2.6 Image quality and the signal-to-noise ratio

In Chapter 1, we have already encountered some examples of imaging artifacts which stem from the particular properties of imaging systems. These artifacts can, of course, reduce diagnostic information content of images and are to be avoided. However, another source of reduced image quality is noise. The term *noise* describes all types of stochastic signals which are not related to image content. In x-ray imaging, noise is usually introduced by a lack of dose; in photography, images tend to get noisy if the ambient lighting is insufficient. The reason in both cases is simple – since image noise is a stochastic signal, its amplitude is independent of the usable signal. If the imaging signal itself is of small amplitude, the noise becomes more visible. A measure to quantify the noisiness of images is the *signal-to-noise-ratio* (SNR). One possible definition is given as:

$$\begin{aligned} \mathrm{SNR} &= \frac{\bar{\rho}}{\sigma\left(\rho_{\text{Uniform Area}}\right)} \qquad (2.5) \\ \bar{\rho} &\ldots \text{ average pixel gray value} \\ \sigma\left(\rho_{\text{Uniform Area}}\right) &\ldots \text{ standard deviation in a signal-free area of the image} \end{aligned}$$

From Equation 2.5, it is obvious why an underexposed image becomes grainy. The energy impacted (in terms of visible light or dose) is proportional to the gray value ρ, whereas the noise remains the same. The reason lies in the fact that the image noise, which can have numerous origins, is not necessarily connected to the original signal. While SNR is an important measure of detector efficiency and image quality, it does not give an idea of the ability of an imaging system to resolve fine detail, and it does not necessarily give a figure of the information content of an image. This is characterized by two more important measures, the point spread function (PSF) and the modulation transfer function (MTF) , which will be introduced in Chapter 4. Lesson 2.7.4 gives an idea how the SNR can be computed from an image in dependence of the dose applied. However, since the average signal is not always a useful measure, on can also define the *contrast-to-noise-ratio* (CNR). Here, the average gray value is replaced by the absolute value of the difference between the maximal and minimal signal:

$$\begin{aligned} \mathrm{CNR} &= \frac{\left|\rho_{\max} - \rho_{\min}\right|}{\sigma\left(\rho_{\text{Uniform Area}}\right)} \qquad (2.6) \\ \rho_{\max},\ \rho_{\min} &= \text{maximal and minimal gray value} \\ \sigma\left(\rho_{\text{Uniform Area}}\right) &= \text{standard deviation in a signal-free area of the image} \end{aligned}$$

2.7 Practical lessons

2.7.1 Opening and decoding a single DICOM file in MATLAB

In this first lesson, a single DICOM file will be opened and displayed using MATLAB or Octave; for this purpose, we provide a single slice of a T_2 weighted MR of

a pig. The slice shows the remarkable small brain and the huge temporomandibular joint of this species. The file is named `PIG_MR`. The image is uncompressed, has a matrix of $512 * 512$ pixels, and a depth of 16 bit. The machine used to acquire the volume was a 3T Philips Achieva MR. The script for this lesson is called `OpenDICOM_2.m` and can be found in the `LessonData` folder.

The single steps to be taken are as follows:

- Determine the length of the header.
- Open the image file in MATLAB using the `fopen(...)` function, and skip the header using `fseek`.
- Assign a matrix of dimension $512 * 512$, and fill it with the values from the image file using `fread`.
- Display the image using the `image` command

First of all, the header dimension has to be determined; under LINUX or other UNIX-variants, this can be accomplished by typing the `du -b` – command:

```
wbirk@vela: $ du -b PIG_MR
535586 PIG_MR
```

This is the size of the image file – 535586 bytes; under Windows-type operating systems, the size of the image can be determined using the `Properties`-dialog, where both the file size and the occupied disk space are presented. Figure 2.10 shows such a dialog from a GNOME-based Linux flavor.

