Visual Computing

by dcamenisch

1 Introduction

This document is a summary of the 2022 edition of the lecture *Visual Computing* at ETH Zurich. I do not guarantee correctness or completeness, nor is this document endorsed by the lecturers. If you spot any mistakes or find other improvements, feel free to open a pull request at `https://github.com/DannyCamenisch/vc-summary`. This work is published as CC BY-NC-SA.

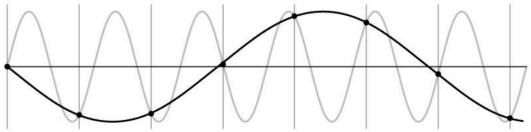
2 The Digital Image

An image is simply a continuous function over 2 or 3 variables (XY-coordinates and possibly time). Usually we use brightness as the value of the function, but other physical values can also be used. For a computer this is just a collection of numbers, but instead of continuous values we have discrete. Note that in real life images are never completely random and almost always contain some structure. It is important to know that **pixels are not little squares**, they are point measurements.
When taking a picture with a digital camera, we can encounter various problems, e.g.:

- Transmission Interference
- Compression Artefacts
- Spilling
- Sensor Noise

2.1 Sampling

When taking an image, we are sampling such a continuous. When trying to reconstruct the original function, we can encounter undersampling, i.e. when we loose information due to a too low amount of sampling points.

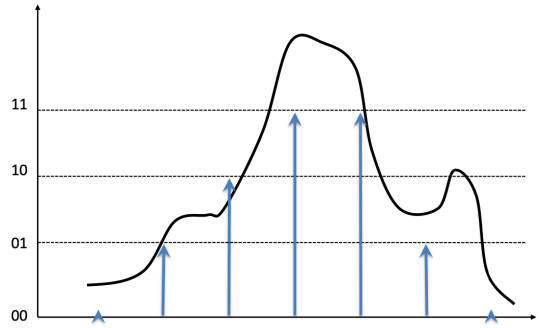


Due to undersampling, the result can not be distinguished from a lower or a higher frequency wave. Signal disguised as other frequencies is also called **aliasing**.

Nyquist-Shannon Sampling Theorem
For sine waves we have to sample at half the wave length. This corresponds to double the frequency, we also call this the **Nyquist Frequency**.

2.2 Quantization

Another problem we have to deal with is **quantization**, since the real valued function will get digital (integer) values, it is lossy. Compared to sampling which lets us reconstruct the original function. Simple quantization uses equally spaced levels with k intervals.



2.3 Image Properties

Image resolution is divided into two parts:

- Geometric Resolution: How many pixels per area
- Radiometric Resolution: How many bits per pixel

2.4 Noise

When taking pictures we can almost always encounter some noise. A common way to model this is additive gaussian noise:

$$I(x,y) = f(x,y) + c, \qquad c \sim \mathcal{N}(0,\sigma^2)$$

The signal to noise ratio (SNR) is an index of image quality:

$$\text{SNR} = \frac{F}{\sigma}, \qquad F = \frac{1}{XY}\sum_{x=1}^{X}\sum_{y=1}^{Y} f(x,y)$$

The usefulness for this metric can vary drastically depending on the type of image (dark images will have a higher SNR compared to bright images). Therefore we introduce peak SNR:

$$\text{PSNR} = \frac{F_{\max}}{\sigma}$$

3 Segmentation

Image segmentation is often viewed as the ultimate classification problem, once solved, computer vision is solved. A complete segmentation of an image is a finite set of disjunct regions $R_1, ..., R_n$, such that $I = \bigcup R_i$.

3.1 Thresholding

Thresholding is a simple segmentation process, that produces a binary image by labelling each pixel in or out of the region of interest. We do this by comparison of the grey level with a threshold value T. Another, better approach can be chromakeying. Hereby we measure the distance from a defined color g:

$$I_\alpha = |I - g| > T$$

One limit of thresholding is that it does not consider image context.

3.2 Segmentation Performance

If we want to choose the best performing segmentation algorithm or determine a good value for T, we need a performance metric. To use automatic analysis, one needs to know the true classification of each test, for this the test images have to be segmented by hand.
One performance metric is the ROC curve. This curve characterizes the error trade-off in binary classification tasks, by plotting the true positive fraction against the false positive fraction. We often choose the operating point on the ROC curve, by assigning cost and values to each outcome:

- V_{TN} - value of true negative
- V_{TP} - value of true positive

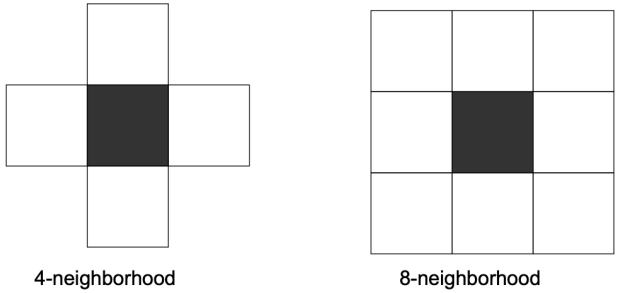
- C_{FN} - cost of false negative
- C_{FP} - cost of false positive

We then choose the point on the ROC curve with the gradient:

$$\beta = \frac{N}{P} \cdot \frac{V_{\text{TN}} + C_{\text{FP}}}{V_{\text{TP}} + C_{\text{FN}}}$$

3.3 Pixel Connectivity

We try to define which pixels are neighbors.



A 4 (or 8) connected path between p_1, p_n is a set of pixels such that every p_i is a 4 (or 8) neighbor of p_{i+1}. Now we can defined a region as 4 (or 8) connected if it contains a 4 (or 8) connected path between any two of its pixels.
With this we can introduce **region growing**. We start from a seed point or region and add neighboring pixels that satisfy the criteria defining a region until we can include no more pixels. There are different approaches to selecting the seed and we could also use multiple seeds. For the inclusion criteria we could choose thresholding or a distribution model.
Another criteria is a snake (active contour). While each point along the contour moves away from the seed, it always has to have some smoothness constraints (minimizing energy function).

