Multimedia Systems

Algorithms and structures

Lecture notes (2022)
Luka Čehovin Zajc

Material provided as is, only basic spell-checking was performed, some factual errors may also be lurking around (you are encouraged to report them).

Introduction

Source material: Li and Drew, Fundamentals of Multimedia, Chapter 1, 1.1 - 1.2

The word “multimedia” has origins in latin words “multum”, meaning **many** and “medium”, originally meaning “center, middle”, but later also used for “**channel for communication**”. In modern world the term “multimedia” is used in various contexts:

- Computer salesman: “This is a multimedia computer!”
- Entertainment industry - multimedia performance/experience, video-on-demand
- **Computer science student** - application that uses **multiple modalities** + interactivity

All of the claims above are essentially correct at different levels because multimedia is an interdisciplinary field that is defined by convergence:

- **Convergence** of research and development fields that have been separated in the past.
- **Convergence** of technologies/hardware (PC, tablet, phone) into a unified product

The types of content that are most frequently associated with multimedia are **video** and **sound**, but the list also includes still **images**, **text**, **animation**, as well as the option of having the content delivered in an **interactive** manner. Less frequently, the term is associated with other human senses, such as touch (**haptics**) or smell.

Perhaps the most well known multimedia application nowadays is **hypermedia**, an extension of **hypertext** term, coined by Ted Nelson in 1965 which uses a **non-linear text** (multiple documents, connected by links) together with images, video and sound that we know as the **World-Wide-Web**.

In terms of research we have three main branches, all of them are very interdisciplinary in their own research direction:

- Multimedia **processing** and **storage**: content acquisition, analysis, compression, security
- Multimedia **tools**, **applications**: manipulation, interactivity, user interfaces, collaboration
- System **support** and **networking**: content delivery, quality of service, networks, storage

We will talk mostly about the first branch and a bit about the second one.

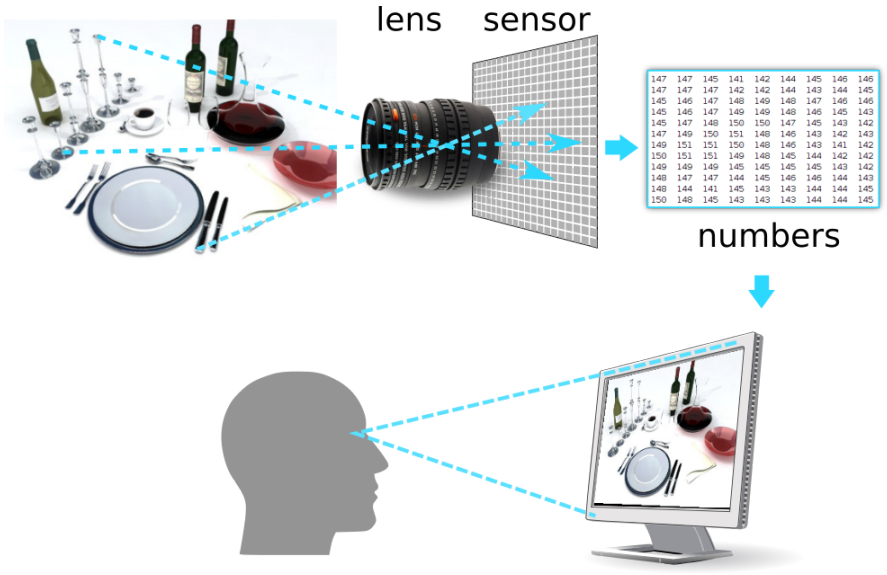
Images

This section of class is dedicated to **digital images**. Digitalization is one of the key aspects of multimedia as it allows us to process information using (digital) computers. We must also be able to present digitized visual information back to users in an understandable way, therefore we will first talk about image acquisition or formation.

Image acquisition

Source material: Li and Drew, Fundamentals of Multimedia, Chapter 4

Image is essentially a **snapshot** of **light** that passes a given point in space and falls to an image or sensor plane in a given **interval** in time. If we want to acquire this light in a digital form we have to **digitize** it, convert it to a **matrix of values** that denote how much of light has fallen to a specific region. We can then display this information back to the user as is or processed in the form of a digital image on a digital screen.



The entire process is a bit more complicated. To understand it we have to start with light and how we humans perceive it.

Human perception of light

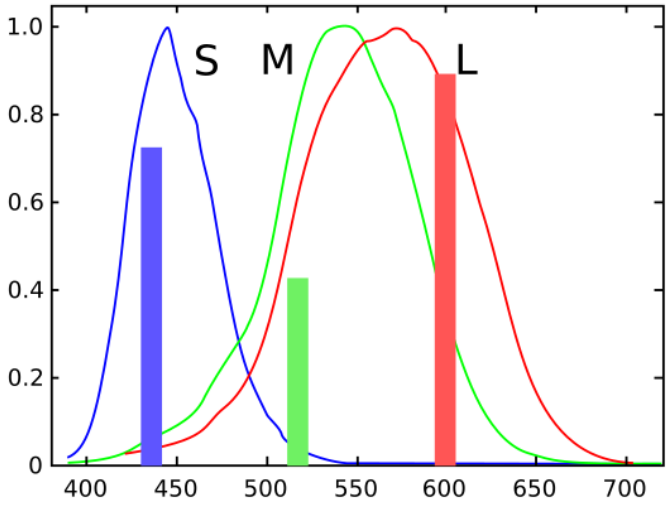
Light is an **electromagnetic wave**, it is also made of **photons**. Different wavelengths are perceived as different colors. Most light that we see is a mixture of wavelengths, but a laser is a light with a single wavelength. We do not see the entire spectrum, only waves in range from **400** to **700** nm, below is infrared, then heat, above is ultraviolet.

Our eyes that detect the light are not all that different from our cameras, there is a lens and a “sensor” called **retina** composed of two types of cells that are sensitive to different wavelengths of light:

- **Cones** - perceive higher intensity, sensitive to colors
- **Rods** - perceive low intensity, not good with color, there are more rods in an eye

We are called **trichromatic** because our cones are sensitive to **three** different wavelength spectra that roughly correspond to red, green and blue color (at least the peaks of sensitivity). There are three types of rod cells with different response spectrum. It is not yet entirely clear how these responses are then combined into the final color, but the neurons seem to be sensitive to differences between their three primaries. All three channels are also combined into achromatic information.

The eye is the most sensitive in the middle of the light spectrum, the distribution of rod cells is 40:20:1 (RGB)



These are the **sensitivity curves** for cone sensitivity, the response of each cone neuron is equal to the integral of the product of the light spectrum and sensitivity curve. To achieve the

same perceived light we do not have to reproduce its complete spectrum, we only have to stimulate the cones to match the responses. We call this effect **metamerism**, which means that two light sources with different spectrums are perceived as of equal color because they stimulate the cone cells in the same way.

If we want to **simulate** a color we have to **stimulate** cone receptors. We do this by producing a light that matches their **high-sensitivity** interval. We call the lights **color primaries** and they differ in different color system standards. This means that different combinations of values in different standards may be perceived as the same color. The side effect of this is that exact color reproduction between two standards (even devices) may require additional calibration.

The standard experiment to establish a relationship between primaries and perceived color is by a **tristimulus colorimeter** experiment. A person is asked to reproduce a **reference color** by controlling the intensity of the **three primary colors**. Since some colors cannot be reproduced due to the overlap of the cone type response curves, a light has to be subtracted from the reference color, we call this **negative light**. Since the color perception changes with the field of view due to non-uniform distribution of cone receptors, a **standard observer** is introduced to eliminate this variable. The standard observer specifies what is the angle of observation, i.e. the widely used CIE 1931 2° Standard Observer specifies a maximum of 2 degree angle of observation.

This is how the **CIE sensitivity curves** were established by the International Committee on Illumination (CIE); these curves are the first **quantitative link** between human color perception and visible spectrum wavelengths. The values were determined with a colorimeter experiment with a negative light with three color primaries, R/S (short), G/M (medium), and B/L (long).

Since responses to the CIE SML/RGB primaries have negative values (people were not able to reproduce it with three primaries), it had to be transformed with a linear transformation to match the overall sensitivity. The non-negative curves, denoted as X, Y and Z are used to reproduce an arbitrary color visible to an average human. We call this the **CIE XYZ** color space. The three components, X, Y, and Z are defined based on human perception - when judging the relative luminance (brightness) of different colors in well-lit situations, humans tend to perceive light within the green parts of the spectrum as brighter, therefore the Y curve is denoted as luminance and roughly matches the curve of the G/M curve. The X roughly matches the R/S curve and Z the L/B curve.

The CIE XYZ color space can be visualized with the **CIE xy chromatic diagram**, it is a **2D representation** of all colors that are visible to an average human in terms of chroma. The diagram visualizes the CIE xyY color space, a transformation from the CIE XYZ where the first two components are normalized X and Y components and the third component (Y) is fixed, usually to the maximum value. The colors in the diagram are more saturated on the outside, the brightness component is not visualized. The colors outside of the locus are called imaginary colors (some can be seen by dogs, bees, “non-standard” humans).

Digital camera

A digital camera also has a sensor, but its composition is a bit different. As we know, a digital image is a proper 2D array of values/pixels, but in many cases the technology behind getting these values is not as simple. It is technically challenging to measure three color components at the same spot, so many **sensor layouts** were invented that simulate this with **post-processing**. The most well known is the **Bayer pattern** that has twice as many green pixel sensors (chroma + luminosity) as red and blue (chromatic components). If you want to get a full image with colors for every pixel, you have to **interpolate** red, green and blue values for missing pixels. This leads to some artifacts where colors change rapidly. A recent alternative is a **Foveon X3** sensor that can retrieve all three components at every pixel sensor because of a completely different sensor structure where sensors are placed one upon the other.

CIE XYZ color space is not used in technology because of **cost** and **technical** issues. **Different standards** are used for different purposes, they can reproduce different **subsets** of visible colors. Based on the medium used we classify color models into **additive** or **subtractive**.

- **Additive** models start with black, then colors are added by mixing primaries. Most digital devices use this approach, e.g. monitors, TVs, projectors. The most known additive color space is **RGB**.
- **Subtractive** models start with white color and then add components that absorb light with a particular wavelength, pigments. They are primarily used in printing and other analogue image recreations (photography, crayons). In printing the color space that is most known is **CMY** or **CMYK**. K stands for black pigment because true black is hard to achieve just by mixing CMY components.

The RGB color space is perhaps the most known because it is used in display hardware. It's foundations are in **cathode television**. There are different RGB standards with **different color primaries** and therefore cover different subsets of the CIE XYZ color space.

The coverage of CIE XYZ color space differs between different RGB and CMYK. This means that some **loss** can be caused when **converting** between spaces (some colors cannot be reproduced, arithmetic loss).

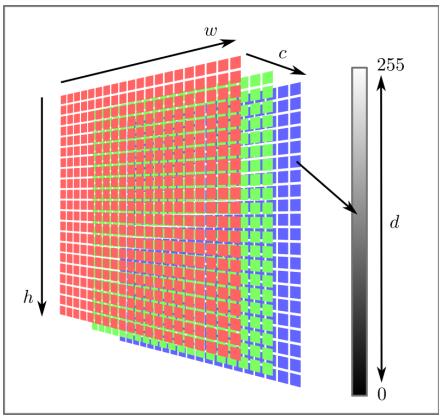
Besides RGB we also know other color spaces, these are primarily **re-interpretations** or **re-projections** of the RGB color space with a specific purpose and have a 1:1 mapping of colors (if we do not consider rounding during conversion).

- CIE Lab color space mimics color perception, this means that similar colors are close in the color space. It has a separate **luminosity** component and **two chroma** components. The space is represented as a cylinder.
- The HSV color space has a more psychological motivation, it separates color into **hue**, **saturation** and **value** so it is understandable what you will get if you change one of the components.