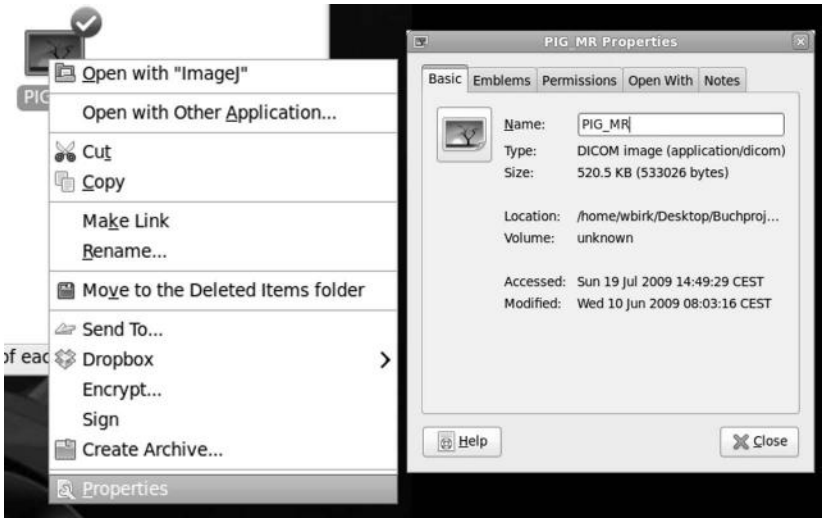

Figure 2.10: The size of the `PIG_MR` DICOM file as displayed by clicking the `Properties ...` button on a GNOME-Desktop (left hand side of the image). We are interested in the filesize. Based on this information, we can determine the size of the DICOM-header.

Sometimes, two sizes for a file are given under some operating systems – the true file size, and the disk space occupied. We are only interested in the smaller number of the two, the file size. The image size of `PIG_MR` is also known, and it can be computed as $512 * 512 * 2$. The first two numbers are the matrix size of the image, and the last one is the image depth – the number of bytes per pixel (16 bit = 2 byte). This results in an image size of 524288 bytes; computing the difference between the file size and the image size results in the header size, which is 8446 bytes. The first step now is to open the file in MATLAB, and to skip the header so that the relevant image information can be read. For this purpose, a file pointer – a variable that holds the actual information as read from the file – is to be assigned using the command `fopen(...)`. The working directory (the directory accessed by MATLAB/Octave unless specified otherwise) is to be assigned using the `cd` command, or in the menu bar of MATLAB (see also Figure 2.11). If `fopen('...')` is successful, a value different to -1 is returned by the interpreter. The first 8446 bytes – the header – are to be skipped. This is achieved using the `fseek(...)` command, which positions the file pointer.

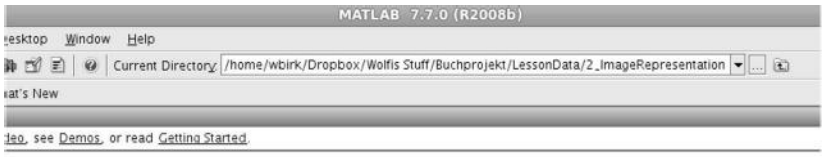

Figure 2.11: A screenshot of the MATLAB main menu; the working directory – the directory where all scripts and images are found – has to be set in this text entry-widget. Alternatively, one can also type in the `cd` command, followed by the path to the `LessonData` folder provided with this book.

```
1:> fpointer=fopen('PIG_MR','r');
2:> fseek(fpointer,8446,'bof');
```

By now, the working directory is set to the location of the DICOM file, a file pointer with read-only access (specified by the `'r'` character in `fopen(...)`) is successfully assigned, and this file pointer is shifted for 8445 bytes starting from the *beginning of the file* (thus the `'bof'` qualifier in line 2). The operation is successful, therefore the interpreter delivers a non-negative number for the file pointer. However, output is suppressed here by using the `;` character at the end of each line.

The next step is to assign a $512 * 512$ matrix that holds all the gray values ρ; this matrix is initially filled with zeros using the `zeros(512,512)` command. The trailing `;` is of great importance here since it suppresses the output.

```
3:> img = zeros(512,512);
```

The next step is to read the following $512*512 = 262144$ values of ρ, represented as numbers of two bytes length. This is accomplished by assigning each position in the matrix to a value of data type `'short'` using the command `fread`:

```
4:> img(:)=fread(fpointer,(512*512),'short');
```

The suffix `(:)` prompts the interpreter to read the values in the order of 512 numbers per row of the matrix; again, the semicolon suppresses output of the full matrix. `'short'` is the datatype of the image, in this case 16 bit. Finally, the matrix can be displayed graphically. For this purpose, the command `image` is used. The result depends on the actual version of MATLAB or Octave used; my Octave version spawns *gnuplot* as an external viewer, but this may vary and we witnessed problems with Octave sometimes; if such problems occur, it may be necessary to install another viewer application for Octave. In MATLAB, an internal viewer is started. It has to be noted that `image` is a method to show general geometrical features of a matrix, and it is not a genuine tool for viewing images. Therefore, a rather colorful image is the result since a standard-LUT is applied. The LUT can, however, be changed by calling the command `colormap(gray)`. Finally, it is obvious that rows and columns are interchanged, therefore the matrix has to be transposed. This is achieved by using the command `transpose(img);`. Denote that the `image` command does usually not care about the proper ratio of width and height for an image – this has to be adjusted manually.

```
5:> img=transpose(img);
6:> colormap(gray)
7:> image(img)
```