3.4 Distance Measures

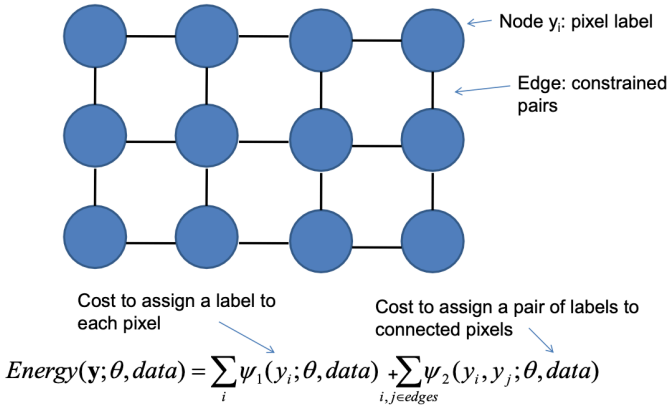
Plain background substraction $I_\alpha = |I - I_{bg}| > T$, where I_{bg} is the background image, we get this by fitting a Gaussian (Mixture) model per pixel. Even better would be:

$$I_\alpha = \sqrt{(I - I_{bg})^\top \Sigma (I - I_{bg})} > T$$

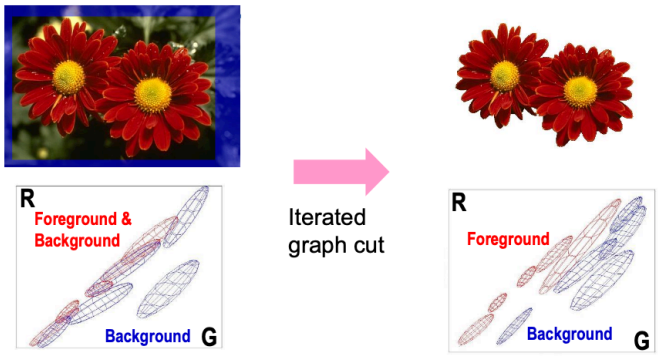
Where Σ is the background pixel appearance covariance matrix.

3.5 Markov Random Fields

We can add spatial relations with Markov Random Fields (2D Markov Chains).



Using a graph cut algorithm we can determine the optimal segementation. We can further optimize this by using iterated graph cut and k-means for learning the colour distribution (GMM) of the image.



4 Convolution and Filtering

Convolution and filtering are some of the most basic operations of image processing.

4.1 Filtering

Image filtering is the process of modifying pixels in an image base on some function of a local neighborhood of the pixel.

4.1.1 Linear Shift-Invariant Filtering

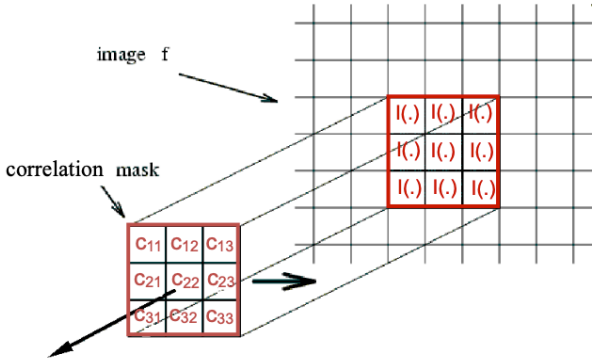
Linear lhift-invariant filtering means using linear combinations of neighbors and doing the same for each pixel (shift-invariant). These filters are often used for low-level image processing, smoothing / noise reduction, sharpening and feature detection. Linear operations can be written as:

$$I'(x,y) = \sum_{(i,j)\in N(x,y)} K(x,y;i,j) I(x,y)$$

Here I is the input image, I' the output image, K is the kernel and N is the neighborhood. Operations are shift-invariant if K does not depend on (x,y).

4.2 Correlation

Correlation, e.g. template matching:

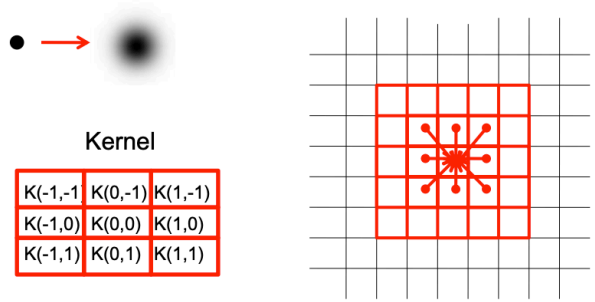


$$I' = K \circ I, \quad I'(x,y) = \sum_{(i,j)\in N(x,y)} K(i,j) I(x+i, y+j)$$

Correlation takes an input image and a weight mask, then each pixel gets "replaced" by the weighted sum of its neighborhood. This can be described as taking multiple input location and writing one output location.

4.3 Convolution

Convolution, e.g. point spread function:



$$I' = K * I, \quad I'(x,y) = \sum_{(i,j)\in N(x,y)} K(i,j)I(x-i,y-j)$$

This is similar to the correlation, but **the kernel is reversed**. This can be described as taking one input location and writing multiple output location, the opposite of the correlation. By default we use convolution for filtering. An example for a kernel would be:

$$K = \begin{bmatrix} 0 & 0 & 0 \\ 0 & 2 & 0 \\ 0 & 0 & 0 \end{bmatrix} - \frac{1}{9}\begin{bmatrix} 1 & 1 & 1 \\ 1 & 1 & 1 \\ 1 & 1 & 1 \end{bmatrix}$$

This kernel is used for sharpening by accentuating differences with the local average. Another example would be:

$$K = \begin{bmatrix} -1 & 0 & 1 \\ -2 & 0 & 2 \\ -1 & 0 & 1 \end{bmatrix}$$

This kernel looks for differences in the horizontal direction, this corresponds to finding vertical edges.

4.3.1 What about the Edges?

If we apply our filters to images, we need to **deal with the edges separately**. This is due to our window falling off the edge of the image. There are different techniques to deal with this problem:

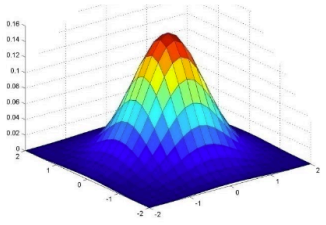
- Extend the image with black border
- Wrap the kernel around the edges
- Copy out the edge
- Mirror the image at the edge
- Vary filter near the edge

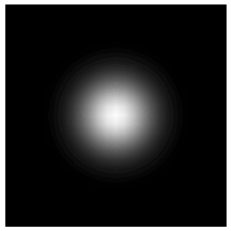
4.3.2 Separable Kernels

A kernel is separable, if it can be written as $K(m,n) = f(m)g(n)$. This means that the kernel can be separated into a function for the first coordinate and another for the second coordinate. If this is the case we can apply the separated functions individually to the image.

4.3.3 Gaussian Kernel

The idea of the **Gaussian Kernel** is to weight the contributions of neighboring pixels by nearness.





$$G_\sigma = \frac{1}{2\pi\sigma^2} {}^{-\frac{(x^2+y^2)}{2\sigma^2}}$$

We can use the Gaussian Kernel for image smoothing, the best part being that the kernel is separable. The actual amount of smoothing depends on σ and the window size.

If we repeatedly apply the Gaussian filter, we produce the scale space of an image.

4.3.4 High-Pass Filters

High-pass filters are used to detect areas of the image where a lot is happening (high frequencies). Examples for these are the Laplacian operator K or the high-pass filter K':

$$K = \begin{bmatrix} 0 & 1 & 0 \\ 1 & -1 & 1 \\ 0 & 1 & 0 \end{bmatrix} \qquad K' = \begin{bmatrix} -1 & -1 & -1 \\ -1 & 8 & -1 \\ -1 & -1 & -1 \end{bmatrix}$$

High-pass filters can be used to perform image sharpening $I' = I + \alpha|K * I|$.