Image processing

Source material: Gonzalez and Woods: Digital Image Processing

With an image acquired we may want to process it, change it in a meaningful way. This requires knowledge of some **image processing algorithms**. The core concept in image processing is a **digital image** which is represented as a **multi-channel 2D matrix**, or a **3D matrix**. The first two dimensions denote **width** and **height** while the number of channels or the third dimension denotes the **individual color components** (e.g. red, green and blue). Another important aspect is the **sampling resolution** - how many colors can we represent with a single value. The most frequent size in this case is **8-bit**, we can represent values from **0** to **255**, but other sizes are also common (e.g. 16 bit).



The other view on the image is a signal-processing one, an image is a **discrete function** from two dimensional space into a 1 or 3 dimensional one. This view allows us to operate with mathematical operations, such as convolution

Image processing is a very broad field that mostly includes low-level pixel operations, but also high level algorithms that lead to computer vision and computational photography. Some concepts are frequently used in multimedia. The simplest class of operations is called **per-pixel operations** because they operate on each pixel independently.

- **Grayscale** - many image processing techniques work only on single channel images, to obtain a single channel image from a color one, we have to convert it to grayscale - most commonly by averaging the three channels and with that approximating luminosity information. This works for the RGB color space, not for HSV, in that case we can only use the value component because luminosity is already separated.
- **Inversion** - Inversion means that we “invert” every value in an image by subtracting the pixel value from its potential maximum value. In the case of 8 bit images this means 255.

- **Thresholding** - Thresholding compares each pixel to a value called threshold. There can be many outcomes of the comparison, but generally the result is different if the value of the pixel is greater or lower than the threshold.
- **Brightness** change - Brightness is intensity of a pixel relative to another pixel. Changing brightness means offsetting all pixels by a given value
- **Contrast** change - Contrast is a difference between minimum and maximum pixel. Changing contrast means scaling all values by a given scaling factor.
- **Non-linear mapping** - Values can be also mapped with a nonlinear function. This allows us to, for example, change the contrast of the image.

Histogram

Histogram is a statistical structure that describes the **distribution of values**. In image processing we use it to see how many times a certain value appears in an image. An image histogram is constructed by counting individual values or intervals of values, the counters are known as **cells** or buckets. We can use this information to **describe the content** of an image in a way that is **invariant** to the position of individual images and thus also rotation, translation or scale.

Histograms can tell us if the image quality is poor, e.g. if only an interval of all pixel values is taken, which means that the image probably has low contrast. The simplest way of adjusting the contrast is by "**stretching**" the histogram of an image. The operation does not actually work on a histogram, but is again a per-pixel operation that is dependent on the minimum and maximum value in the image (which can be obtained from an image directly, but also from a histogram).

A more complex contrast adjustment technique is called **histogram equalization**. In this case we want to convert the distribution of values in an image in a non-linear manner so that the resulting histogram will be (approximately) **uniform**. This is not always possible since we cannot change half of the pixels with the same value differently than another half.

Applying contrast adjustment operations to color images has to be done correctly, otherwise the results may not be desired. Since contrast is a luminosity property, we have to first transform the image into a color space where luminosity is separated in its own channel, adjust it, and transform the image back, leaving the chroma channels intact.

We will talk about histograms in the future, for now let's just look at how we can describe **color with a histogram**. Since color images have multiple channels, we have to take this into account.

- **3 histograms** - Each channel is treated separately, producing three histograms. This way the joint information is not included.

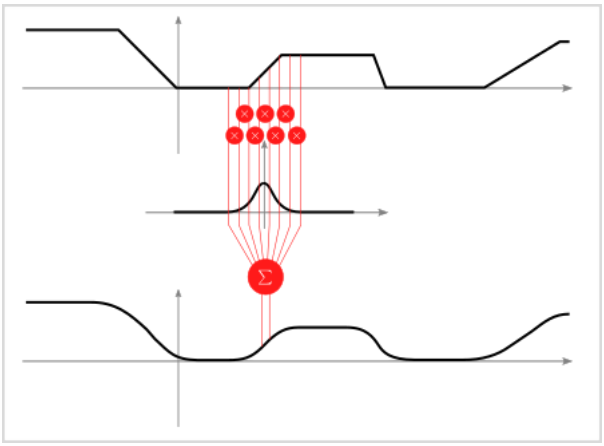
- **3D histogram** - Each channel is a dimension in a 3D index into a 3D histogram. This way joint information is preserved, but this histogram uses more space (for equal size bins).

Image filtering

Image filtering is a class of operations where the result of a **single pixel** depends on the **values of its neighborhood** in the input image. We know two types of filters:

- **Linear filters** - An operation over the source pixel values is a linear operation (weighted sum). This enables us to use the mathematical concept of convolution which is an associative and in some cases separable operation to speed up computation.

- **Non-linear filters** - An operation is an arbitrary nonlinear operation on pixels, e.g. max, min, median. These kinds of filters do not have nice properties like separability and associativity.



A linear filter can be interpreted as a **weighted sum**. What therefore defines the operation are the weights in a matrix that is called a **kernel**. We multiply corresponding pixels together and sum up the numbers to get a filter response for a pixel, we do this for all the pixels. Some examples of frequently used kernels:

- **Uniform** kernel - all weights are equal, smoothing (averaging)
- **Identity** kernel - only copies original center value, image is preserved
- **Shift** kernel - only shifts image in one direction
- **Gaussian** kernel - achieves better smoothing effect without artifacts, low-pass filter

The important property of kernels in image processing is that the sum of all their elements is 1, otherwise the result will be scaled up or down.

Convolution is defined on an infinite interval, so another question in case of finite images is what to do when we come to a **border**. There are multiple strategies (names of these strategies are not standardized):

- **Crop** border - resulting image is smaller than original
- **Constant** value - missing values are replaced with a constant, produces a vignetting effect
- **Edge** values - copy border values
- **Warp** - copy values from the other side of the image
- **Reflect** - mirror image values

Some other operations that can be done with linear filters:

- **Detecting edges** - image is a discrete signal that can be derived, this means that we get high values where the image value changes significantly. Since an image is a 2D signal we have to compute partial derivatives by convolution and computing magnitude.
- **Sharpening** - We can amplify high frequencies in an image by first removing low frequencies (blurred image) and combining low and high frequencies back together (weighted).

The most well known **nonlinear filters** are:

- **Median**: take median value in the neighborhood
- **Max**: take maximum value in the neighborhood
- **Min**: take minimum value in the neighborhood
- **Bilateral** filter: weighted sum where weights are re-calculated for each neighborhood based on intensity similarity with the center pixel. Preserves edges because only values that are close together are smoothed.

One simple use case for image filters is noise removal. Noise is a random signal added to the image information. The two most common noise profiles are **Gaussian** noise and **salt-and-pepper** noise.

In case of Gaussian noise the per-pixel values are sampled from a zero-mean Gaussian distribution. Since the error for each pixel is sampled independently, this results in a high-frequency signal addition. A Gaussian filter is a low-pass filter, it removes the high frequencies from the signal. However, an image can also contain high frequencies by itself and a filtering with a Gaussian filter will result in a blurred image. This is where a bilateral filter is useful since it preserves edges (in most cases). In case of salt-and-pepper noise, median filter is more effective: large deviations in a local neighborhood are filtered out.

Salt-and pepper

Image transformations

Pixel-wise and filtering operations are intensity operations. Geometry information, on the other hand, changes the geometry of the image, they **move pixels around** and modify the dimensions.

The geometric transformations are **parametric**, meaning that all pixel locations are transformed in the same manner with a parameterized transformation function or **non-parametric**, where the transformation cannot be described by a parameterized function and only exist as a pixel-to-pixel mapping. We will only cover parametric transformations.

Parametric transformations can be **linear**, meaning that we can describe them with a projection matrix or nonlinear. There are several types of basic linear transformations that can be arbitrarily combined by multiplying their matrices:

- Translation
- Rotation - rotation around origin
- Scaling - either uniform or non-uniform
- Euclidean - translation + rotation
- Similarity - translation + rotation + uniform scaling
- Affine - translation, rotation, scaling, shear (preserves parallel lines)
- Projective / homography - projects points from one surface in 3D space to another

Linear transformations are usually specified with a matrix that transforms points in **homogeneous coordinates**. This means that a 2D position is specified as a 3D vector where the last coordinate is always 1. A 2D linear transformation matrix is therefore also a 3x3 matrix.

Warping is the process of applying a geometric transformation to a digital image. The naive process is simple, for every pixel in the original image compute the location in the transformed image based on transformation and **copy** the value there. But there is a problem with this approach, we visit all pixels in the source image, but we do not visit all pixels in the destination image. This means that some pixels in the destination image remain **unassigned**, while others get **overwritten** multiple times.

To avoid the problem we have to ensure that we visit all pixels in the destination image exactly once. This means that we have to reverse the process and calculate **inverse transformation** for all pixels in the destination image to their corresponding source pixel. Still, we encounter a problem that the location of where the color in the source image should be taken is not a coordinate of a single pixel, but a real value somewhere between multiple pixels. The simplest strategy for determining the value is by using the closest pixel, i.e. finding the **nearest neighbor**. This simple strategy is fast, but produces an **aliasing** effect. The interpolation effects are most visible when we are **resizing** an image, i.e. increasing the resolution. In case we are increasing the size of an image, the need for interpolation is clear, we want more data than it is

available. Better strategies involve **interpolation**, using multiple neighborhood pixels to determine the color based on their proximity. It is easiest to first look at the interpolation in one dimension. A **nearest neighborhood** strategy takes the closest neighbor value. We can use **linear interpolation** that fits a linear function to the two neighbors and samples it at the desired location. The **cubic interpolation** does a similar thing, but uses four points to fit a third-degree polynomial. In a digital image, we have two dimensional signals, therefore the interpolation strategies have to be adapted. We still have the nearest neighbor strategy for four pixels, we have **bilinear interpolation** that uses four pixels to fit a plane and we have **bicubic interpolation** that uses sixteen pixels to fit a polynomial surface.

Lanczos sampling is used when resizing or rotating an image, it is used to smoothly interpolate between values, avoiding aliasing. It is done by convolution of the signal with a Lanczos kernel, a finite approximation of the Sinc kernel that is theoretically the best low-pass filter. Lanczos sampling is also the slowest among the mentioned approaches.

In the case of **decimation** (reducing the size of an image), we also have to take special care. If the details in the image are small (e.g. thin white lines on black background), they will get lost when using a naive approach, e.g. nearest neighbor because we are essentially sampling the image with a too low sampling rate (Nyquist–Shannon sampling theorem). To produce a result that contains at least some of the details, we have to first remove high frequencies (e.g. by convolution with a Gaussian kernel) and then use nearest neighbor sampling.

Non-linear transformations are image transformations where the mapping between pixels is not determined by a linear function. There are multiple options, e.g. in **camera image rectification** we try to remove the effect of a lens distortion. For this we have to know the lens distortion model that we obtain with camera calibration. This model is not linear. Another use-case is to use **locally-linear transformation** to perform **image morphing**.

Image morphing is a process of gradually changing one image into another. The simplest way to do this is by **weighted averaging** of corresponding pixels with gradually changing weights, however, this does not really produce good results, because it is not **content sensitive**. We can obtain a more realistic effect if we gradually **geometrically transform** one image into the other. The transformation that we are trying to achieve is non-parametric, specified by a **dense deformation field**. The field tells us where every pixel in the image moves to. Of course specifying the entire field is time-consuming and not very intuitive. We can achieve a good approximation using locally linear transformation on a set of **correspondences** or **control points**, those are pairs of pixels that have to move one into another. The transformation for other pixels can be computed either by **linear interpolation** or by dividing image into triangular mesh using **Delaunay triangulation** algorithm and then compute **affine transformation** for all triangles.

The image morphing algorithm is the following:

- Input: two images, A and B, correspondence pairs

- Repeat for time steps t = 0 … 1
- Interpolate correspondence to position t
- Warp image A from 0 to t and image B to t from 1
- Blend warped images using factor t

Content-aware image resizing

If we want to resize an image, the traditional approach using linear transformation will modify values of all the pixels. In some more advanced cases this would want to be avoided, e.g. there are some regions in the image that are **semantically** more relevant. Content-aware resizing will change the size of an image while preserving important structures and reducing unrealistic artifacts. Knowing what is important is a weakly conditioned problem and is in the scope of multimedia connected to human perception. A fast approximation of importance are **image edges**, fast changes in image intensity. Using first order derivatives, we can estimate the energy or importance of each pixel.

