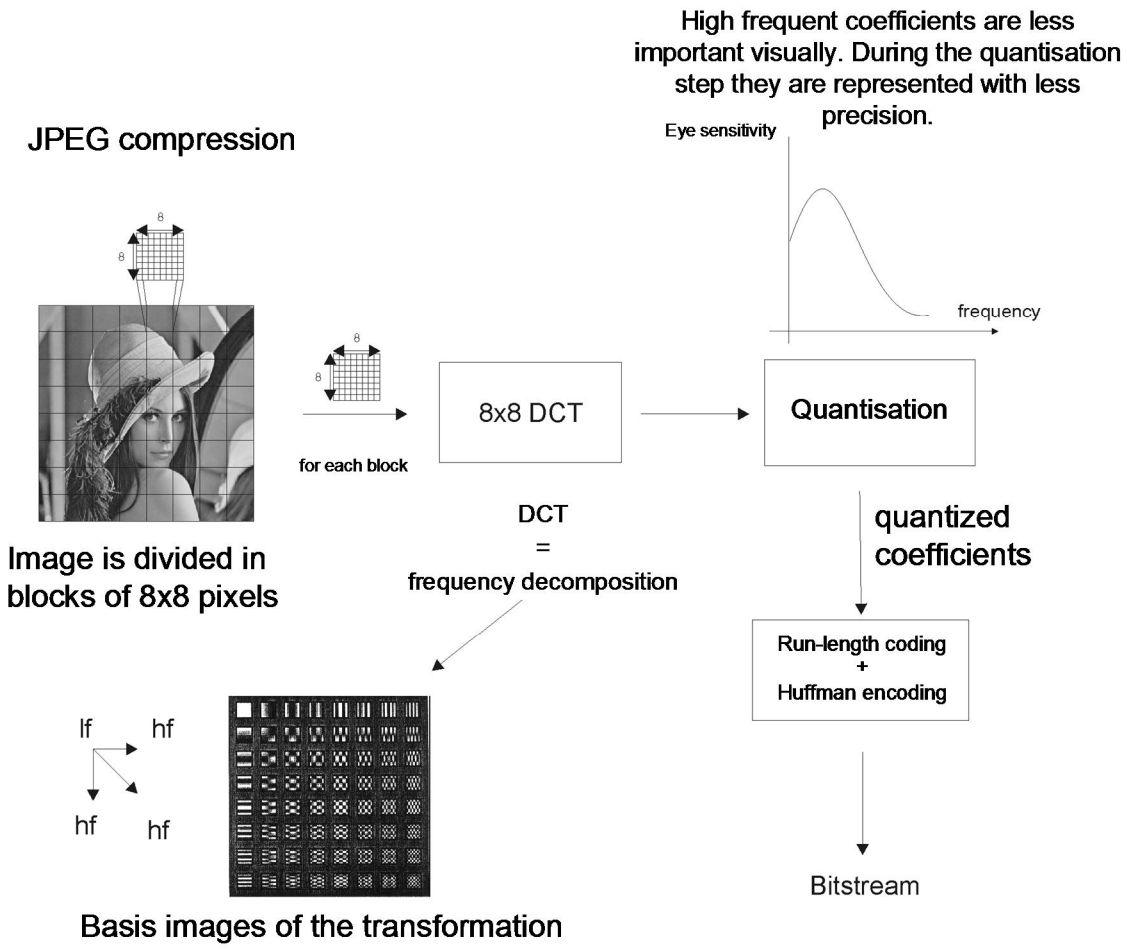
Lossy compression codecs are compared by looking at the quality of the reconstructed image. To objectively quantify this quality the PSNR (*peak signal-to-noise ratio*) is used. This is defined as:

$$PSNR = 10 \cdot \log_{10} \frac{MAX^2}{\sigma_e^2}$$

$$\sigma_e^2 = \frac{1}{N} \cdot \sum_{i=0}^{N-1} (x_i - x'_i)^2$$

MAX is the maximum grey value that can be achieved in the image (255 for 8 bit, 32767 for 16 bit), σ^2_e is the variance of the error between the original and reconstructed image, x_i de ith grey value in the original image, x'_i the ith grey value in the reconstructed image and N the number of pixels that are considered. It boils down to the comparison between the original and the reconstructed image.