5 Image Features

Image features are about detecting the location of patterns in images, e.g. edge detection or facial landmarks.

5.1 Template Matching

Given an template t, e.g. template describing an eye, we want to locate an area, in an image s, that best fits this template. Alternatively we could also look for areas that match this template by a certain threshold.

To search for the best match, we try to minimize the mean squared error. This is the same as maximizing the area correlation:

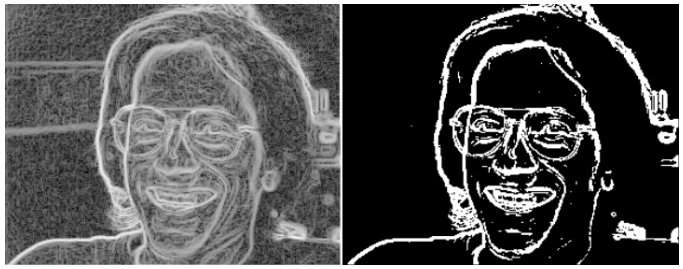
$$r(p,q) = \sum_{x=-\infty}^{\infty}\sum_{y=-\infty}^{\infty} s(x,y)\cdot t(x-p,y-q) = s(p,q) * t(-p,-q)$$

5.2 Edge Detection

We have previously seen kernels that can detect horizontal edges. To expand on this, we differentiate the following kernels:

$$\text{Prewitt}\begin{bmatrix} -1 & 0 & 1 \\ -1 & 0 & 1 \\ -1 & 0 & 1 \end{bmatrix} \qquad \text{Sobel}\begin{bmatrix} -1 & 0 & 1 \\ -2 & 0 & 2 \\ -1 & 0 & 1 \end{bmatrix}$$

We can also transpose these kernels to detect horizontal edges. From the resulting images (horizontal / vertical edges) we take the log sum squared and use different thresholds to achieve the final result.

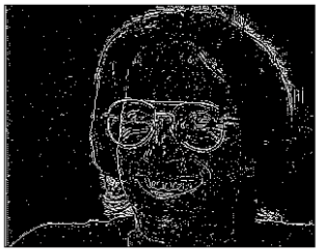


5.2.1 Laplacian Operator

The idea behind the Laplacian operator is to detect discontinuities in the second derivative. This corresponds to detecting zero-crossings. The operator is isotropic (rotationally invariant) and can be implemented with one of the following kernels:

$$\begin{bmatrix} 0 & 1 & 0 \\ 1 & -4 & 1 \\ 0 & 1 & 0 \end{bmatrix} \quad \text{or} \quad \begin{bmatrix} 1 & 1 & 1 \\ 1 & -8 & 1 \\ 1 & 1 & 1 \end{bmatrix}$$

This operator is very sensitive to fine details and noise, therefore we might need to blur the image first. Additionally it will respond equally to weak and strong edges, so we want to suppress edges with low gradient magnitude. Blurring and applying the Laplacian operator can be combined into a convolution with Laplacian of Gaussian (LoG). Combining LoG with gradient based threshold delivers the best result.

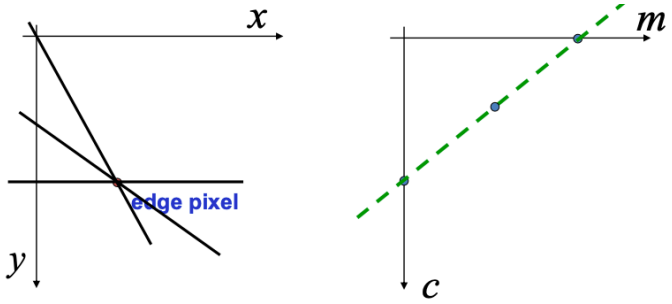



5.2.2 Canny Edge Detector

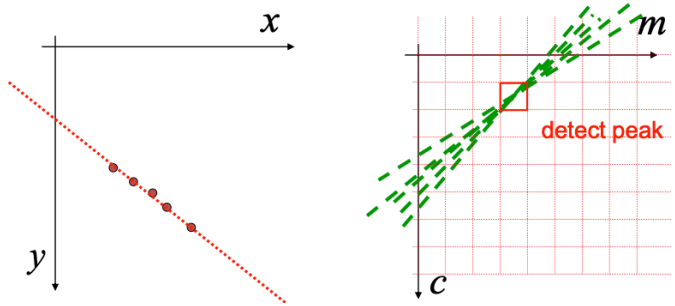
The Canny edge detector works by first smoothing the image with a Gaussian filter. Then we compute the gradient magnitude (Sobel, Prewitt, ...) and the angle of the gradient. After this we want to apply non-maxima suppression tot he gradient magnitude image. Combining this with double thresholding, to detect strong and weak edge pixels, and rejecting weak edge pixels not connected with strong edge pixels, results in the Canny edge detector.


5.3 Hough Transform

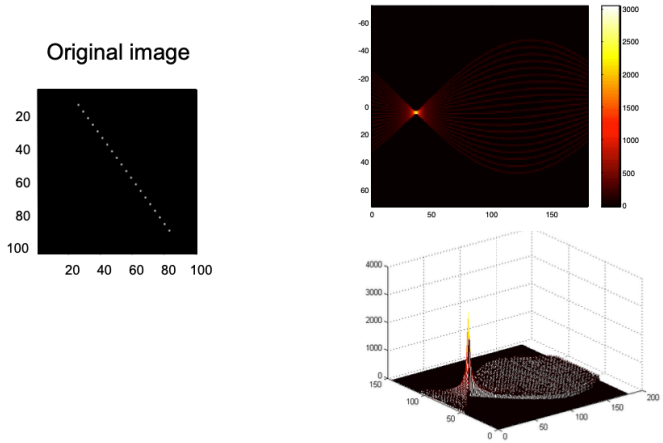
Hough transform can be used to find higher order entities in an image, e.g. lines or circles. The Hough transform is a generalized template matching technique. Considering detection of straight lines (y = mx + c), for each edge pixel there are infinitely many possible lines. We plot these possible lines in the 2D space formed by its parameters, we end up with a single line.



If we do this for multiple edge points and subdivide the parameter space into discrete bins, we can find the bin with the most possible lines. This gives us the detected line.



There is a problem with this approach, the parameter space is infinite. To avoid this problem we choose an alternative parametrization, in this case we represent a line as an angle and the distance from the origin. Now the representations in parameter space are not lines but sine waves. Again we find the maxima to find our lines.



To find multiple lines we do non-maxima suppression and keep every strong peek. To expand this concept to circle detection we simply change the parameter space.