It is of course also possible to store the results of our efforts using the PGM format; according to the example given in section 2.3, the header of an ASCII-encoded PGM file consists of three lines containing the format prefix `P2`, the image format ($512 * 512$ pixels in this case), and the maximum data value. The maximum value of a matrix is computed in MATLAB by applying the operator `max(max(...))`. Calling the function `max` for a matrix returns a vector with the maximum element for each column of the matrix; calling `max` the second time returns the biggest value from this vector, which is the maximum value for the array. All we have to do is now to open another file pointer for *write* access – that is the `'w'` character in `fopen(...)`; all values to be saved have to be converted to strings, which is done by using the function `sprintf`. `sprintf` – well known to C-programmers – assigns a series of characters to a single string `str`, whereas `fprintf` writes this string to a file specified by the file pointer `pgmfp`. The character `\n` is a non printable *newline* character that causes a line feed. The `%d` specifier tells `sprintf` that an integer number – `maximumValue` in our case here – is to be inserted.

```
8:> pgmfp = fopen('PIG_MR.pgm','w');
9:> str=sprintf('P2\n');
10:> fprintf(pgmfp,str);
11:> str=sprintf('512 512\n');
12:> fprintf(pgmfp,str);
13:> maximumValue=max(max(img))
14:> str=sprintf('%d\n',maximumValue);
15:> fprintf(pgmfp,str);
```

By now, we have made a perfectly fine header for an ASCII-encoded PGM-file

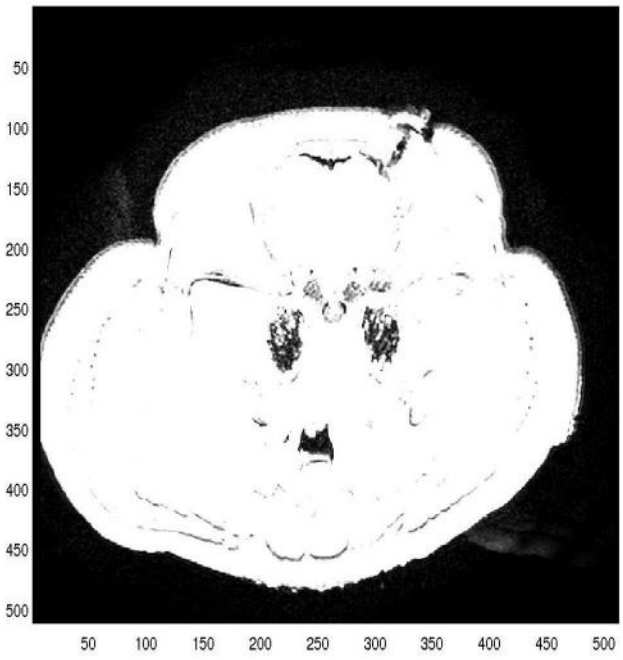

Figure 2.12: The outcome of calling the `image` command; the range of gray values ρ is not optimized, therefore the image is too bright. Optimizing the intensity range of images will be introduced shortly in Chapter 3.

that contains an image of $512 * 512$ pixels and a maximum gray value of 892; now, the single gray values ρ have to be saved, separated by spaces. In the following `for`-loop, the semicolon-suffixes for suppressing output are especially important.

```
16:> for i=1:512
17:> for j=1:512
18:> str=sprintf('%d ',img(i,j));
19:> fprintf(pgmfp,str);
20:> end
21:> end
22:> fclose(pgmfp);
23:> fclose(fpointer);
```

By now, a PGM-file readable by any graphics program that supports this format should appear in your working directory. The final call to the function `fclose` is

especially important for forcing all data to be written to the hard disk. Figure 2.13 shows the looks of `PIG_MR.pgm` in the GIMP.

Figure 2.13: The `PIG_MR.pgm` file as generated by the `OpenDICOM_2.m` script, opened in the GIMP. The gray values are now represented properly, and we can identify several anatomical structures such as the temporomandibular joint, the brain, and huge amounts of muscles and fatty tissue.