In a simple case that can be generalized easily we want to remove unimportant pixels in a single direction (reduce width or height). To preserve structure we have to use an algorithm called **seam carving** that is based on dynamic programming to determine a path of pixels from one side of an image to the other that has the lowest cumulative energy. A path or a seam is determined by greedily computing the minimum possible path for each end pixel and then backtracking from the pixel with minimum cumulative energy back to the source.

Image merging

Image morphing is a good introduction to **image merging**, we are back to per-pixel operations where we would like to combine two images, taking some regions from one image and some from another, or merge the color of some pixels with different weights. When merging color images we are simply applying the merging operation to all color channels individually.

The simplest form of image merging is merging with a **binary mask**. We have two images and a mask, all of the same size. The mask only has zero and non-zero values that signify which image should the corresponding pixel value be taken from.

Binary masks produce unrealistic results in many cases, it is hard to define a clear pixel-wise boundary between objects. **Alpha blending** is an extension of the binary mask approach where images are blended using an alpha mask. In this case values between 0 and 1 in the alpha channel signify the degree of pixel value from the first and the second image taken when computing the value of the corresponding pixel. Alpha blending helps us when merging images with the same content, but different contrast, e.g. when merging well aligned panorama images we can ensure that the contrast change is not too sharp.

Alpha blending can produce smooth transitions, but the **degree of smoothing** is hard to define for a general use case. If the transitions in the mask are too sharp the merging becomes a **binary cutoff**, if they are too smooth, we end up with a **ghosting effect**.
The problem with a smooth alpha mask is that not all image frequencies respond to it equally. It is much more natural to merge low-frequencies with a smooth mask, but high image frequencies should be blended in sharper transitions, otherwise they get lost. This is how **frequency-aware blending** works. It decomposes the image into several frequency bands and blends each of them with its own alpha mask. Then all the bands are combined to form the final image.

To decompose an image into bands we have to use image pyramids. They are multi-scale image representations. **The Gaussian pyramid** is used to decompose the image into layers where each higher layer only includes lower frequencies, with a specific frequency band removed. This is achieved by convolving lower layer images with a **Gaussian** (low pass) **filter**.

The other pyramid that we have to know is the **Laplacian pyramid**. This pyramid is obtained from the Gaussian pyramid by **subtracting** subsequent layers, except for the last layer that is the same as in the Gaussian pyramid. This means that each layer will only contain the frequencies that are included in the lower layer and not in the one above it. A nice property of the Laplacian pyramid is that we can **reconstruct** the original image if we sum together all layers, all frequency bands, together.

The frequency-aware blending algorithm is as follows:

- Input: Two images and a (binary) mask
- Do the following steps for all color channels:
    - Construct Laplacian pyramids for both images and a Gaussian pyramid for the mask
    - Combine corresponding image pyramid layers with layers from the mask pyramid
    - Collapse resulting merged Laplacian pyramid into the merged image channel

Image segmentation

Image merging assumes that the mask of the foreground is known in advance. Sometimes this is not true and determining a good segmentation mask is time consuming. A semi-automatic approach that requires coarse input and produces a detailed mask of the desired object is called **GrabCut**. It is based on the observation that segmentation labels are highly structured. This means that two pixels that are similar in color (data similarity) and/or near one another (smoothness) are also likely to share segmentation labels.

The problem of segmentation in GrabCut is formalized using the **Markov random field**. It uses Gaussian Mixture Model to model a generalized color probability and Graph Cut to improve segmentation in iterations. The initial segmentation has to be provided by an outside source.

The iterative algorithm uses a current mask to build models of foreground and background. Then, it estimates the likelihood that each pixel belongs to each model using Bayes' theorem. This probability serves as the data similarity estimate, the smoothness is usually determined using image derivatives. Using these estimates, a Graph cut algorithm is used to determine **maximum flow** through a network of pixels. The resulting cut determines a new segmentation estimate which is used in a new iteration. The process is repeated until the changes are not noticeable (convergence) or for a limited number of times.

Video

Digital video is in a nutshell a sequence of digital images that can be displayed one after another with sufficient speed to create an illusion of motion. Video is frequently accompanied by sound, therefore we can say that video is a digital medium for the recording, copying, playback, broadcasting, and display of moving visual (and audio) media.

Acquisition and reproduction

Source material: Li and Drew, Fundamentals of Multimedia, Chapter 5, 5.1 - 5.3

While digital video acquisition is similar to image acquisition, the display of video requires some technical details. Human perception can perceive about 10 to 12 individual images in a second as individual images, for rates that go above 20 frames per second individual frames are integrated into motion. We call this **persistence of vision** and it is related to how long the image stays in the visual cortex, for how long the neurons stay saturated.

The first video projection technologies were film projectors, they displayed individual images by illuminating a frame from a **film reel** for a fraction of a second and then moving to another image, with **shutter** closed in between. Without a shutter the images would move in a blur. Original projectors used display rates at about 25 frames per second, which should be enough for an illusion of motion, however, the rapid changes between bright image and dark period between the changes introduced a **flickering** effect. It turns out that the brighter the images, the shorter is the interval required for persistence of vision effect. Since the time without an image is also longer than the period of persistence of vision, it is registered by the brain. This is why the frame rates of displays were increased, showing a single image multiple times for shorter intervals. This way the image was less bright, with the same light source, but the projection did not flicker.

The other well known technology that has been used for decades is **cathode** television. In this case beams of electrons are flying in a **vacuum tube** and hitting **phosphorus** particles on the other side that glow when excited (**fluorescence**). In color television there are groups of three types of phosphorus that glow in red, green and blue (color primaries). To maintain an illusion of motion, the image has to be refreshed fast enough. Typical refresh rates are **50 or 60 Hz**, which

should be enough for most people, but since some individuals still perceive flicker at this rate, there exist also **75 or 100 Hz** systems, especially as computer monitors. The reason for flicker is similar to projectors, the phosphorus quickly starts losing its brightness before it is excited again.

The cathode displays are now more or less obsolete, replaced by flat-panel technologies, most prominently the **LCD displays**. The principle of these displays is that the backlight is blocked or passed through certain liquid crystal pixels and is then filtered through color filters to produce pixel colors. These displays became thiner and thiner as the backlight technology became smaller, e.g. using **LED lights**. The liquid crystals do not flicker themselves, older LCD displays flickered because of the fluorescent backlights.

Video interlacing

The **interlacing** in video means that the **odd** and **even** lines of a frame are taken at slightly different times. This approach was used to **double the perceived framerate** at no cost of bandwidth, but was also used in CRT displays to **reduce flickering**. The downside of interlaced video is that half of the data is really missing, which introduces artifacts (**combing effect**) when a video contains fast horizontal movement.
The opposite option is to use **progressive encoding**, which essentially means that each frame is encoded in its full resolution, doubling the video size at the same framerate. Despite being introduced during the analog television era, many digital video formats also support it, although support is slowly being dropped (e.g. HEVC family of codecs does not support interlacing anymore).

Color in video

In analog video color information can be encoded in two ways. In **component video**, each color channel is transmitted over separate lines with separate signals. This reduces channel cross-talk and therefore results in better image quality, but requires more bandwidth.

Traditionally, color in video was encoded as a separate component, next to the luminosity. All components were mixed in a common signal. This approach is called **composite video** and is suitable for lower bandwidths.

In digital video the separation of color information is also used to preserve bandwidth or space. Although chroma separation formats were already known in analog video, e.g. YUV and YIQ an alternative format, called **YCrCb,** is used in digital video. The main reason for reduced bitrate is that we can **subsample chroma channels** without a noticeable degradation in image quality. This is due to the nature of human sight, we are less sensitive to changes in chromaticity than changes in luminance. A standard notation for types of subsampling is J:A:B

- J - horizontal sampling frequency, number of luminance values in a row
- A - number of chroma values in odd rows
- B - number of chroma values in even rows

Television formats

Video is strongly related to television, therefore we should say a few words about television formats. In the analog era, several analog formats were preferred in different regions of the world. **PAL** in Europe, **NTSC** in North America, **SECAM** in Russia. They are a bit different, but also quite similar in terms of resolution and other characteristics. PAL, for example, was developed to address problems that NTSC would have in Europe because of more difficult terrain and unpredictable weather.

Nowadays, a lot of countries have transitioned to digital television. The advantages of digital video are numerous, direct manipulation, ability to include video in applications, easier access, better error correction, no degradation in quality when copying. There are also multiple standards in this case, e.g. DVB-T, ATSC, …

3D Video

3D video has its roots in **stereoscopic photography**, in both cases the goal is to create an **illusion of depth** by presenting the eyes with two slightly different images of the same scene. The images are taken at two different positions approximately 50 mm to 75 mm apart. The technology of presenting stereoscopic video has been developing in different directions, either as a **wearable device** or using **autostereoscopic** technology.

There are different 3D video wearable display devices available. The most straightforward is a **binocular HMD**, two displays mounted on a head. Other approaches rely on a modified version of a video that contains both images and a wearable filter that separates the images and presents them to the corresponding eye. The most simple technology is **anaglyph**, images are coded with complementary colors (e.g. cyan and red). The glasses have lenses that filter out these colors. Another, more expensive option is using **polarized** light and lenses that pass through only light of horizontal or vertical polarity. **Active shutter** glasses are synchronized with a high frame-rate projector and shutter individual eye when an image for the opposite eye is shown.

The autostereoscopic devices include **eye or head tracking** using a camera and adjusting display or a **parallax barrier** or a **lenticular lens** in front of a display that results in showing a different image at different viewpoints.

The issues that the 3D video is facing are either technological (ensuring resolution and frame rate, minimize cross-talk) with respect to the **limited added value** and increased cost of acquisition and presentation. There have also been numerous **health concerns**, e.g. motion sickness, headaches, nausea, disorientation observed when people have been using 3D video technology for longer periods of time.

Omnidirectional video

One format of recording video that is becoming more and more popular is omnidirectional video, also known as **360 degree video**. Omnidirectional video is recorded using an array of cameras or using special mirrors. Acquired images are then projected to a common image plane using one of the projections:

- **Flat** - only used for 180 degree projections
- **Equirectangular** - full sphere, data redundancy at the poles
- **Stereograph** - “tiny planet” effect
- **Cubemap** - useful for texture mapping, better pixel utilization than equirectangular projection

Omnidirectional videos can be converted to normal video at the desired camera angle using projective geometry. The limitations are the **resolution** of the original omnidirectional video and the **camera view angle** that should not be too wide, in this case the distortion at the edges becomes too noticeable.

Omnidirectional video has a lot of potential for interactive viewing experiences in entertainment, sports, and virtual tourism.

Video stabilization

The need for video stabilization is caused by the fact that a lot of video is recorded with unwanted motion of the camera (shaking) or is unable to follow an object in a smooth manner. There are many techniques for stabilization during video acquisition, we call those approaches **mechanical stabilization**. A camera can have a lens that detects unwanted vibrations and moves to counteract them or shifts the sensor, we call this **optical stabilization**. We also have special equipment like Steadicam or tripods to make camera motion more smooth or stable.

We can also stabilize video after it is acquired, we call these approaches **digital stabilization**, it is a post-processing step to improve the quality of the video. Individual images may be **geometrically transformed** to fulfill certain requirements and filters can be applied to reduce motion blur. We know two types of digital stabilization, **object-centric**, or **local stabilization**, where the camera is making a manually selected object or a point stationary in a video, and **global stabilization** where the overall movement of the camera is made smooth.

Stabilization by alignment

The simplest approach to global stabilization is to match individual frames and **shift / translate** one of them to match the other one as much as possible. This does not work if the scene changes too much or if the motion is not only translative.

Stabilization by feature tracking

Similar effect can be obtained by using a specific point, a **feature point**, in the image and tracking it over multiple frames. The change in point's position can be used to negate the translation and keep the point stationary. Feature points are tracked using feature point trackers. A commonly used feature point tracker uses **normalized cross correlation** to find the best match for a patch around feature point in the following frames. This approach is robust to changes in illumination because the similarity function is normalized to the value variations within the patches. Feature point trackers cannot be positioned on arbitrary locations, usually corners or blobs work best, the feature also has to be visible for the entire duration of the stabilization. **Locality of motion** helps too, if multiple candidates are detected in a new frame, the one closest to the previous position of the feature point is usually taken.