5.4 Keypoint Detection

We might want to only find corners and not edges. We want this corner localization to be accurate, invariant and robust. We define the following:

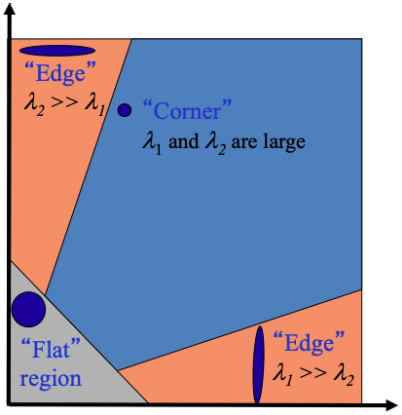
$$\mathbf{M} = \left(\sum_{(x,y)\in \text{window}} \begin{bmatrix} f_x^2(x,y) & f_x(x,y)f_y(x,y) \\ f_x(x,y)f_y(x,y) & f_y^2(x,y) \end{bmatrix} \right)$$

$$S(\Delta x, \Delta y) = (\Delta x, \Delta y)\ \mathbf{M}\ (\Delta x, \Delta y)^\top$$

Hereby f_x is the horizontal gradient and f_y the vertical gradient. $\mathbf{M}$ is called the structure matrix or nomal matrix. To detect feature points we know try to find points for which $\min \Delta^\top \mathbf{M} \Delta, ||\Delta|| = 1$ is large. This is the same as maximizing the eigenvalues of $\mathbf{M}$. The eigenvalue allow us to define a measure of "cornerness" (smaller k means more strict):

$$C(x,y) = \det \mathbf{M} - k \cdot (\text{trace } \mathbf{M})^2 = \lambda_1 \lambda_2 - k \cdot (\lambda_1 + \lambda_2)^2$$

If we plot the values for the eigenvalues, we can divide the space as follows:



One problem of this approach is that it is not invariant to scale.

To compare different images, e.g. for combining images. We use thresholded image gradients that are sampled over an array of locations in scale space. This can then be used for example to stich images together (SIFT).

6 Fourier Transformation

We have already seen the problem of aliasing. Now we want to understand how we can avoid aliasing and for this we introduce the **Fourier Transformation**.

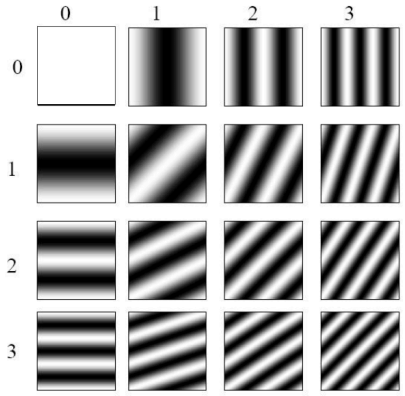
The idea behind the Fourier transformation is to perform a change of basis, where the new basis elements are of the form $e^{-i2\pi(ux+vy)} = \cos(2\pi(ux+vy)) - i\sin(2\pi(ux+vy))$ (u, v are the parameters for the new basis).

$$F(g(x,y))(u,v) = \int\int_{R^2} g(x,y)e^{-i2\pi(ux+vy)}dxdy$$

The basis functions of Fourier transform are eigenfunctions of linear systems (one of the reasons it is so popular). Discrete Fourier transformation (DFT) can be represented as:

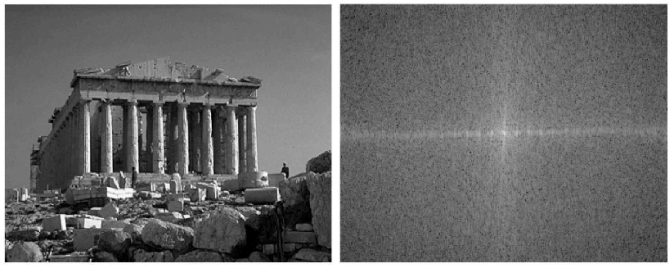
$$F = \mathbf{U}f$$

Here $\mathbf{U}$ is the Fourier transform base. The vector (u, v) determines the frequency by its magnitude and the orientation by its direction (only looking at the real part):



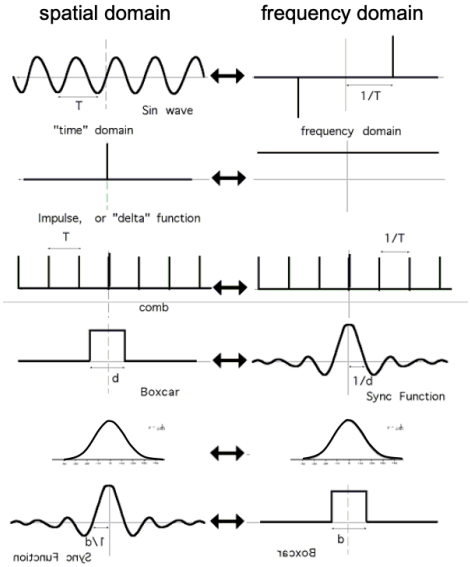
6.1 Phase and Magnitude

Since the Fourier transform is complex, it is difficult to plot. Instead we think of the phase (angle) and magnitude (length of the vector) of the transform. Note that natural images have about the same magnitude transform, hence phase seems to matter more. The phase part seems more random than the magnitude. Here you can see an example of the magnitude of an image.



6.2 Properties of the Fourier Transform

We already said that the Fourier transform is linear. Further it is important to note that the FT of a Gaussian is a Gaussian.



Convolution Theorem

The FT of the convolution of two functions is the product of their FT and the other way around:

$$F \cdot G = \mathbf{U}(f * g), \qquad F * G = \mathbf{U}(f \cdot g)$$

6.3 Sampling

Now we have a look at aliasing again. We define our sampling function as follows:

$$\text{Sample}_{2D}(f(x,y)) = f(x,y)\sum_{i=-\infty}^{\infty}\sum_{j=-\infty}^{\infty}\delta(x-i,y-j)$$

Where δ is the Dirac delta function. The FT of a sampled signal now corresponds to:

$$F(\text{Sample}_{2D}(f(x,y))) = \sum_{i=-\infty}^{\infty}\sum_{j=-\infty}^{\infty}F(u-i,v-i)$$

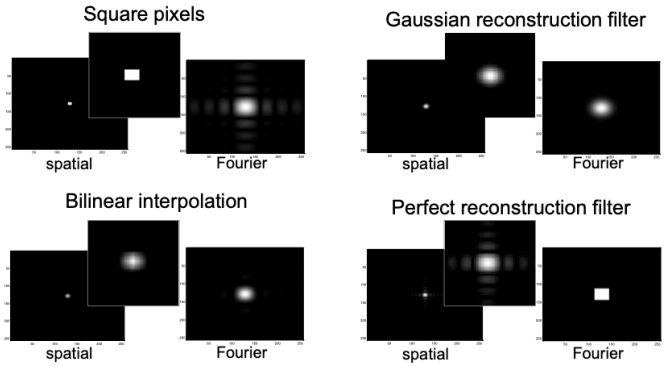
This approach can still lead to aliasing as high frequencies can lead to trouble. To avoid this we first need to suppress high frequencies before sampling. We can do this by convolution with a low-pass filter, this corresponds to multiplying the FT with the same filter. As a low-pass filter we use a Gaussian.