5.4 DCT-based compression



DCT of a block of NxN pixels:
Forward transformation:

$$DCT(i,j) = \sum_{x=0}^{N-1}\sum_{y=0}^{N-1} image(x,y)\cdot c(i)\cdot \cos\left[\frac{(2\cdot x+1)\cdot i\cdot \pi}{2\cdot N}\right]\cdot c(j)\cdot \cos\left[\frac{(2\cdot y+1)\cdot j\cdot \pi}{2\cdot N}\right]$$

$$i = 0 \rightarrow c(i) = \sqrt{\frac{1}{N}}$$

$$i > 0 \rightarrow c(i) = \sqrt{\frac{2}{N}}$$

Inverse transformation:

$$image(x,y) = \sum_{i=0}^{N-1}\sum_{j=0}^{N-1} DCT(i,j)\cdot c(i)\cdot \cos\left[\frac{(2\cdot x+1)\cdot i\cdot \pi}{2\cdot N}\right]\cdot c(j)\cdot \cos\left[\frac{(2\cdot y+1)\cdot j\cdot \pi}{2\cdot N}\right]$$

$$i = 0 \rightarrow c(i) = \sqrt{\frac{1}{N}}$$

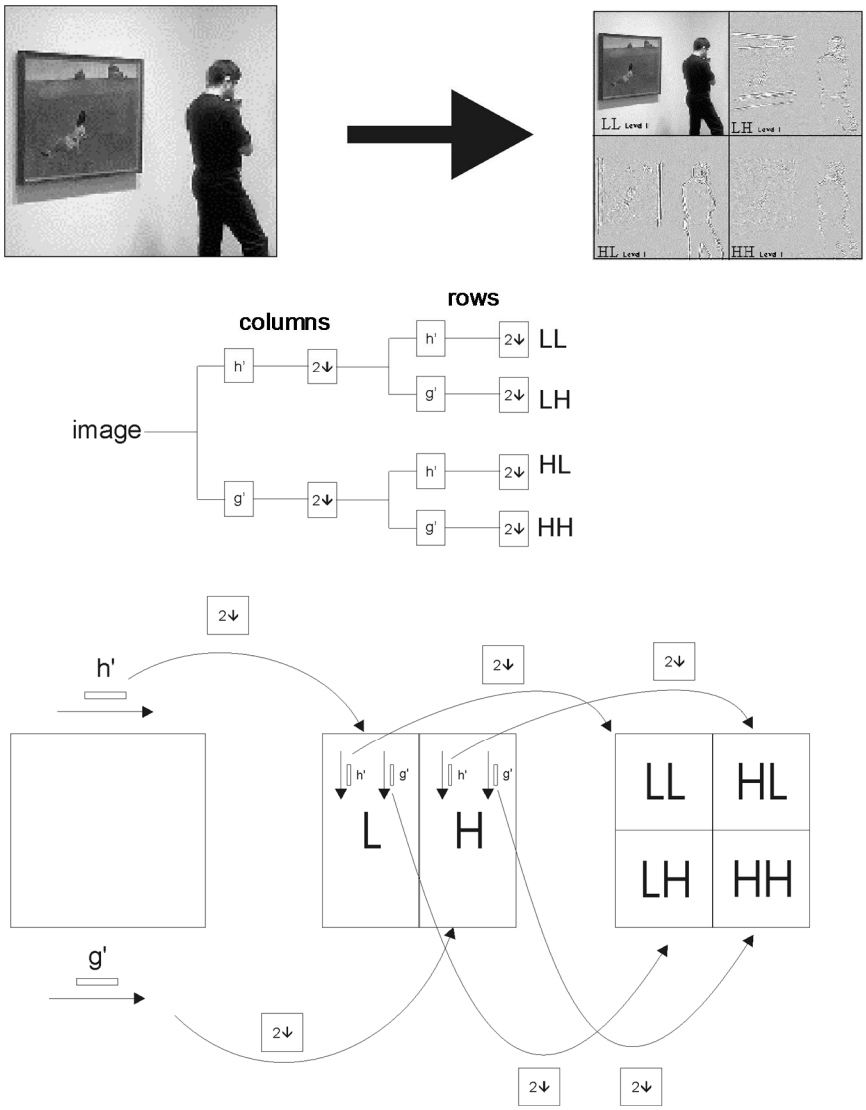
$$i > 0 \rightarrow c(i) = \sqrt{\frac{2}{N}}$$

The DCT is a frequency decomposition. In the transformed block stand the received coefficients ordered from the DC coefficient in the left upper corner to the high frequency coefficients when we advance towards the lower right part. As the human eye is less sensitive to higher frequencies, the high frequent components can be coded with a lesser precision or even be omitted without having a lot of effect on the image quality.