More points can be added to get a more complex linear transformation. With two feature points, **rotation** or **scale** can be estimated and stabilized. With three points **affine transformation** can be modeled. Even more complex linear transformations, such as **perspective transforms**, assume that the content of the image is planar, this can destroy the **illusion of depth**.

Stabilization with keypoints

Feature points have to be manually initialized, but we can achieve a similar effect automatically by detecting **keypoints** in every image and matching them across image pairs. There are multiple algorithms for keypoint detection and description of their neighborhood, the most known being **SIFT** (Scale Invariant Feature Transform). Despite additional precautions that we can take, this approach does not guarantee that all matches will be correct. Therefore, a robust fitting algorithm, such as **RANSAC** (RAndom SAmple Consensus) has to be used to fit a reasonable transform to each set of correspondences.

Stabilization with optical flow

Another option for getting even more correspondences between two frames is **optical flow**. Optical flow algorithms estimate dense **pixel motion** between two consecutive frames. Optical flow cannot be estimated for all pixels, some areas disappear or appear during two frames, but generally the optical flow yields a much denser vector field than keypoints. The density depends on the algorithm used, the well known algorithm by **Lucas and Kanade** is fast, but cannot provide a globally optimal solution (some regions do not contain enough information in them for estimation of optical flow). The algorithm by **Horn and Schunk**, is slow, but also enforces global constraints.

Stabilization in space

All methods so far were operating in 2D. While the methods can produce good stabilization, the result does not look professional, it looks like the camera is wandering aimlessly. This is

because the stabilization algorithm is not actually **aware of the position of the camera** and therefore cannot improve it directly.

3D stabilization, on the other hand, operates on 3D camera trajectory. The trajectory can be obtained from a video sequence using the **SfM** (Structure from Motion) algorithm. This family of algorithms simultaneously produces a **sparse 3D reconstruction** of the observed scene and recovers **camera position** in it. Using this information, a more stable trajectory can be fitted, e.g. a straight line or a parabola. These kinds of trajectories are usually achieved using props such as a dolly or a Stedicam.

The challenge of 3D stabilization is that we are essentially moving the camera in space and every change in viewpoint will cause small non-linear changes in observed scenes because the information is usually not planar. Simply using a projective transform will destroy the illusion of depth. But since the changes are small, a satisfying result can be obtained using non-linear warping. This will produce small non-linear distortions based on the content of the image that will preserve the depth information. One way to obtain such a transformation is to project points from the SfM sparse reconstruction back into the pairs of images and observe how they move between two frames. Points close to the camera will move more than those further away. Based on these correspondences, we fit a quad mesh that generalizes the transformation and ensures its local smoothness.

Missing regions and mosaicking

One of the problems with stabilization is that transformation of image produces **missing border regions**. For a given frame, the information can be shifted in relation to the initial frame so we end up with some excess information on one side and missing regions on the other. There are several techniques to limit or remove this effect. If the missing regions are not too big, we can simply **crop the video** to the region that is visible in all frames, essentially zooming in. Trajectory smoothing is also used to limit the size of missing regions. The process involves a low-pass filter that processes the transformation to allow slow movement, but suppress high-frequency jerky movement that is the most problematic part of the camera movement. In some cases smoothing is even preferred to complete stabilization as it appears more real.

The more advanced option is to use **video mosaicking** to fill in the missing regions based on information from the previous frames. Video mosaicking is a technique that **fuses multiple frames together** into a larger image, similar to panorama stitching. The requirement is that the content of the video does not change a lot because all images are projected into the same image space. The base technique is similar to stabilization, key-points or optical flow may be used to calculate transformations. In its simplest form, the mosaic can then be used to fill in information for the current frame from the past (or future) frames that have the information for that part of the scene. A more advanced approach uses **optical flow** to estimate how individual parts of the image move and take this into account when filling in the missing information (this way also moving objects will appear at more accurate positions).

Local stabilization

Local stabilization or image re-centering produces a video cut-out of a single object focused in the center of the video. The object trajectory is obtained using an object tracking algorithm. The trajectory is used to cut out regions of the image. Object tracking works by remembering the appearance of an object (e.g. color, texture) in the initial frame and finding it in the frames of the sequence. In case the object comes too close to the border of the image frame, we again have problems with undefined regions. We can mitigate this problem with image extrapolation or clamping the coordinates of the crop region.

The trajectory can also be smoothed for better effect as the center of the object is usually poorly defined and the trajectory will be pretty shaky. A low-pass filter will remove these high frequencies.

Background removal

TODO

Detecting transitions

Between production, a video is usually combined from multiple shots that form multiple scenes. In multimedia applications, it is sometimes good to know that we are observing a new video shot or even a new scene, however, this information is typically not available anymore in a video stream. We will look at some simple techniques to detect transitions that are the basis for detecting boundaries of individual shots.

There are multiple primitive transition types:

- **Cut** - a clear cut between two frames
- **Fade-out** - gradual fade to a color
- **Fade-in** - gradual transition from a color
- **Dissolve** - alpha fade from one shot to another
- **Wipe** - gradual spatial replacement with a rolling binary mask

Detecting transition depends on transition type. Generally a transition causes a rapid change in distribution or presence of **low level image features**, e.g. a change in color distribution, edges, etc. For cut transitions observing only two frames is enough, in crossfades observing more frames is needed. Sometimes setting a **fixed threshold** is enough, while in the case of more dynamic shots, the threshold should be set **adaptively**. We are essentially talking about two steps - (change) **scoring** and **decision**.

One of the most simple forms of shot detection is by comparing color histograms of consecutive frames. **Adaptive threshold** means that we are observing differences between multiple frames

before and after the current frame and setting the threshold to the higher of the two means increased by a portion of variances for the previous and after frame intervals.

In case of **gradual changes,** shot detection can be improved by **averaging features** over several frames and comparing two intervals. Another option is to look at partial changes, i.e. changes in only part of an image. An image is divided into a grid and each cell is checked for a large change. If enough cells are triggered, then we can say that the shot has changed.

Besides color, **edges** can also be used for shot detection. In this case we do not use histograms, but compare edges spatially, how many edge pixels are common to both frames and how many not.

Detecting fades can also be detected using a **dual threshold** approach. Since the change is not as apparent in case of fades, it cannot be detected with a single high threshold, however, we can start monitoring change if the change exceeds the **low threshold**. We compare all following frames against the potential start frame and call this **cumulative difference**. We do this until the cumulative difference is increasing, if it is not, we compare the difference to the **high threshold** which tells us if the fade actually occurred.

Sound

Besides images and video, sound is the most important and best studied medium in multimedia environments. When a sound is recorded or stored on a digital device it is commonly referred to as **audio**, which is a technical term for the digital sound. There are many specifics about how audio is recorded, processed and stored, some of which we will discuss in this chapter.

Sound acquisition

Source material: Li and Drew, Fundamentals of Multimedia, Chapter 6

Sound is a physical phenomenon caused by **waves of pressure** in the **medium**. These waves are **longitudinal waves** in which particles of the medium are compressed and expanded. A typical medium for sound is **air** and **water**, without it, the sound will not be perceived (e.g. in space). As a wave phenomenon, sound is subjected to several effects that occur in complex spaces with multiple different materials.

- **Reflection**: caused when sound hits a surface and bounces back, causing an echo.
- **Refraction**: when the sound enters a different medium with different density, its angle may change.
- **Diffraction**: the sound bends around an obstacle

To record a sound, one must first know its physical properties and how to measure them. The main properties of sound are **frequency** and **amplitude**. Frequency tells us how many medium

contractions occur in a unit of time. It is measured with **Herz** (Hz), the number of contractions in a second. Amplitude tells us how large the changes in pressure are, i.e. the largest change in pressure caused by the sound. In the physical medium it is measured in **Watt per square meter** (W/m2). An observed sound also has **duration** (what is the total length of the observation), **direction** (in which direction do the waves travel), and the speed of the waves, which is based on the medium that we are observing the sound in. In air the **speed of sound** is approximately **331 meters per second**.

Human auditory perception

In multimedia, the acquisition and presentation of audio is strongly motivated by the good human experience and perception. It is therefore important to understand how we perceive sound.

The human organ that is responsible for perception of sound is called an **ear**. Ear is not only the visible part of the head but a system of canals and bones inside the skull that process the sound. The sound travels through the **ear canal** to the **eardrum**, a membrane that transmits the vibration to **ossicle** bones, where vibrations are amplified and are transmitted to the liquid inside the cochlea. Inside the **cochlea** there are **hair cells** that are sensitive to different frequencies, their activation is then transmitted to the **auditory nerve** and then to the brain.

Humans cannot hear just any frequency of sound, a rough general estimate is that we can perceive vibrations between 20Hz and 20kHz as sound. Below the 20Hz we have **infrasound**, which is used in investigations of Earth's crust among other things. Above 20kHz we have **ultrasound** which is used in sonars and medicine, bats also use ultrasound frequencies in echolocation.

For a sound to be heard, we do not only need the right frequency, but also **sufficient power** or amplitude. The amplitude required to perceive a certain frequency depends on the frequency, we need more energy in frequencies towards the ends of the hearing interval. The amplitude at which at least half of the tones were perceived by test subjects is called **threshold of hearing**. The threshold changes with years, it is well known that young people hear high frequencies (above 2 kHz) better, while older people may require louder sounds in this range.

At the high level humans do not perceive the sound as frequencies, we assign it perceptual qualities, which are the effect of a combination of physical properties of the sound. A **pitch** of a sound can be low or high, it is primarily related to the frequency of a sound, but also to other aspects, such as intensity and waveform. **Loudness** is primarily related to sound intensity, but also related to frequency. **Timbre**, or the **color of a tone**, is primarily related to the waveform, the presence of different frequencies in the sound. In relation to this humans perceive **sonic texture** when a sound is coming from multiple sources. A sonic texture can be perceived as euphonic sensations, such as unison, polyphony, and homophony, or as cacophony (harsh,

discordant sound). Since the sound is perceived with two ears, we can also perceive an approximate **spatial location**.

Signal and noise

One of the characteristics of sound is how well we can separate it from the unwanted fluctuations of the environment that we call noise. In practice the loudness of a sound is not described by its amplitude, but by the relative loudness in comparison to the noise. The factor is measured in tenths of a bel, or **decibels** (dB). Most commonly the loudness is specified by comparing signal to the sound of a just audible 1 kHz noise.

Digital sound

The advantage of digital sound is of course easier processing and storage with compression. Sound is an analog signal that can be **measured** as a changed **air pressure** with time resulting in vibration of the recording membrane that is transferred to mechanical motion and then to electrical signal is measured. To digitize it, we have to **sample the signal** at fixed intervals and **quantize** the real values to a fixed resolution. Improper digitalization corrupts the signal in several ways, making it less real. The two main sources of corruption are **aliasing** and **quantization noise**.

Signal is sampled at uniform time intervals, the frequency of sampling is also known as **sampling rate**. According to the **Nyquist-Shannon theorem** the sampling rate should be at least twice as high as the highest frequency in the signal. If this is not true, the signal will contain **aliasing** due to insufficient number of samples that produce ambiguous results. To ensure that the signal only contains suitable frequencies, it is usually passed through an analog **low-pass filter** before it is sampled.

The **quantization** occurs after sampling. Each **measured value** in the interval of **perceivable amplitudes** is assigned an **integer value**. This involves **rounding** which produces **quantization noise**. Quantization noise is measured with **signal-to-quantization-noise ratio** (SQNR), higher value is better since this means that the signal is stronger in comparison to the noise. A rule of thumb is that **12 bits** are already enough for adequate sound reproduction. Additionally, the quantization errors can be made more **statistically independent** and therefore less noticeable if a small amount of noise is added to the signal, making rounding errors more random.