Nyquist Sampling Theorem

To understand why we need to do the low-pass filtering, we can take a look at the Nyquist sampling theorem: *The sampling frequency must be at least twice the highest frequency (of the signal).*

6.4 Signal Reconstruction

In image reconstruction we want to recreate our image from the samples data. To avoid pixelation we look at different reconstruction filters.



In the Fourier domain, determining the inverse of a kernel / filter becomes a lot easier, as:

$$F(h)(u,v) \cdot F(h^{-1})(u,v) = 1$$

With this we can for example try to remove motion blur from images. But we have to be careful to regularize our reconstruction filter to avoid noise amplification.

7 Unitary Transforms

Images can be either written as matrices or as vectors to make math easier. So a linear image processing system can be defined as:

$$g = \mathbf{A}f$$

So we ask ourself the question how to choose $\mathbf{A}$. We say $\mathbf{A}$ is unitary (or orthonormal for real values) if $\mathbf{A}^{-1} = \mathbf{A}^H$. Transformations by unitary matrices are energy conserving, meaning the length of the transformed vector will stay the same.
We introduce the following notation: f_i one image, $F = [f_1, ..., f_n]$ collection of images and $R_{ff} = E[f_i \cdot f_i^H] = \frac{F \cdot F^H}{n}$ image collection auto-correlation function. While unitary transformations preserve the energy, it will often be unevenly distributed among coefficients. The autocorrelation matrix of the transformed image will look like this:

$$R_{cc} = \mathbf{A} R_{ff} \mathbf{A}^H$$

The eigenmatrix Φ of R_{ff} is unitary and defined as follows:

$R_{ff}\Phi = \Phi\Lambda$ ← Λ is a diagonal matrix of eigenvalues λ_i

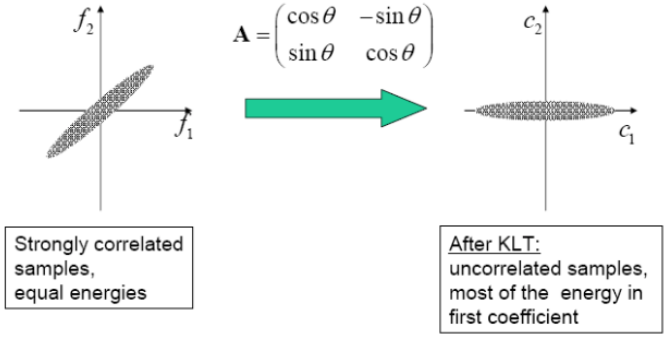
$$\Lambda = \begin{pmatrix} \lambda_0 & & 0 \\ & \lambda_1 & \\ & & \ddots \\ 0 & & \lambda_{MN-1} \end{pmatrix}$$

7.1 Karhunen-Loeve Transform / PCA

If we choose $\mathbf{A} = \Phi^H$ we get:

$$R_{cc} = \Phi^H R_{ff} \Phi = \Phi^H \Phi \Lambda = \Lambda$$

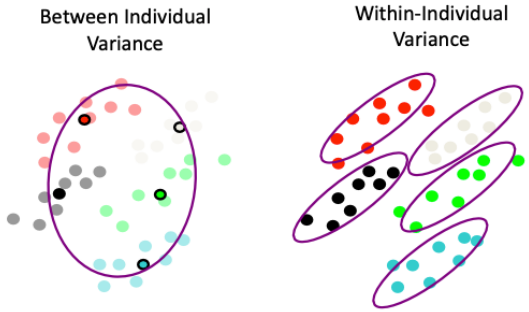
We can interpret this transformation as follows: "No other unitary transformation packs as much energy into the first k coefficients, where k is arbitrary". The mean squared approximation error by choosing only the first k coefficients is minimized.



The basis images, which are the eigenvectors of the autocorrelation matrix, are called **eigenimages**. We can use eigenimages for recognition tasks, as the high dimensionality of the images space can be reduced to k dimensions. To perform recogintion, we tailor a KLT / PCA to the specific set of images we want to recognize, for example this lead to Eigenfaces.

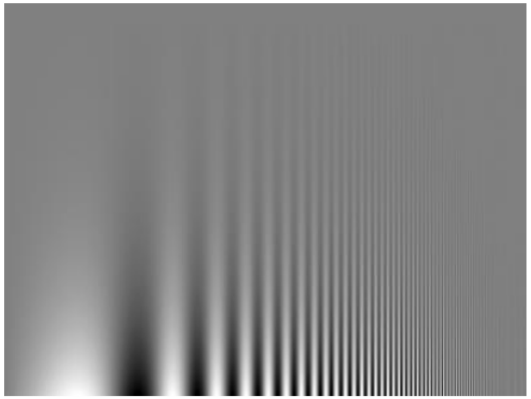
7.1.1 Fisherfaces

To improve on the short comings of Eigenfaces, Fisherfaces try to find the direction where the ratio between individual variance are maximized. As the math behind this is rather complex, it is left out.

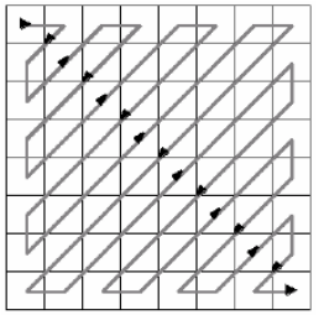


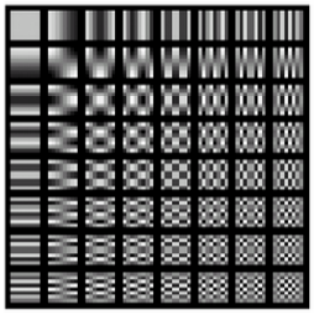
7.2 JPEG Compression

We notice that humans do not resolve high frequencies too well, therefore we can leave some of them away to reduce the size of an image.



This is one of the main concepts in JPEG compression. Instead of FT, JPEG uses discrete cosine transform (DCT), which has no imaginary part. We go through the components of the DCT in a snake like pattern.

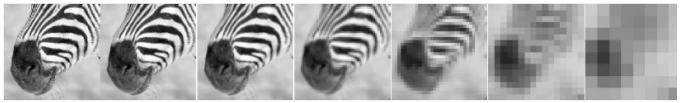




If the coefficients get too small after a certain point, we simply leave them away. In the end we apply Huffman encoding to further reduce the size of our image.

8 Pyramids and Wavelets

The **scale-space** is the family of signals generated by successive smoothing with a Gaussian filter. Using the scale-space we can downsample the image step by step, giving us an **image pyramid**.

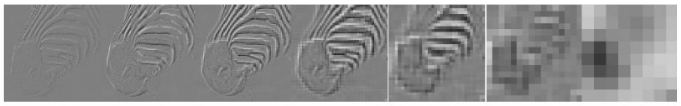


512 256 128 64 32 16 8

Such image pyramids can be used for edge tracking, search for correspondence, etc. One of their main benefits is that they allow for control of detail and computational costs.