Those who have the Image Processing Toolbox of MATLAB available may also use the command `dicomread`. Throughout this book, however, MATLAB is used a teaching vehicle, and we therefore abstain from using ready made solutions for decoding image data when possible.

Additional Tasks

- Repeat the same procedure with the data set `PIG_CT`, which is a 512×512 image of 16 bit depth. Saving the image as PGM is, however, problematic due to to the fact that the minimum value is negative, not zero. If you cannot figure out a way to deal with this range of image intensities, you may read Chapter 3 first.

2.7.2 Image subtraction

In the script `dsa_2.m`, we see an example of image subtraction; two x-ray images of the pelvis, one before and one after injection of a contrast agent named `DSA1.jpg`

and `DSA2.jpg` are provided in the `Lesson Data` folder. Figure 2.14 shows these two images.

Figure 2.14: The images `DSA1.jpg` and `DSA2.jpg`, two x-ray images of a pelvis taken before and after injection used in Example 2.7.2. Subtraction of the two images yields an image that only shows the change between the two exposures – in this case, a vessel flooded by contrast agent. In clinical routine, this method is called digital subtraction angiography (DSA). (Image data courtesy of Z. Yaniv, ISIS Center, Department of Radiology, Georgetown University.)

If we subtract the two images, we will get an image that only shows the part of the image that has changed. In clinical routine, this method of visualizing vessels after application of a contrast agent is called *digital subtraction angiography* (DSA). As opposed to reading the images byte wise, we may use a MATLAB function named `imread` for loading image data. This is definitely more handy than the verbose way of loading an image as shown in Example 2.7.1. For storing an image, a function `imwrite` also exists. If an 8 bit representation of our images is sufficient, we may therefore use `imread`. Two matrices `img1` and `img2` are filled with data from the x-ray images stored as JPG files using `imread`:

```
1:> img1=imread('DSA1.jpg');
2:> img2=imread('DSA2.jpg');
```

Next, we carry out the subtraction of the images, which have the same dimension; the result is displayed using `image` and can be found in Figure 2.15:

```
3:> resimg=(img1-img2);
4:> colormap(gray);
5:> image(resimg)
```

Figure 2.15: The result of `dsa_2.m`; only the vessel which shows up in the second x-ray `DSA2.jpg` is clearly visible after image subtraction. (Image data courtesy of Z. Yaniv, ISIS Center, Department of Radiology, Georgetown University.)

2.7.3 Converting a color image to gray scale

In the case of a color image, `imread` returns three 3D-arrays of numbers containing the three color channels. This information can be fused to a gray scale image by computing the luminance of the single colors as perceived by the eye. Therefore, a gray scale image (as seen, for instance, by a color blind person) is computed from the RGB-images as $\rho = 0.3\rho_R + 0.59\rho_G + 0.11\rho_B$. The factors stem from the spectral sensitivity of the eye in daylight, which is highest at a wavelength of approximately 550 nm – which is actually green. First of all, the working directory has to be determined, and the color PET-image is loaded; the script for this example is called `ConvertColorImage_2.m`:

```
1:> colorimg = imread('PET_image.jpg');
```

We can determine the size of `colorimg` by calling the function `size`, which returns the actual dimensions of an array.

```
2:> size(colorimg)
```

The script returns the following line: `ans = 652 1417 3`. Apparently, three images, each of size 652×1417 pixels were loaded. MATLAB has to decode the JPG-type file first into a raw data format. The color components are therefore also converted from the JPG-type color space Y-Cr-Cb, and the three images in the array `colorimg` contain the R-G-B components. The only thing left to do is to allocate memory for the gray scale image and to compute the gray value ρ from the three components. Finally, the result of our efforts is scaled to 64 gray values ranging from 0 to 64 and displayed. The result should look similar to Figure 2.16.

```
3:> grayimg=zeros(652,1417);
4:> for i=1:652
5:> for j=1:1417
6:> grayimg(i,j) = 0.3*colorimg(i,j,1) +
0.59*colorimg(i,j,2) + 0.11*colorimg(i,j,3);
7:> end
8:> end
9:> imin=min(min(grayimg));
10:> grayimg=grayimg-imin;
11:> imax=max(max(grayimg));
12:> grayimg= floor(grayimg/imax*64);
13:> image(grayimg)
14:> colormap(gray)
```

`floor` generates an integer value by cutting of the numbers behind the decimal point – it is a crude operation that always rounds off.