The term **Pulse Code Modulation** (PCM) is a formal term for the combination of **sampling and quantization**. The PCM approaches vary in the type of quantization used. **Linear quantization** maps analog values to a linear interval. This may not be desirable in some applications, for example in telephony. The reason is that low-amplitude signals are encoded with a larger error than high amplitude ones. This affects the clarity of **human speech**, which is more efficiently encoded (higher SQNR) using **non-linear resolution** schemes, such as **A-law** and **μ-law**. The idea of non-linear compounding is that lower amplitudes are represented with more resolution

than high-amplitude ones - this idea mimics human perception. Other quantization approaches are **Differential PCM**, which encodes difference to previous value or the difference to a predicted value is encoded, and **Adaptive Differential PCM** where the quantization levels change with time.

Frequency spectrum

One of the basic concepts in sound analysis is a **frequency spectrum**. It shows how strongly (with how much amplitude) are certain frequencies present in the sound waveform. Frequencies are represented by **sinusoidal functions**, each of them is assigned a coefficient that signifies their strength. Spectrum can be shown for a long signal by cutting it into fixed intervals and analyzing each interval separately.

The method that decomposes continuous signals in a set of basis sinusoids is called **Fourier transform**. Working with frequency distribution simplifies certain operations, like noise removal. A frequency spectrum can be transformed back to a time-domain with **inverse Fourier transform**. In digital signal the Fourier transform has a counterpart method called **Discrete Fourier Transform** (DFT), it is a linear transformation of a n-point time-domain signal to a n-point frequency spectrum. The DFT can be efficiently calculated using an algorithm called **Fast Fourier Transform**, which takes O(n log n) time for a signal of n points.

Sound processing

Source material: J. O. Smith III, Introduction to Digital Filters

Audio signal processing is a subfield of general signal processing. Audio processing can be done with **analog audio signals**, represented by continuous functions in the form of changing electrical current or voltage. In this case the properties of the signal are changed using electronic components, like capacitors, resistors, inductors, transistors, etc.

Digital audio processing is performed by algorithms that run general purpose computers and manipulate a **digital signal** (sampled and quantized). This approach is more general and powerful, and usually also more efficient. Digital filers also generally have better **signal-to-noise** ratio because noise only occurs during conversion from analog signal and back and potentially due to quantization. In analog filters every component is a potential source of noise, the more complex the filter, the greater the noise.

Digital filters can be characterized in multiple ways, knowing where a filter belongs to is important because filter classes have different properties and limitations. A filter is **linear** if it can be written as a **linear difference equation**, otherwise it is **non-linear**. If a filter only uses past signal values for computing the output, it is considered **causal**, if it also uses future values, it is considered **non-causal**. If the filter returns the same signal no matter when the input is fed into it, it is considered **time-invariant**, if the output varies with time, it is considered **time-variant**.

The filters with nicest properties are linear time-invariant filters (LTI), which have the property of **additivity** (sum of filtered signals is the same as filtered sum of signals) and **homogeneity** (scaling input signal equals scaling output signal). These properties are also used for analysis; we can observe how a primitive sinusoid signal is transformed by a filter in terms of its parameters (amplitude, phase, frequency). This kind of analysis is called **frequency analysis**.

Linear filters

Most audio filters can be implemented or approximated as linear filters, meaning that the new output value is a result of a **linear combination** of a finite number of past values in the signal. The most general LTI filters are called **subband filters** that only pass frequencies in a certain interval (passband) of the spectrum and suppress others frequencies (stopband). The most common filters of this class are:

- **Low-pass** - passes only frequencies below a threshold
- **High-pass** - passes only frequencies above a threshold
- **Band-pass** - passes frequencies in an interval
- **Band-stop** - blocks frequencies in an interval

All subband filters can be implemented with a single **prototype filter**, e.g. we can get a high-pass filter by subtracting low-pass signal from an unfiltered signal.

Implementing good subband filters is challenging. An ideal low-pass filter, for example, should pass all frequencies in passband up to the cutoff frequency with a constant gain and completely stop all frequencies above that. Obtaining a low-pass filter with ideal properties is not possible since you would need to consider an infinite signal. But there are many finite approximations with different advantages and disadvantages:

- **Butterworth** - frequency response as flat as possible in the passband (uniform gain)
- **Chebyshev** - minimize the error between the idealized and the actual filter characteristic over the range of the filter, but with ripples in the passband (non-uniform gain) in case of Type 1 filter or or stopband in case of Type 2 filters
- **Elliptic** - equalized ripple behavior in both passband and stopband

Equalization

A classical case for using subband filters is **multi-band equalization**. A signal is divided into multiple bands which are then gained with different factors and combined back together. This allows us to boost bass frequencies or treble frequency range.

Comb filters

A comb filter is a building block of many real effects. It is called this way because it produces a **comb-like pattern** when observing the **amplitude response** in frequency analysis. A general

comb filter combines a signal with a past input sample and a past output sample, each of them weighted with its corresponding factor.

$$y(t) = x(t) + \alpha x(t - a) + \beta y(t - b)$$

The two cases where either past input or past output samples are weighted with zero are called **feedback** and **feedforward** comb filters.

One of the more easy-to-understand comb filter applications is **delay**. The delay filter propagates the same signal with a given time difference and sums it with the original signal.

Echo filter is an extension of the delay filter with the aim of **simulating acoustic properties** of real rooms. In a room, sound bounces off multiple walls potentially multiple times, creating a unique effect that is perceived as listening in a large or small space. An **acoustic fingerprint** of a room is divided into **early reflections**, individually perceived delayed signal copies with decreased gain and **subsequent reverberation**, a random noise that is the result of multiple echo signals joining together.

Flanger is another derived delay effect. The delay time difference varies with low frequency in this case, which produces a bouncing effect. Because of the additional time component, the flanger filter is not time-invariant, but is an example of linear, time-variant filter.

Chorus is technically similar to flanger, but the delays are longer in this case, the pitch of the duplicated signal can also be shifted a bit. The effect that chorus is trying to achieve is that of multiple instruments being played together, each with a bit different timing (and/or pitch). Naturally, the effect appears in orchestra or choir. The resulting sound is more rich than the input.

Non-linear filters

Non-linear filters are generally all filters that cannot be formulated as a linear equation. There are many applications for non-linear filters. The first example is noise removal using a median filter, which is a non-linear operation. Other examples are dynamic range compression filters which apply a dynamic scaling of signal amplitude based on its value. Dynamic range compression is used to efficiently utilize available range on magnetic recording tapes (companding) or to clip high amplitudes (clipping).

A classical example of non-linear clipping filters in audio processing is **distortion** or **overdrive effect**. This effect was usually achieved by overloading vacuum tubes that were used in amplifiers, pushing them beyond their maximum. This resulted in soft amplitude clipping. This clipping resulted in the presence of new frequencies in the output signal, which makes the distortion a non-linear filter.

Digitally, distortion is implemented as a **non-linear amplitude scaling function**, the low amplitudes are mostly left as they are, while high values are significantly scaled. Based on the function used, we call the output either soft or hard clipping (thresholding).

Spectrum Analysis

An efficient way to implement FIR filters on a long signal is using spectrum modification techniques. This approach assumes that a signal can be split into multiple segments that can be processed individually in frequency space using DFT and then recombined back.

These signal segments are obtained by multiplying the signal with an offsetted **window function**. The definition of a window function is that it is zero outside of the area of interest, it is also symmetrical. The most basic window function is a rectangular window, but it has a disadvantage of having a lot of **spectral leakage**, i.e. cross-talk between segments. Better window functions with different properties include: triangular, Hann, Hamming, etc.

The segments can be processed in Fourier space and then re-combined using Overlap-Add or Overlap-Save methods. These methods assume that parts of segments overlap and are summed together (in case of Overlap-Add).

TODO

Compression

Multimedia data takes up a **lot of storage and bandwidth**, compression is therefore a crucial component of many multimedia applications and is used when data is either stored or transmitted. Fortunately sensory data (e.g. visual and auditory data) lends itself very well to the compression process. The first observation is that a lot of data is **strongly correlated** and part of it is therefore **redundant**. This is a fact exploited by **lossless compression**, a type of compression that only reduces the size of the data, but retains its content intact. In terms of visual information we talk about **spatial correlation** where color in neighborhood pixels is unlikely to change a lot for the majority of pixels and **channel correlation** where information in the three color channels are also correlated. In the video we also know **motion correlation**, which means that for most pairs of consecutive frames most pixels do not change a lot.

On the other hand, the primary goal of multimedia data is to be **enjoyed by humans** and is therefore considered useful as long as this goal is achieved. Since human perception has certain properties that make us less sensitive to some local aspects of data, this fact can be exploited by **lossy compression** algorithms. Human perceptual system is **differently sensitive to different types of information**, e.g. it is more sensitive to changes in intensity than in chroma. This fact can be exploited by using the same chroma information for multiple intensity samples (pixels), effectively saving a lot of space.

Compression basics

Source material: Li and Drew, Fundamentals of Multimedia, Chapter 7

A typical compression process is performed in **several stages**. The first stage is the **transformation** of data to a form that is more suitable for compression. The transformation may be done so that it reduces correlation in data or to change statistical properties. Typical transformations in image compression are predictive coding that transforms signal to an offset of predicted value at a given time, color conversion and representation of a part of an image in frequency domain.

The second stage of compression is **mapping** where data is mapped to symbols that are more efficient to encode. Data can be split into smaller chunks at this stage, e.g. an image is split into blocks. Another well known mapping technique is **Run-length-encoding** (RLE), which partitions the data into repetitions of the same data values. In RLE each symbol is composed of a data value and the number of times it is repeated in a sequence.

Transformation and symbol mapping are considered preprocessing steps, most data compression occurs in the final, symbol encoding step where each symbol is assigned a binary code word. The encoding dictionary, i.e. the mapping of symbols to words, can be either formed online or offline. The main goal of lossless compression is to minimize the number of bits without losing any information. There are two general approaches to achieve this. The first one uses symbol **statistics** to code more frequent symbols with shorter words and less frequent with longer words. This approach is also called **variable length coding** (VLC). The statistics can be computed for a part of the data or for the entire stream. The most well known representative of this family is the **Huffman coding** algorithm. The second family of approaches builds a **dictionary** of frequent subsequences of symbols and uses it to shorten the final representation by only referencing the subsequences from the dictionary where possible. The dictionary is generated dynamically without using any symbol frequency statistics. The representatives of this family are derivatives of the **Lempel-Ziv** algorithm.

The reverse process of compression is called decompression and is performed in reverse order with the inverse operations: **decoding**, **reverse mapping** and **inverse transformation**.

When designing a data compression solution for an application, one has to consider multiple aspects that define which compression approach is the best.

- **Efficiency** - what is the ratio between raw data size and the size of a compressed message.
- **Delay** - time required to compress (or decompress) the data, also the amount of data that is processed at once.
- **Implementation complexity** - how complex is the implementation, also memory usage.
- **Robustness** - can a corrupted compressed stream be corrected.

- **Scalability** - support for different compression profiles.

Statistical encoding methods

The **Huffman encoding** is the most well known statistical compression algorithm. It is based on the phenomenon of information entropy, which tells us how many bits are needed to encode a single character based on the probability of its appearance in a sequence of symbols. The naive way of coding each symbol with one of the N equally long words is not optimal, unless all symbols appear with equal frequency.

The Huffman algorithm is a type of optimal prefix coding that determines code words that minimize the average length of words and satisfy additional requirements:

- **Regular** – different symbols are assigned different words
- **Unique** – message can be understood in only one way
- **Instantaneous** – each word can be decoded as soon as it is read

The algorithm first sorts symbols by decreasing probability of occurrence. It then iteratively merges the two symbols with the lowest probability and combines their probabilities. It also assigns them one bit in the final word to distinguish them. This process is repeated until all symbols have been merged. The more frequent symbols will therefore have less bits in their code words.

The lower bound of coding is one bit for a symbol in case we have a very frequent symbol - each symbol separately requires a non-zero integer number of bits. The Huffman algorithm is optimal when we have to encode each symbol separately. This constraint is avoided in an improved statistical coding method called arithmetic coding. Another problem of Huffman coding is that a deviation of symbol frequency statistics will make coding sub-optimal. Since updating the code words online can be computationally expensive, some improvements to the original algorithm limit the maximum possible size of the code word, making a compromise between the initially optimal encoding which then deviates and the always suboptimal equal-length code words.