8.1 The Laplacian Pyramid

While the Gaussian pyramid successively suppressed high frequencies, the Laplacian pyramid represents a different frequency at each level. This is similar to a bandpass filter.

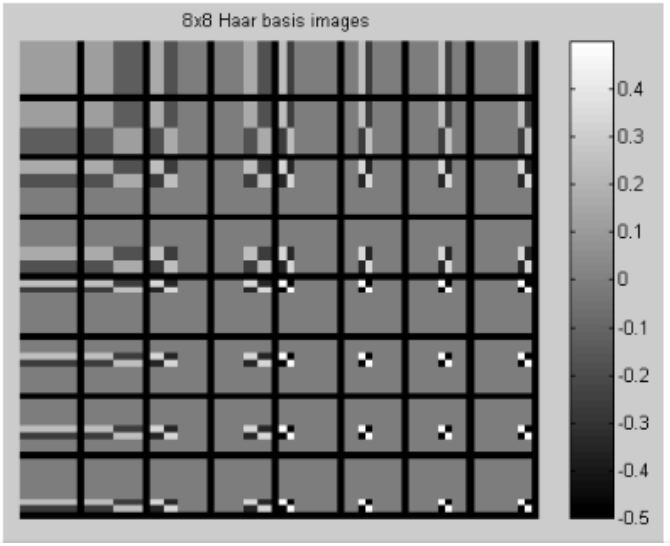


512 256 128 64 32 16 8

With a Laplacian pyramid we mean a difference of Gaussians (DoG). The main benefit is that it removes the redundancy of having the lower frequency parts in every level.

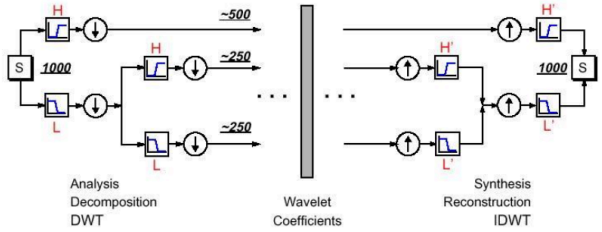
8.2 Wavelet Transform

The **Haar transform** is one of the most basic wavelets.



We can see that the outer vectors correspond to recursively applying a two-band filter to the bands of the previous stage (like a pyramid). The Haar transformation has poor energy compaction, meaning it is not really good for image compression.

In general wavelet transform works by splitting the signal into a low frequency and a high frequency pass, then the process is applied to the low frequency band recursively.



This concept is then again expanded to 2D images.

9 Optical Flow

The goal of optical flow is to estimate motion in videos. It is used for tracking, motion segmentation, video stabilization, compression, etc. Optical flow is defined as apparent motion of brightness patterns. This definition is important, as uniform, moving objects have an optical flow of zero, while non moving objects with change in lighting can have an optical flow not equal to zero.

We define $I(x, y, t)$ as the brightness of at (x, y) at time t. So the optical flow constraint is given by the equation:

$$\frac{\mathrm{d}I}{\mathrm{d}t} = \frac{\partial I}{\partial x}\frac{\mathrm{d}x}{\mathrm{d}t} + \frac{\partial I}{\partial y}\frac{\mathrm{d}y}{\mathrm{d}t} + \frac{\partial I}{\partial t} = I_x u + I_y v + I_t = 0$$

9.1 Aperture Problem

If we look at the previous equation, we see that we have two unknowns u and v, meaning we have an under constraint problem. This is called the **aperture problem**. The simplest solution to this problem is called the **normal flow**.

$$u_\perp = -\frac{I_t}{|\nabla I|}\frac{\nabla I}{|\nabla I|}$$

9.2 Regularization

Regularization introduces an additional smoothness constraint:

$$e_s = \int\int (u_x^2 + u_y^2) + (v_x^2 + v_y^2) dxdy$$

Beside the optical flow constraint:

$$e_c = \int\int (I_x u + I_y v + I_t)^2 dxdy$$

Now we try to minimize $e_s + \lambda e_c$. This can lead to errors at boundaries as it is the opposite of what we try to enforce with the smoothness term.

9.3 Lucas-Kanade

We introduce the assumption of a single velocity for all pixels within an image patch:

$$E(u, v) = \sum_{x,y\in\Omega} \left(I_x(x, y)u + I_y(x, y)v + I_t\right)^2$$

We now have the constraints:

$$\frac{\mathrm{d}E(u, v)}{\mathrm{u}} = 0 \qquad \frac{\mathrm{d}E(u, v)}{\mathrm{v}} = 0$$

We solve with:

$$\begin{bmatrix} \sum I_x^2 & \sum I_x I_y \\ \sum I_x I_y & \sum I_y^2 \end{bmatrix} \begin{pmatrix} u \\ v \end{pmatrix} = - \begin{pmatrix} \sum I_x I_t \\ \sum I_y I_t \end{pmatrix}$$

This is equivalent to:

$$\mathbf{MU} = \left(\sum \nabla I \nabla I^\top\right) \mathbf{U} = -\sum \nabla I I_t = b$$

The Lucas-Kanade algorithm works by computing $\mathbf{U}$ for at each pixel and then solving $\mathbf{MU} = b$. $\mathbf{M}$ is singular if all gradients point in the same direction, i.e. only normal flow is available.

We can refine our estimates by repeating the process multiple times.

9.4 Coarse to Fine

There are some failure modes to the local gradient method, for one if the intensity structure within our window is poor (uniform area). Another failure mode is when the displacement is too large or the brightness is not constant. To make our approach more robust, we can again use an image pyramid, first estimating the optical flow on a coarse image and then iteratively start using finer images from the pyramid.

9.5 Parametric Motion Models

Global motion models offer more constraint solutions and integration over larger areas. For affine motions (rotation, translation, sheer) we introduce the following model:

$$I_x(a_1 + a_2 x + a_3 y) + I_y(a_4 + a_5 x + a_6 y) + I_t = 0$$

As each pixel provides one constraint in six global unknowns, we need a minimum of six pixels. The error we try to minimize here is again the square loss.

We can change the model further to allow for more types of transformation / warping of the image. There are also model that allow for 3D motion.

9.6 SSD Tracking

For large displacements we can use template matching, we do this by defining a small area around the pixel as the template and match the next image against that template. This will not work for uniform or noisy patches.

10 Video Compression

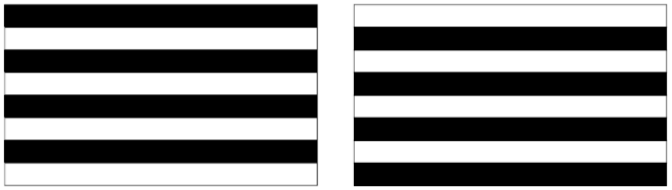
One of the main concepts of video compression is that while the human visual system is specifically sensitive to motion, some distortions are not as perceivable as in static images. Visual perception is limited to < 24Hz, but flicker can be perceived up to > 60Hz.