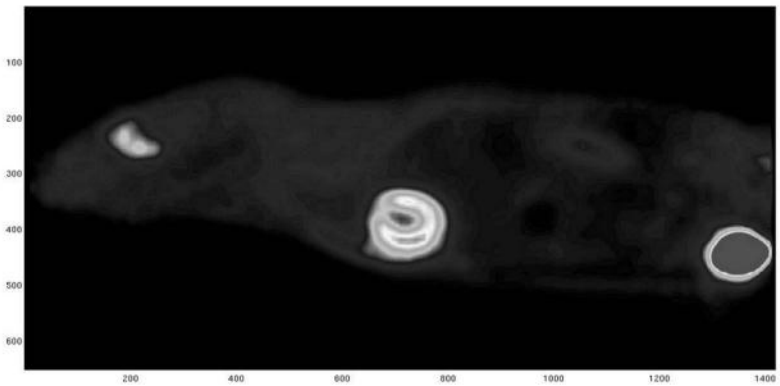

Figure 2.16: Result of converting a color PET image to a gray scale image using the weighted addition of red, green and blue channels according to the photopic sensitivity of the eye. (Image data courtesy of C. Kuntner, AIT Seibersdorf, Austria.)

2.7.3.1 A note on good MATLAB programming habits

As already pointed out in the introduction, we have tried to keep the code as simple and as straightforward as possible, at the cost of performance. In this example it is, however, easily possible to demonstrate the strengths of MATLABs vector notation style for avoiding `for` loops. Lines 4 to 8 in `ConvertColorImage_2.m` can be replaced by a single line that shows the same effect:

```
4:> grayimg(:)=0.3*colorimg(:,:,1) +
0.59*colorimg(:,:,2) + 0.11*colorimg(:,:,3);
```

By utilizing the vector notation `(:)`, the double `for` loop becomes obsolete. In the `LessonData` folder, you find this modified example under `BetterConvertColorImage_2.m`. With larger datasets, the result of this programming technique is a considerable improvement in performance. One cannot replace all statements which access single matrix elements, but if you run into performance problems with one of the scripts provided, you may try to apply the vector notation instead of nested loops.

2.7.4 Computing the SNR of x-ray images as a function of radiation dose

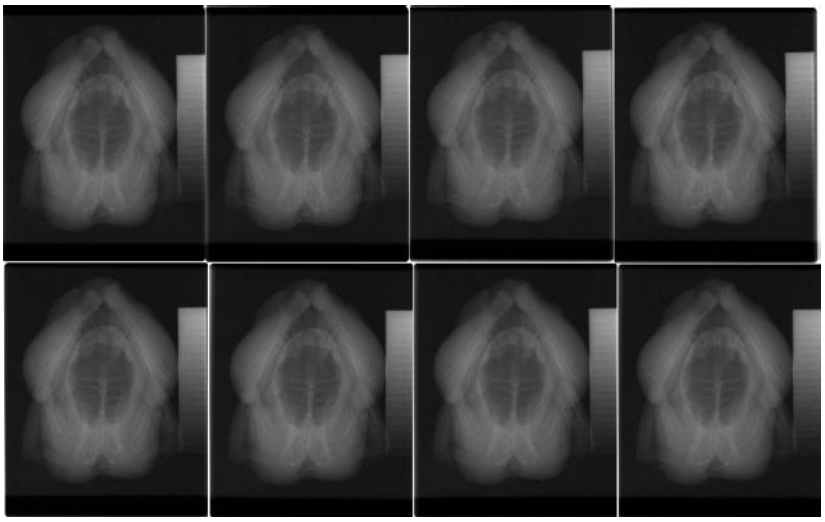

Figure 2.17: The eight x-ray images of a chicken, stored in original resolution as DICOM-files acquired from a CR plate reader. All images were acquired with different dose; while the overall appearance is similar due to the large tolerance of CR plates to under- and overexposure, the SNR varies considerably. (Image data courtesy of P. Homolka, Center for Medical Physics and Biomedical Engineering, Medical University of Vienna.)