Dictionary encoding methods

The problem of statistical coding is that it requires knowing symbol distribution in advance, which may not be desirable in some cases. The distribution may also change over the sequence and the mapping dictionary has to be re-generated.

Dictionary coding algorithms use a dynamic dictionary generation for variable length sequences of symbols. Code words are of fixed length and tell us which subsequence from the processed message to use. This makes dictionary algorithms suitable for on-the-fly compression and decompression, however, this compression technique is not suitable for short data sequences and may even result in longer encoded messages in this case.

The fundamental algorithm for dictionary compression is Lempel-Ziv, or **LZ77** (the algorithm was presented in 1977). The algorithm uses a sliding-window, a buffer of the uncompressed processed data that is used as a dictionary for new data. The dictionary is therefore dynamic, it changes with the data stream. It also means that the data can be decompressed from the beginning as some later code words may refer to the previous symbol subsequences. The code-words in LZ encoding are triplets that tell us the offset and length of the repeated subsequence as well as a single symbol that follows this repeated segment. The length of the repeated subsequence may also include symbols that will be written by the current command. This makes several repetitions of the same string encoded in an even more compact manner.

The main problem of LZ77 encoding is the limited dictionary size that corresponds to the buffer. The buffer can be increased, but this leads to high CPU load. An alternative algorithm, called **LZ78** was proposed by the same authors. In this case the dictionary is explicitly built over the entire sequence or subsequence. The code-words are simply indices in the dictionary. This also allows partial decompression as long as we have the dictionary available. Another improvement is the LZW algorithm in which the dictionary is pre-initialized with all possible symbols. When a match is not found, the current symbol is assumed to be the first symbol of an existing string in the dictionary.

Based on the data at hand sometimes the statistical approach is better and sometimes the dictionary approach provides shorter encoding. Both approaches can also be combined, e.g. in case of the **DEFLATE** algorithm, where LZ77 is first used to eliminate duplicate subsequences of symbols. The resulting code-words are also encoded with prefix codes using Huffman algorithm. The DEFLATE encoding is scalable, we can set the time that can be spent searching for substrings. The algorithm is used in well-known file-compression formats, such as PKZIP and gzip.

Image compression

We know of many image compression approaches, some lossless and other lossy. We will look at the key properties of the lossless PNG format and the lossy JPEG format.

Portable Network Graphics

The **Portable Network Graphics** (PNG) format was introduced in 1996 as a substitute for another format, the **Graphics Interchange Format** (GIF) that was burdened by patents and had several limitations (limited transparency, only limited palette) that were limiting its use-cases in the World-Wide-Web. The PNG format provides full alpha channel transparency and supports full RGB as well as indexed palette modes. To achieve an acceptable compression ratio for lossless use-cases, the PNG uses two-stage compression. In the first stage, the pixel values are filtered using **predictive filtering**, the value of the pixel is predicted using previous values and only the difference (error) is preserved. In the second stage the differences are compressed using the DEFLATE algorithm.

The predictive filtering is a common approach for compressing sequential data that can be at least partially **modeled**. In case of images, the predictive filtering usually works **per-row**. There are multiple models that can be used, they use different **combinations of past** (already processed) **values** to determine prediction. The PNG format uses several approaches that can be switched on per-row basis (a heuristic algorithm is used to determine the most suitable approach for the current row):

- Unaltered value - each value is encoded without prediction
- Use value of previous pixel in a row
- Use value of previous pixel in a column
- Use mean value of previous pixels in row and column (round down)
- Use a combination of several previous values

The important aspect of compression that will be also visible later on in other compression approaches is that the choice of prediction model is up to the compression algorithm implementation and does not influence the ability of the decompression algorithm. Some compression algorithms can be more successful in choosing the suitable model, but the format is PNG even if it is not optimally compressed.

Block truncation coding

The main motivation in lossy compression of image data is to exploit spatial correlation. This means that local pixel neighborhoods are usually very similar. This can be exploited by dividing images in small regions and encode each region in a way that some information is discarded (the lossy component).

Block truncation coding (BTC) is a simple example of lossy encoding for images. Each block of 4x4 pixels is encoded only with **two intensities**: the mean value plus or minus a standard deviation weighted by the ratio of pixels above and below standard value. The encoded size of each block for 8-bit pixels is therefore 4 bytes (1 byte for mean, 1 byte for deviation, 2 bytes for 4x4 bit array), while the uncompressed size is 16 bytes. The BTC compression, although very simple, was used in real life, e.g. on NASA Mars Pathfinder mission in 1997.

The JPEG format

Source material: Li and Drew, Fundamentals of Multimedia, Chapter 9

The JPEG is an abbreviation for **Joint Photographic Experts Group** which proposed the format in 1992. Until then it has become the most popular standard for lossy image compression. The standard primarily defines **lossy compression**, lossless compression was added as a late addendum and is rarely used. The encoding and decoding was designed so that it does not have high computational requirements. This means that it can be used also on embedded devices, such as cameras. The format is suitable for any type of images, it supports color depths between 8 and 12 bits per channel. The format supports **sequential** and

progressive encodings. In sequential encoding, the image is encoded in single resolution, while the progressive encoding means that the image is first encoded at a lower resolution and then more details are added progressively. The main difference is in loading the image when not all data is yet available. Progressive image can be shown whole at lower resolution, while sequential image can only be shown partially in this case.

The JPEG file format contains image data as well as some metadata. The format is also known as **JPEG File Interchange Format** (JFIF). A version of the format with additional metadata about a digital camera it was created on is known as **Exchangeable Image File Format** (EXIF). Since only more metadata is available in EXIF, both containers are compatible with regards to the image content.

The lossless compression is similar to PNG, it uses predictive coding, followed by Huffman coding. It is not frequently used, the more popular option is the lossy mode. The main properties of the lossy JPEG compression are used also in video coding. The idea is to take into account the human perception system and discard information that is less relevant to humans. The less relevant information in this case are essentially higher frequencies within local regions.

The image is decomposed into **8 x 8 pixel blocks**. If the image dimensions are not divisible by 8, extra data is added. Each block is then transformed to frequency domain using **Discrete Cosine Transform** (DCT). The DCT is similar to the Discrete Fourier transform, but is more suitable for compression as it describes the majority of the signal with less basis functions, which we call **spectral compaction**. The DCT decomposes 64 bytes into a **direct current term** (DC) which is essentially the mean value of the block and 63 **alternating current terms** (AC) which are coefficients of different cosine basis functions. The coefficients are quantized based on the **quantization table**, which is based on psychophysical tests and **quantizes higher frequencies more coarsely**. The exact **level of quantization** can be controlled with an input parameter which makes the format scalable and allows choice between final size and image quality.

The quntized DC coefficients are then encoded using **predictive coding** as differences to the DC of the previous block. The quantized AC components for each block are ordered using a **zig-zag pattern** and then represented as an **RLE sequence** (the reason for this is that a lot of coefficients will be zero after quantization). All symbols are then encoded using **Huffman encoding**, which can use predefined code words or calculate them on-the-fly.

The JPEG format also supports **chroma subsampling** as another measure of lossy size reduction. Since the human eye is less sensitive to changes in chroma than to changes in luminance, we can use chroma-separated color format, in case of JPEG this is YCrCb, and downsample chroma channels with a specific factor (usually 2). The quantization table used for chroma data is also different from the one for luminance.

Video compression

Source material: Li and Drew, Fundamentals of Multimedia, Chapters 10, 11, 12

Video data is even more demanding than single image data because the size is multiplied by its temporal length. Exploiting only inter frame correlation (spatial, chroma) is one option, a practical example of this is **Motion JPEG** codec which essentially encoded each frame as a JPEG image. Since JPEG is designed to be implemented efficiently, encoding and decoding can be done in real time, but the resulting bitstream has low compression ratios, e.g. 1:10 or 1:20, which are unacceptable for modern streaming and other multimedia use-cases.

Exploiting **temporal correlation** is therefore necessary. A typical widely-used approach to temporal encoding is using **block matching**. An image is divided into blocks that are then searched for their match in the previous frame[1]. Usually a perfect match is not found, but the idea is to find the **most similar patch**. The block can then be encoded using **translation vectors** and the **differences** to the reference block. More sophisticated techniques, such as grouping blocks together into macroblocks with shared compression parameters can also be used this way.

The video compression and decompression standard is usually referred to as **video codec**. Choosing the right video codec for a specific use case can be difficult and depends on multiple parameters.

- Compression level
- Transfer speed (bit-rate) vs. distortion/loss
- Algorithm complexity
- Communication channel characteristics (delay, errors)
- Fixed / variable bit rate
- Standard compatibility
- Losing information during compression

In the past video codecs were developed by various companies who defended their market shares also by restricting access to their decoders which resulted in poor interoperability. Nowadays, most well-known and used video codecs are proposed by the Motion Picture Expert Group (MPEG), a working group, a joint working group by ISO and IEC, established in 1988, that proposes standards for video, audio and other multimedia forms. Some notable standards are:

- Video and audio on digital devices (MPEG-1, MPEG-2)
- Video, audio, 3D graphics, Web, ... (MPEG-4)

[1] The block matching technique can be simple or more complex, the cost function depends on the encoder and is not needed during decoding.

- Storage and retrieval in video (MPEG-7)
- Interaction with multimedia applications (MPEG-21)

MPEG-1

The MPEG-1 is the first standard, published by the workgroup that included video and audio compression specifications. The entire standard has five parts: video, audio, compliance, reference implementation, system. The aim was to achieve storage and playback for VHS-quality video and audio and enable real-time transmission at bandwidths at around 1.5Mbit/s. The compression aim was therefore 1:26 for video and 1:6 for audio without excessive loss of quality. The intended use is asymmetric, the content is compressed once and decompressed many times, therefore the encoder can be complex and encoding slow, but the decoding process should be simple and fast and should run on various hardware configurations. The video codec of the MPEG-1 meets several requirements, set by the proponents:

- Normal playback with random access
- Support for video editing
- Reverse playback and fast playback
- Different resolution, frame rate
- Cheap hardware implementation of decoders

Each frame in a MPEG-1 video sequence is represented using **YCrCb color space**. The standard supports 2:1:1 subsampling, meaning that a block of four pixels has a shared chroma information. This is shown in block division, luminance Y channel is divided in 16x16 pixel blocks, while Cr and Cb channels are divided into 8x8 pixel blocks. Blocks are combined in groups, called macroblocks, which are then joint into sequences, called slices. Each slice can have different compression parameters that can adjust the size of the slice versus the quality. Additionally, since each slice is encoded separately this makes error recovery easier.

To account for temporal redundancy, there are three main **frame types** in video codec, the I frame, the P frame and the B frame. There are also D-frames, which are low-resolution representations of key-frames and can be used for faster navigation, however, they are rarely used.

The **I frames** are **inter frames**, each frame is self-contained and can be decoded completely in a similar manner to a JPEG image, although using a bit different quantization tables. The frames have their chroma subsampled and are organized in strides of macroblocks. These frames are also called keyframes for the purpose of decompression because they allow random access to the video content - they do not require any additional information to decode them.

The **P frames** are **predicted frames** that depend on the previous P or I frame for data and only encode differences to it. The data for each macroblock is encoded as a motion vector, usually computed using the similarity of the Y channel and the difference to the corresponding data from

the previous frame. Only if the differences in the macroblock are too big and would require too much space, the macroblock is encoded as is and not as difference.

The **B frames** are **bi-directional predicted frames** that are encoded using previous and next I or P frames. In this case an **average** of patch from previous and patch from the next reference frames is used as the basis for difference calculation. B frames require prior decoding of multiple frames, some from the future of the stream, therefore we also call their decoding process **delayed decoding**.

The video sequence datastream of MPEG-1 video is organized hierarchically. A sequence is composed of multiple groups of pictures (GOP), each picture is composed of multiple slices, each slice of multiple macroblocks, each macroblock contains five blocks and each block is represented as DC coefficient and a set of Huffman code words (variable length codes - VLC).