Bloch's Law

Up to a time frame of 100ms, it does not matter "how" light arrives only the sum, e.g. 10ms of double the intensity is equal to 20ms at the normal intensity.

10.1 Video Format

A video sequence is a bunch of images aligned in a time sequence. The interlaced video format uses two temporal shifted half images and increases the frequency from 25Hz to 50Hz.



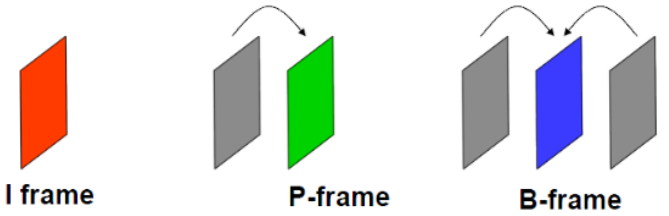
Today this is not done anymore, we rather use a progressive format that updates the whole screen at a time.

10.2 Temporal Redundancy

One way of compressing video is to take advantage of similarity between successive frames. Especially for high frame rates this works well.

The most used representation along the temporal dimension are predictive methods. There we define tree typed of frames:

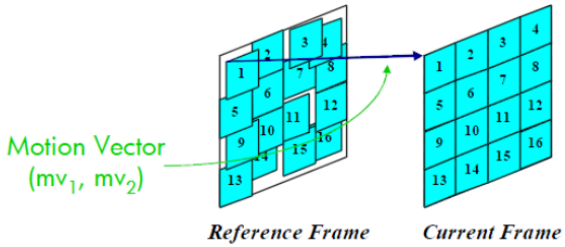
- **I Frame**: Intra-coded frame, independent of all other frames
- **P Frame**: Predictively-coded frame, based on the previous I and P frame
- **B Frame**: Bi-directionally predicted frame, based on both the previous and future I and P frames



P frames can send motion vector plus changes. The frames starting at an I frame until the next I frame is also called a group of pictures (GOP). Temporal redundancy becomes inefficient if there are many scene changes or there is a lot of motion, e.g. confetti.

10.2.1 Motion-Compensation Prediction

One possibility of dealing with a high amount of motion is to use MC-prediction. Ideally we would partition the video into moving objects and describe these object motions. In reality this is very difficult, therefore we partition each frame into blocks, e.g. 16x16 pixels, and try to describe the motion of each block.



The algorithm first divides the frames into blocks and then uses the best matching blocks of the reference frame as prediction of the blocks in the current frame. For the block matching we use some best match metric, e.g. mean squared error or mean average error. The candidate blocks for the best match can either be determined by looking at all blocks or select a subset by some assumptions.

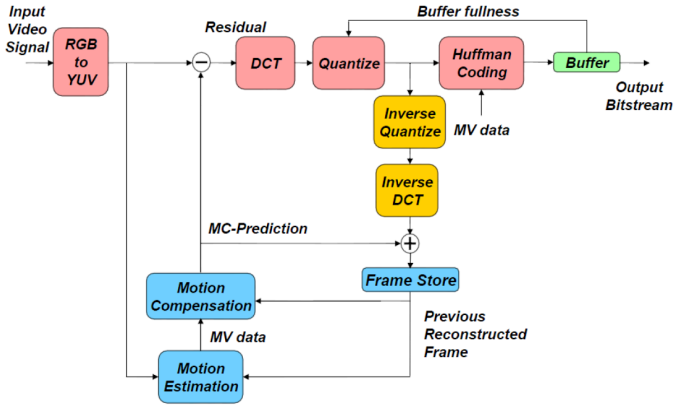
If we combine all the motion vectors, we end up with a **motion field** for all the blocks. As motion is not limited to integer-pixel offsets, one could try to estimate sub-pixel motion by spatial interpolation of the frames. The model of MC-prediction does not work well for more complex motions and might produce blocking artifacts.

10.2.2 Bidirectional MC Prediction

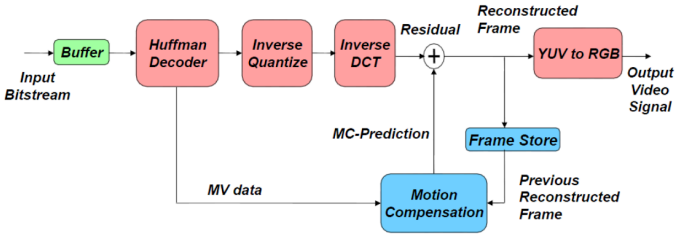
Instead of only looking at the previous frame we also take a look at the next frame. Bidirectional MC-prediction then estimates the position of a block in the current frame from either the previous frame, the next frame or the average of the previous and next frame.

10.3 Video Compression Architecture

Basic video compression architectures use temporal, spatial and color redundancies. Spatial redundancy uses DCT on the blocks and color redundancy does a color space conversion. A basic encoder could look as follows:



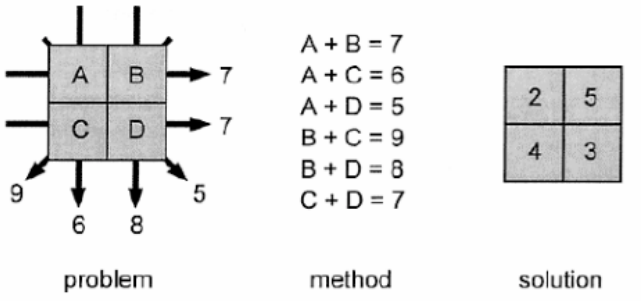
The corresponding decoder then looks like this:



11 Radon Transform

Radon transformation is often used in medical imaging, e.g. computed tomography (CT). In CT data collection works by shooting x-rays through the material we try to image and collect them on the other side. If we do this from multiple angles, we can try to reconstruct an image from the measured data, as different material absorb a different amount of x-rays. We take the logarithm of this value, since absorption is a multiplicative process.

This can be seen as an image reconstruction problem.