Next, a set of eight x-ray images of a dead chicken is provided; the images were

exposed using 1/8, 1/4, 1/2, 1, 2, 4, 8, and 16 times the dose required. We will compute the SNR of these images and take a look at the SNR-values in dependence of the dose. The `SNRExample_2.m` does the following: first of all, we have to change to the sub-directory `SNR`, where eight DICOM files named `chicken1.dcm` to `chicken8.dcm` can be found. The size of the headers is stored in a vector named `headersize`. A vector `snrs`, which will hold all the signal-to-noise ratios, is defined as well:

```
1:> cd SNR
2:> headersize=zeros(8,1);
3:> headersize(1,1)=1384;
4:> headersize(2,1)=1402;
5:> headersize(3,1)=1390;
6:> headersize(4,1)=1386;
7:> headersize(5,1)=984;
8:> headersize(6,1)=988;
9:> headersize(7,1)=984;
10:> headersize(8,1)=988;
11:> snrs=zeros(8,1);
```

Next, we have to read the images and the signal-free sub-area where the standard deviation of the detector is determined:

```
12:> for i=1:8
13:> signalFreeArea=zeros(350,300);
14:> img=zeros(2364,2964);
15:> fname=sprintf('chicken%d.dcm',i);
16:> fp=fopen(fname);
17:> fseek(fp,headersize(i,1),'bof');
18:> img(:)=fread(fp,(2364*2964),'short');
19:> fclose(fp);
20:> img=transpose(img);
21:> for j=100:450
22:> for k=200:500
23:> signalFreeArea(j-99,k-199)=img(j,k);
24:> end
25:> end
26:> sigmaSignalFree=std(signalFreeArea(:));
27:> snrs(i)=mean(mean(img))/sigmaSignalFree;
28:> end
```

What has happened? The eight files were opened just as we did in Example 2.7.1, and the image matrix was transposed in order to keep a sufficient image orientation compared to the GIMP program, where the signal-free area was defined (see Figure 2.18). The signal-free area was stored in a separate matrix `signalFreeArea`. The standard deviation for the elements in the matrix `signalFreeArea` was computed using the command `std`.

By now, we have a vector `snrs` containing the SNR of images `snr1.jpg`

Figure 2.18: Selection of an appropriate signal-free area of the image using the GIMP. For this example, an area defined by the rectangular section in the upper left of the screenshot was used. The actual coordinates are given in the lower left of the dialog.

...`snr8.jpg`; it would be a nice idea to sort those SNRs. Practically, this is a linked list since we want to know which image has the best SNR. Such a hash table can be programmed – but it is cumbersome and usually slow. Therefore we may use a nasty little trick. We encode the file name into the values. The SNRs have values between 107.03 to 363.39; these values can be rounded to integer values without loss of information:

```
29:> snrs=round(snrs);
```

Now comes the ugly part. In the following section, we *encode* the ordinal number of the image in the decimal area of the values, and we sort the vector; this is a trick to circumvent a hash-table – a linked list of paired values, where one of the entries is sorted. Such structures are usually computationally expensive. Therefore we add the ordinal number of the image that belongs to the SNR as the decile rank to an otherwise integer number:

```
30:> for i=1:8
31:> snrs(i)=snrs(i)+i/10;
```

```
32:> end
33:> ssnrs=sort(snrs)
34:> cd ..
```

From the output, we see that the images, sorted by SNR, are given as `chicken3.dcm`, `chicken4.dcm`, `chicken1.dcm`, `chicken2.dcm`, `chicken7.dcm`, `chicken8.dcm`, `chicken5.dcm`, and `chicken6.dcm`. Finally, we return to the parent directory.

Additional Tasks

- Compute the CNR of the image series.
- The MATLAB function `sort` also returns a permutation from the original vector to the sorted version. Use this to sort the images by their SNR.

2.8 Summary and further references

Two- and three dimensional images are, strictly speaking, scalar mathematical functions defined on two- or three dimensional grid structures. The single 2D picture element, which is represented as a coordinate, an area, and an associated gray value, is called a pixel. The 3D equivalent is a volume element or voxel. Color images are similar to gray scale images; each color channel is stored in a separate gray scale image – however, color does not play a very important role in medical imaging. Images can be saved in various uncompressed and compressed formats. The most important format for 2D and 3D medical image data is DICOM, which does not only consist of a method of storing pixels and voxels, but also of communication and storage methods and more.

Literature

- J. D. Murray, W. van Ryper: Encyclopedia of Graphics File Formats, O'Reilly, (1996)
- O. S. Pianykh: Digital Imaging and Communications in Medicine (DICOM): A Practical Introduction and Survival Guide, Springer, (2008)
- http://medical.nema.org/
- A. Oppelt (ed.): Imaging Systems for Medical Diagnostics: Fundamentals, Technical Solutions and Applications for Systems Applying Ionizing Radiation, Nuclear Magnetic Resonance and Ultrasound, Wiley VCH, (2006)