MPEG-2

The second standard, denoted as MPEG-2, was designed with **higher frame resolutions** (HDTV) and a bit **better transmission bandwidth** (4-15 Mb/s) in mind. It is meant to be suitable for **digital TV** and introduces support for **interlaced video** and better **robustness** (error correction). The video codec is also designed to be highly **scalable** and compatible with MPEG-1 video codec, i.e. MPEG-2 decoders should also decode MPEG-1 streams.

The interlaced video can be encoded in two ways. In **frame format** odd and even lines are encoded as a single image, this means that they are encoded using a single header (parameters). This is also the format used in progressive mode. On the other hand, the **field format** encodes odd and even lines separately. Every field encoded as a separate image with its own header. The encoder can switch between two modes, choosing the one that is more suitable regarding size and image quality.

The video codec standardizes different profiles for various applications and quality levels (e.g., DVD, recordings from two cameras). The **scalability** aspect of the MPEG-2 standard is important because some consumers of multimedia content cannot consume content in its full resolution either because of **bandwidth or computational limitations**. Since the standard does not define two-way communication, the video has to be separated into **multiple layers**, a **base layer** that is self-contained with only coarse information and **enhancement layers** that gradually improve the quality. The transmission of layers is separated and a client can simply only retrieve and decode some layers and still get a decent quality. There are multiple types of scalability:

- **Spatial scalability** (image size), base layer provides frames with lower resolution, resolution is improved with enhancement layers.

- **Frequency scalability** - base layer provides only DC coefficients and motion vectors, enhancement layers provide more and more AC coefficients.
- **SNR scalability** (signal-to-noise ratio) - base layer provides heavily quantized intensity, coded in original resolution, enhancement layers provide residual information
- **Temporal scalability** - base layer provides images with lower frame-rate (e.g. only I frames), enhancement layers provide missing frames in between, using motion prediction to reconstruct them (e.g. B frames)

MPEG-4

While the MPEG-1 was primarily a VHS digital substitute and MPEG-2 addressed digital television, the focus of MPEG-4 is **interactive multimedia content**. The video codecs that are part of this standard have even better compression rates with higher throughput (bitrates). They are robust in environments with frequent errors. One of the aspects of the standard is also the definition of a **media objec**t and how multiple media objects can be randomly accessed and used together to form an interactive experience.

Media objects have different sources, they can be real or synthetic images, sounds, vector graphics, video. The key idea is that individual objects can be **interacted** with and **manipulated**, but they can also be **indexed** and **retrieved**. Media objects are separated into multiple **channels** and encoded separately, they are transferred in separate streams. The objects can then be organized hierarchically to form a **scene** when decoded on the render device. The standard defines a scene description language, **BInary Format for Scenes** (BIFS). The BIFS defines the following hierarchy

- Video-object Sequence (VS) - Complete visual scene
- Video Object (VO) - Arbitrary non-rectangular shape
- Video Object Layer (VOL) - Scalable coding support for video objects
- Video Object Plane (VOP) - Snapshot of VO at a given point in time, in MPEG-1 and MPEG-2 the entire frame is a VOP
- Group of Video Object Planes (GOV) - Optional group of VOPs

In terms of video compression, MPEG-4 defines two video codecs. The first video codec in MPEG-4 standard is MPEG 4 Part 2, better known as **H.263**. It defines 21 video profiles designed for various applications. The main improvements in comparison to MPEG-2 video codecs are **quarter-pixel motion compensation** (Qpel), an ability to define motion vectors down to a quarter of a pixel. This requires interpolation of blocks, but can improve image quality. The other improvement is **global motion compensation** (GMC) which encodes the global motion of a scene using an affine transformation. This can be useful in case of camera motion or zooming. Each macroblock can be encoded using GMC or by using a local motion vector. The most notable implementations of the H.263 are known as DivX and Xvid video codecs.

The second video codec in the MPEG-4 standard is called H.264 or Advanced Video Coding (AVC). The first finished version of the coded specification was published in 2003, the coded

was initially intended to be used in high-definition video (Blu-Ray), but its adoption has since then grown to be the most widely used video codec, also used for HD terrestrial television, it is included in Adobe Flash Player and Microsoft Silverlight, and more importantly, it is a de-facto standard for video in **HTML5**. The codec offers an increased compression ratio (up to 1:50) and specifies a number of profiles, suitable for anything between video-conferencing and high-resolution stereoscopic streams. The main advantages of the H.264 are:

- New VLC techniques, the **context-adaptive variable length coding** (CAVLC) and **context-adaptive binary arithmetic coding** (CABAC), both offer an efficient way of encoding data using entropy coding. The CABAC can achieve better results, but requires more computational power.
- **Variable block sizes** - macroblock can contain blocks of different sizes.
- **More accurate motion compensation** by enabling more than one motion vector per macroblock.
- The H.264 also introduces **intra-frame prediction**, an ability to predict values of entire blocks using neighboring blocks (encoding difference to the block).
- Since compressing blocks individually can produce artifacts at the borders of blocks that makes them noticeable (the effect is also called blocking), the codec introduces **signal-adaptive deblocking filters** that smooth edges around the blocks, making them less noticeable and thus increases the subjective quality of the resulting video (even though the result of the filtering will not necessarily be a more accurate representation of the source video). Unlike in MPEG-1 and MPEG-2 the deblocking filter is not an optional addition of the decoder, but a feature in both the decoder and encoder. The in-loop effects of the filter are taken into account during encoding and when a stream is encoded, the filter strength can be selected and sent to the decoder.

Newer video coding standards

The next generation video codec is not part of MPEG-4, but rather of a standard **MPEG-H** (H stands for heterogeneous environments), it is also referred to as **H.265** and is considered a next generation codec that could replace H.264. It's intended use case is **video streaming**. The goal is therefore to reduce the size of the video stream and improve its quality.

One of the main improvements is a **variable block size**, more specifically, a frame is divided into **coding tree units** (CTU), each of them is divided using a quadtree partitioning, each end node is a block of size 32x32, 16x16 or 8x8 pixels called a **coding unit** (CU). H.265 also improves intra-frame and inter-frame prediction. Due to increased complexity, the datastream in H.265 video codec is organized in a way that it supports **parallel decoding** of parts of the frame. All these improvements result in up to 50% smaller streams than H.264, but at the cost of significantly higher computational requirements (around 10x better hardware is required).

An upcoming improvement to H.265 is **H.266** (expected to be standardized in 2020). The standard is also referred to as Versatile Video Coding (VVC) or MPEG-I and introduces the following new aspects:

- Resolutions from 4K to 16K
- Support for omnidirectional videos
- High dynamic range support
- Auxiliary channels (depth, transparency)
- Variable frame rate (0 to 120 Hz)

Expected encoding complexity of H.266 is up to ten times that of H.265. It is also expected that the compression rate will be 30% to 50% better.

Image and video compression evaluation

Developing new compression techniques for multimedia is challenging, compared to lossless compression where the decoded data has to exactly match encoded one, the result of lossy compression has to only match good enough that the audience does not notice it. But qualitative evaluation with real people is time consuming and costly.
Because of this, quantitative evaluation is often employed to objectively evaluate progress of compression algorithms. Quantitative evaluation requires consistent use of the testing material (datasets) and evaluation measures that compare input and output and assess their similarity. Most frequently used evaluation measures are PNSR (Peak signal to noise ratio) and SSIM (Structural similarity index). The first one compares images pixel-to-pixel, the second one tries to compare more high-level properties and compares all pixels in a window of selected size.

Sound compression

Source material: Li and Drew, Fundamentals of Multimedia, Chapters 13 and 14

Sound is a one dimensional signal, therefore some techniques that are used to encode images and videos do not apply here. But there is still a lot of redundancy in realistic sounds, so there are a lot of options that can be used to compress raw audio data.

Lossless audio compression

Lossless audio codecs are not well known to an average music listener. They are important in case where saving a loss-free signal is important (saving for post-processing, data analysis) or to people that are distracted by lossy perceptual models that are suitable for a majority of people, or simply audiophiles, i.e. people who are interested in high-fidelity sound reproduction and seek to reproduce the sound of a live musical performance.

One of modern lossless audio codecs is **Free Lossless Audio Codec** (FLAC). It offers size reduction from 50% up to 80% (in some cases). The compression uses **model prediction** of the signal based on past values, followed by **encoding residuals** (difference between prediction and real signal using **Golomb-Rice codes**. This is then followed by run-length encoding. In case of stereo signal inter-channel correlation is also taken into account. The data is encoded in

multiple frames that are divided into subframes that contain short chunks of audio data. Subframes share some encoding parameters, while the parameters can change completely between individual frames. The coded supports several prediction models

- Zero - encoding digital silence, constant value predicted, encoded with RLE
- Verbatim - zero-order predictor, residual is signal itself, used when no model is suitable
- Fixed Linear - fitting p-order polynomial to p points, efficient algorithm
- FIR Linear - linear combination of previous samples, slower, diminishing returns

Golomb coding is a hybrid encoding technique that is efficient if small values dominate distribution. Each value is split in two parts (divided by parameter M), the quotient is encoded using **unary coding** (number Q is encoded by Q ones followed by a zero), the remainder using **truncated binary encoding**, a type of entropy coding suitable for uniform probability distributions (generalized way of coding a set of 2^N numbers). **Rice coding** is an efficient version of Golomb coding where M is 2^N, meaning that division can be run efficiently on digital computers.

Speech compression

There are several techniques designed especially for coding human speech and relying on its characteristics. **Non-linear signal quantization** takes into account that speech signals have a non-uniform amplitude distribution, i.e. small signal amplitudes are more likely than higher ones. Similarly, a **multi-channel encoding** encodes different frequency bands with different resolution, allocating more space for the frequency spectrum of human speech.

More sophisticated encodings model the speech using a **source-filter model**. Human speech is formed by air traveling from lungs to mouth, the vibration (excitation signal) is added by vocal fold opening and closing, and is *molded* by the vocal tract shape, e.g. mouth opening and closing, tongue moving. The configuration of the vocal tract changes the spectral shape of the excitation signal. Encoder for speech signal is also called a **vocoder** (voice + encoder). We will look at two examples of vocoders used in telephony.

The **Linear Predictive Coding** (LPC) vocoder encodes frames (parts of the signal, usually we use 30 to 50 frames per second) as parameters of a time-varying model of a vocal tract. Transmitted data is not a waveform, but rather a set of parameters (LP coefficients, signal gain, pitch and if a sound is voiced or unvoiced). Additionally, the model coefficients change slowly so they are predicted using a linear model of the previous coefficients (this is where the name LPC comes from, a LPC-10 vocoder uses 10 past coefficients). Thus, the actual size of the data transmitted is low, but the model assumes that the waveform is speech of a single person, anything else will be encoded poorly. Voiced and unvoiced phonemes are encoded differently, for voiced phonemes the source signal is a periodic function, while unvoiced phonemes use a noise wide-band noise generator. This signal is then shaped using model parameters. The **LPC-10** is intended for bandwidths of about **2.4 kbps**, which is suitable for **GPS mobile telephony**.

The **Code-excited Linear Prediction** (CELP) vocoder is a more complex vocoder, its primary purpose is still encoding of single-person speech, but it also performs well if the input sound signal is multiple people talking or arbitrary signal. The signal is encoded using **LPC** (short time prediction) as well as an **adaptive codebook searching** (long time prediction) that uses code books of waveforms. The closed loop process of optimizing the encoded form to be a perceptually best representation of input signal is called **analysis by synthesis**. The CELP vocoder is a part of **MPEG-4 audio compression** standard, it is intended for bandwidths of about **4.8 kbps**.

MPEG-1 audio compression

The MPEG-1 defines three audio codecs, also referred to as **layers** because they offer **downwards compatibility** (e.g. layer 1 stream can be decoded with a layer 3 decoder). Each layer requires a more complex encoder. The quality of the final audio depends on the available space (bitrate), this dictates the bit allocation and the resulting perceptual quality. Higher layer codecs will have a better perceptual quality at the same bitrate, but require more complex decoders.