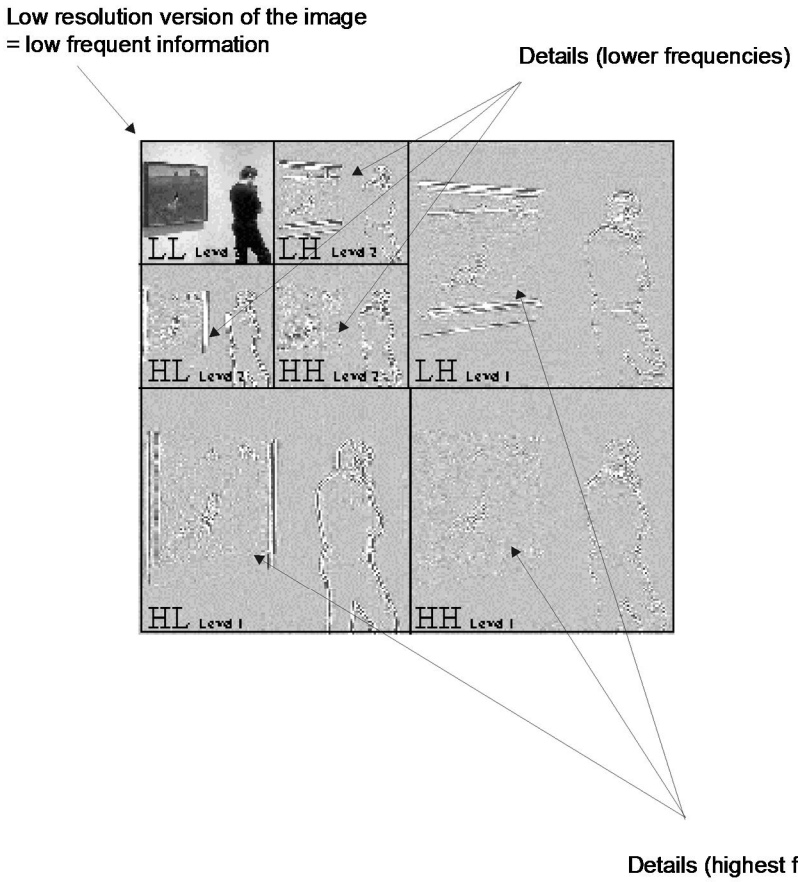
5.4.1 Exercise 20

Load the greyscale image *lenagray.tif* and convert it to double format. Look at the image. Split it up in blocks of 8 by 8 pixels and execute the DCT on each block (function: `DCT2`). Block based operations in Matlab are best executed with `blkproc`. Check what the effect is when you set a part of the high frequency coefficients to zero. Do this in four steps where each time you set more coefficients to zero. Start by setting only the highest frequency coefficients to zero and end up with keeping only the DC components. Use masking for this operation (multiplication with a matrix with only ones on the positions of the coefficients you want to keep and zeros on the other positions). Each time execute the IDCT (function `IDCT2`) and compare the image with the original. Write a program to compute the PSNR and execute it for every case. What can you conclude about the influence of setting the high frequency components to zero? What kind of distortions occur if you retain only a few coefficients?

5.5 Wavelet transformation and compression



The picture above shows the execution of one level of the wavelet transformation. The operation can be summarised as the execution of a filtering on rows and columns of the image followed by a dyadic subsampling. Just as the DCT the wavelet transformation is a frequency decomposition: the detail subbands that come from the first decomposition step contain the information of the image with the highest frequency content.
The information in the detail subbands on the second level contain less high frequent information, etc. The LL subbands (lowpass image) consists of the low frequent information. Further, we notice that each detail subband has highfrequent details associated to a certain direction: horizontal details, vertical details and diagonal details. Research has shown that the diagonal details have a smaller impact on the outlook of the image.



We can also see that the detail subbands of natural images don't contain a lot of information. By removing the coefficients out of the highfrequent subbands or quantifying them more coarsely, efficient lossy compression can be achieved. Lossless compression becomes possible by exploiting the inter- or intra-subband correlation between the coefficients by using a form of entropy coding.

5.5.1 Exercise 21

Load the image *lenagray.tif* and convert it to double format. Execute a wavelet decomposition with 3 levels. Use the function `wavedec2` with filtertype `bior4.4`. Derive all the subbands from C and S (see help wavedec2) with the use of the functions `appcoef2` and `detcoef2`. Display the subbands with *myshow.m* (take 64 as second parameters for the LL subband and 128 for all other subbands). Can you detect the different orientations of the details? The orientations depend on the sequence in which the filtering is performed. Explain. See what happens when you put all the coefficients from the HH1 subband to zero (in the C matrix) and execute the wavelet reconstruction (command `waverec2`). Compare the result with the original. Is there a lot of difference? Repeat the previous steps and successively put all coefficients from

1) HH1 LH1 HL1
2) HH1 LH1 HL1 HH2
3) HH1 LH1 HL1 HH2 HL2 LH2
4) HH1 LH1 HL1 HH2 HL2 LH2 HH3
5) HH1 LH1 HL1 HH2 HL2 LH2 HH3 LH3 HL3

to zero and check the results. What are the kind of errors you see in the last cases between the original and the reconstructed image (2)? Calculate the PSNR for each case and compare.