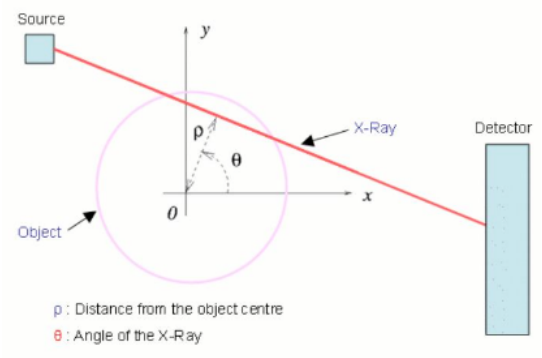
X-rays move along a straight line, at distance s it has intensity $l(s)$ and after traveling δs the intensity is reduced by δl. The reduction depends on the intensity and the optical density $u(s)$ of the material. For small δs it holds $\delta l / l(s) = -u(s)\delta s$. This leads to the following equations:

$$I_{\text{finish}} = I_{\text{start}} \cdot e^{R} \qquad R = \int_L u(s)ds$$

This related to the Radon transform:

$$Rf(L) = \int_L f(x)|dx|$$

Given the following setup:



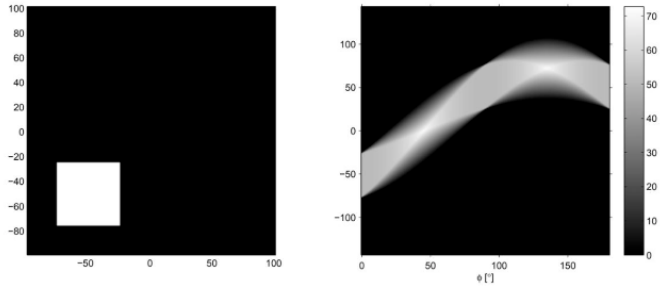
We can calculate:

$$R(p,\theta) = \int_{-\infty}^{\infty}\int_{-\infty}^{\infty} u(x,y)\delta(p - x\cos\theta - y\sin\theta)dxdy$$

The Radon transform has the following properties:

- Linearity
- Shifting only changes the p coordinate
- Rotation of the coordinate system also rotates the Radon transformation
- The Radon transform of a 2D convolution is a 1D convolution of the Radon transformed function with respect to p

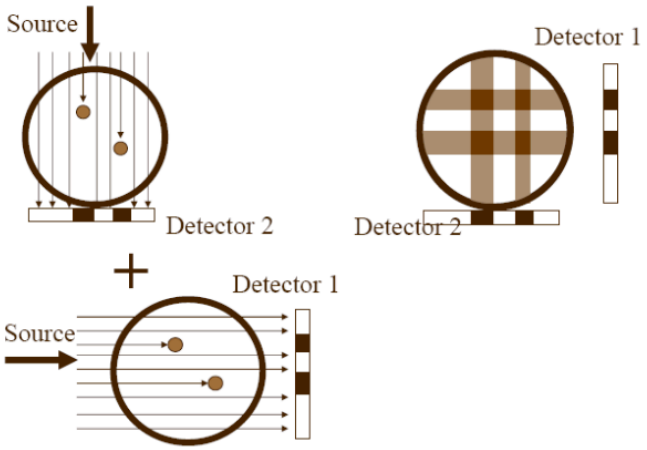
The following image shows the Radon transform of a square. We call this a **sinogram**.



11.1 Backtransformation

Up until know we have seen how to perform a Radon transformation from an image to a sinogram. In reality we want the opposite, as medical imaging devices give us a sinogram and we want to reconstruct the image. This is the same as the question: "Can we find $u(x, y)$ if we know $R(p, \theta)$.

We can compute this with linear algebra, solving the overdetermined system $Kf = g$ using normal equations. This solution is not cheap, therefore we are interested in alternative ways. One possible way of doing this would be backprojection:



If we do this multiple times, adding more projections, we end up with a blurred version of the original image.

11.1.1 Fourier / Central Slice Theorem

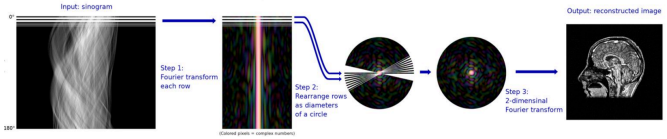
$$G(q,0) = F(q\cos 0, q\sin 0)$$

This tells us that the 1D Fourier transformation of the measurement $g = Rf$ (for a fixed θ) is equal to the 2D Fourier transformation of the object slice $f(x,y)$ evaluated at a particular point. We can apply this for any orientation θ.

11.1.2 Backprojection Algorithm

The backprojection algorithm works as follows, for each of the K projection angles θ:

1. Measure projection data $P_\theta(t)$
2. Fourier transform it to find $S_\theta(w)$
3. Multiply by weighting function $2\pi|w|/K$ (high-pass filter)
4. Sum over the image plane and perform 2D inverse Fourier transform.



There are some practical issues, as it requires many precise measurements and is sensitive to noise it may lead to blurring in the final image.

12 Sparsity

We can use sparsity for image restoration. The idea behind this is, that a sparse signal is good for representing structure but not for white Gaussian noise. For this we model an image as follows:

$$\underbrace{y}_{\text{measured image}} = \underbrace{x}_{\text{original image}} + \underbrace{w}_{\text{noise}}$$

Now we perform MAP (maximum a posteriori) estimation:

$$E(x) = \frac{1}{2}||y - x||_2^2 + \Pr(x)$$

Where Pr is a log prior. Some classical priors to use would be smoothness $(\lambda||\mathcal{L}x||_2^2$ or total variation $\lambda||\nabla x||_1^2$.

Let $\mathbf{D} = [d_1, ..., d_p]$ be a set of normalized basis vectors, we call it **dictonary**. $\mathbf{D}$ is adapted to x if it can represent it with few basic vectors, meaning a sparse vector α exists, so that $x \approx \mathbf{D}\alpha$.

There are predefined dictionaries, but we can also try to learn one ourself. We use the following model:

$$\min_\alpha \frac{1}{2}||x - \mathbf{D}\alpha||_2^2 + \lambda\psi(\alpha)$$

Here ψ induces sparsity and can be the $l_0, l_1, ...$ norm (we call it Lasso if l_1 is used). This idea can be expanded to not only perform denoising, but also do inpainting and demosaicking.

13 Texture

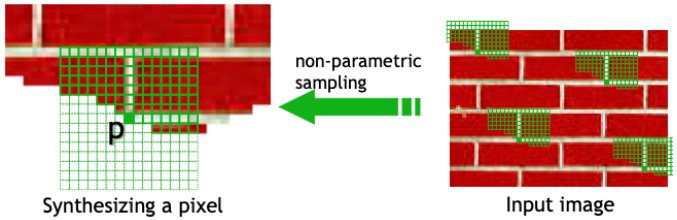
The key issues for textures are analysis / segmentation, representing the texture, and synthesis, generating textures.

13.1 Representing Textures

Textures are made up of stylised subelements, repeated in meaningful ways. To represent a texture we can try to find these subelements and represent their statistics. Finding the subelements can be done by applying filters and looking at the magnitude of the response. Possible filters could be spots and oriented bars at a variety of different scales (image pyramids).

13.2 Texture Synthesis

There is a variety of approaches to texture synthesis. One might use a histogram to capture the intensity probability distribution, but this does not capture any spatial relations. To improve on that one might use a co-occurrence matrix, having the probability distributions for intensity pairs. Other approaches are image-base, trying to perform "cut and paste". This is done by assuming Markov property and computing $P(p|N(p))$ (N is the neighborhood of p).



To synthesize p, just pick one match at random.