Main compression concepts introduced by the MPEG-1 codec are tuned to exploit imperfections in the human auditory system to reduce the amount of encoded information. The core compression mechanism is called **frequency masking**. This psychoacoustic phenomenon arises when two close frequencies are registered at the same time. If one of them is louder it will mask the other one. The masking is more likely to be caused by a lower frequency that masks a higher one. Frequency masking is combined with **non-uniform quantization**, more precisely, the separation of frequency spectrum into **critical bands**. The bands are the result of grouping of hair cells in our auditory system that respond to a certain frequency range. Within a single band a strong frequency overwhelms cells and masks other frequencies within the same band. If a sound spans two or more bands it will be perceived as louder. Based on experiments it was determined that a human has about **24 to 25** critical bands. They are not distributed uniformly across the spectrum, we call this phenomenon perceptual non-uniformity. Below 500 Hz critical bands have approximately equal width of 100 Hz, above this threshold the bandwidth increases linearly.

Another psychoacoustic phenomenon that is taken into account is **temporal masking**. After a loud sound it takes time to hear a quieter sound because hair cells need a time-out as they are **saturated**. The duration depends on time and frequency similarity.

The most important aspect of lossy audio compression is how many bits are allocated to specific bands so that it will not impact the perceived quality of the signal. Bit allocation algorithm is not part of the MPEG-1 standard and may be done in several possible ways, depending on the encoder. Typically, the observations described in the previous paragraphs were measured in psychological experiments and are combined in a psychoacoustical model. Based on the input

signal the model computes masking levels for different frequencies. The frequency masking threshold computed by the model is combined by absolute hearing threshold into **global masking threshold** (GMT). The encoder then computes **Signal-to-Mask Ratio** (SMR) as the difference of Signal and GMT and **Mask-to-Noise Ratio** as difference of Signal to Noise Ratio (SNR) and SMR. These values are then used to determine how many bits can be used for a certain band. The SNR depends on the quantization level used for the band and therefore the number of bits used. We would like to get SNR that would be below GMT, however, this is not always possible. The distribution algorithm iteratively selects the band with the lowest difference between SNR and SMR and increases its bit allocation.

The **MPEG-1 Layer 1** audio codec uses 32 filters to separate signals into uniformly distributed bands. The frequency spectrum is obtained using a FFT on 8 ms blocks of the signal (for 48 kHz signal). Each frame contains 384 samples, 12 samples from each of the 32 filtered **sub-bands**. Samples are scaled and quantized using a simple perceptual model that uses only frequency masking. The Layer 1 codec was intended to be used for stored audio and is the simplest of the three layers. Perceptual tests have shown that Layer 1 achieves excellent performance at a stereo bit rate of 384 kbit/s.

The **MPEG-1 Layer 2** was designed for digital audio broadcast, it adds some improvements to reduce bitrate and improve quality at the cost of a more complex algorithm. It uses three sample groups together (3 times 12 samples), groups can share scaling factors and can be skipped if they do not contain information. The perceptual model also uses basic temporal masking in addition to frequency masking.

The **MPEG-1 Layer 3** was initially intended to be used for audio transmission over ISDN lines, but is nowadays primarily used for music storage (MP3 format). MP3 format has a near-universal hardware and software support, primarily because MP3 was the format of choice during the crucial first few years of widespread music file-sharing evolution. The main improvements of Layer 3 are the introduction of non-uniform critical bands, which makes the frequency spectrum more perceptually segmented. The frequency spectrum is obtained using a **Modified DCT**, which addresses the problem of boundaries that are caused by windowing the signal. The codec uses a perceptual model with both frequency and temporal masking and also takes into account redundancy of the stereo signal (either as joint signal with different intensities or by coding average signal and differences separately). The data is further compressed using Huffman coding. Layer 3 is the only layer that also supports **variable bitrate encoding** (VBR) which means that more bits can be allocated to the more complex segments of the signal and less for simpler segments in contrast to **constant bitrate encoding** (CBR) which is the only option in the Layer 1 and Layer 2 which limits the bitrate for all encoded signal blocks.

Stereo audio also offers a lot of opportunities for additional compression, the two channels are usually highly correlated. A simpler approach to stereo compression only encodes an average **intensity signal** together with **scaling factors** for both channels (same signal, different loudness). Better reproduction is achieved by compressing separately the **middle** channel

(average) and the **side** (difference between channels). Since the difference is usually small, it can be compressed efficiently.

MPEG-2 audio compression

The MPEG-2 introduces two audio coding specifications. In **Part 3** the MPEG-1 audio codecs are extended in a **backwards compatible** way to provide support for up to **5.1 channel audio** (5 + subwoofer), up to **7 additional language channels** and standard support for lower sampling frequencies.

Advanced Audio Coding (AAC), which is a part of **Part 7** is aimed at transparent sound reproduction in theaters. It is **not backwards compatible**, the algorithm changes some parts of Layer 3 encoder that were considered ad-hoc with more clean implementations (e.g. using pure MDCT). The final algorithm is more complex, but in general provides a better perceivable quality at the same bitrate than MPEG-1 Layer 3. AAC is used in DVD and related media.

Information retrieval

Multimedia data is acquired, processed and stored, but with the expansion of various multimedia archives, efficient access of it has also become of paramount importance. The field that deals with efficient access to data is called **information retrieval** and it primarily deals with querying text documents based on their content. We will therefore first explore this topic and then move on to retrieval of visual data, which bears some conceptual similarities with text. Lastly, we will look at how retrieval systems are evaluated.

Text retrieval

Source material: Manning et al., Introduction to information retrieval, Chapters 1 - 4, 6, 8, 9

The goal of text retrieval is to return relevant textual documents based on a search query. In most retrieval scenarios the textual data is not structured in any specific way (apart from the grammar of the specific language) and is therefore not suitable for direct retrieval using languages like SQL. The query can be based on simple expressions, e.g. which words should and should not be included in the documents or may be more complex. Information retrieval addresses the following questions:

- How to use language to specify what we are looking for? What do queries look like? There are multiple ways to look for the same content.
- How to match a query with documents in the database? How to efficiently match documents with a query as documents can be very large and numerous?
- How to optimize the query results for better experience? How to first return the most relevant results? How to iteratively improve results quality?

Since the key question in information retrieval is how to search large quantities (of data and also queries), we have to efficiently store relevant information so that we can quickly interpret queries. Besides the process of **querying** the dataset we therefore also have the process of **indexing** the documents, i.e. organizing them in a way that access to a subset of documents based on a given query is efficient and fast. The central component of information retrieval is called a **query engine**. The engine is a program that takes care of indexing documents, parsing user queries, matching documents to query specifications, ranking matched documents by relevance and returning results to the user.

Boolean expressions

The basic query scenario is that the user searches for documents that contain some words and does not contain other words. These kinds of queries are called Boolean expressions. Searching all documents would be time consuming even for such simple query setups. In order to efficiently index documents we have to keep a record of which documents contain which words. This information is stored in a binary matrix called the incidence matrix. Using the matrix we can simply check which columns (documents) match all required criteria words (rows) and return the list. However, this indexing approach is nonoptimal and requires a lot of memory. Since there are many rare words that do not appear in many documents, the incidence matrix is very sparse. It is therefore much better to only keep lists of documents that contain a certain word. A list of document IDs for a word is called an **inverted index** for that word.

Boolean expressions are combined with two main relations. The AND relation requires expressions at both its sides to be valid in order to keep the document as a valid result. In terms of inverted indices this means that we have to compute an intersection for both words. The OR relation requires that at least one of the sides is valid, which means that a union of inverse indices is computed. The execution of queries can be significantly optimized if we know the length of individual indices and execute their merging in the right sequence. Combining multiple AND relations is more optimal if we start with the two shortest indices as there is a large chance that the result will also be short. In case of nested OR relation, its length can be approximated with a sum of lengths of all included term inverted indices.

Documents and dictionaries

The capabilities of a search engine are largely dependent on how the documents are chosen and how they are indexed. The document unit choice influences the **granularity** of our search. Fine-grained search (small document units) will result in poor recall as we may lose important information that will not be retrieved by the query. Coarse-grained search will have good recall (desired information will be almost certainly in the retrieved documents), but since the documents are huge, these results will not be useful to a user. Document unit selection depends on the use case, sometimes indexing and searching at multiple granularity levels is also an option.

During **indexing**, i.e. the construction of inverted indices, each document is first split into a list of **tokens**, then **linguistic processing** and tokens **normalization** reduces the number of different words by mapping them to their common semantic root. Then, a list of normalized tokens is combined with document indices and the pairs are sorted alphabetically by token and grouped into lists for individual tokens, i.e. an inverted index. We also remember how many documents a term appears in as this will be useful when estimating its relevance.

Tokenization transforms text, which is represented as a sequence of bytes into individual atomic units called **tokens**. Tokenization has to deal with different encoding schemes, direction of text, etc. Some words can be written in various ways, e.g. with punctuations (e.g. abbreviations), some words are connected with hyphens or even merged together. Some languages do not even separate individual words so tokens have to be extracted using different means.

Words that occur very often in all the documents and do not carry any specific meaning are called **stop words**. These words are removed at the very beginning of the processing pipeline to save processing time, space and to improve accuracy. Lists of stop words are language specific. A good search engine takes care that in some cases stop words may also appear in phrases where the entire sequence is important and where stop words have retrieval value. This is why not all search engines use stop word lists.

Token normalization is another pre-processing step that is language specific. Same words can appear in different forms, lowercase or uppercase, accent marks, language variations and number or date formatting conventions. The task of normalization is to reduce the variations by identifying these similarities and replacing all equal tokens with a single representative. Related to normalization are also the processes of lemmatization and stemming. **Lemmatization** is a normalization process based on language rules, while **stemming** is a heuristic approach to normalization based on cutting parts of words that is faster but less accurate.

Word relationships

Searching for individual words can be useful, but many times we are interested in a sequence of words. How can only documents where words appear in a certain order be returned? There are two approaches to encode relationships into the index structure. The first one is called **bi-word index** and essentially means that pairs of tokens are indexed together instead of individual tokens. This approach is simple to implement, but it increases the size of the index with more terms and offers only limited relationships between terms in documents. The other option is called **positional index**. In this case each document index in a list for a term also contains a list of term positions where the term appears in the document. This approach is more complex, increases the size of individual inverted indices and final query execution time, but can support more complex relationships. In some cases both approaches are used in tandem, bi-word index is used for simple and common phrases, while positional index is used for other relationships.

Error tolerance and incompleteness

One of the main usability features in information retrieval is tolerance to query errors, e.g. spelling mistakes or incomplete queries. Incomplete queries are usually recognized by the wildcard symbol which denotes that anything can replace it so the terms are a combination of specified subsequence of letters and parts that can contain arbitrary letters. Normal indexes use a hash table, which is not suitable for wildcard queries. In some cases, e.g. searching for words with the same prefix, an ordered data-structure, e.g. a tree, can be used. Other techniques, like Permuterm or K-gram index can be used, however, processing wildcard queries is generally slower than processing simple Boolean queries.

- **Permuterm**: index contains all shifts of specific terms with a special symbol that denotes end of the word. With permuterm index we can only process single wildcard queries, this is done by adding the end symbol to the query term and shifting it so that the wildcard is at the end. Then the lookup is essentially a prefix matching operation.
- **K-gram index**: dictionary contains all substrings of length K, special sign is used to denote start and stop. Each substring is connected to the terms that include this substring. A query term is split in chunks of length K based on wildcard symbols, then each substring is searched for in the index. Finally, post-processing joins these results together. In cases where subsequences longer than K symbols are present in the query, an extra step has to check that the matched terms actually match all of them (brute-force string matching).

Error correction can be done based on each term separately. In this case words that are not in the dictionary are searched for the most similar words in the dictionary. Levenshtein distance or phonetic distance can be used to compute most similar terms. Additionally, context can be used for phrases, e.g. using grammar or frequently used phrases.

Ranking and comparing documents

Boolean queries can only determine if a document matches the query or not, this still generates a large list of documents that have to be assessed by the user. It is more convenient to rank the documents by their relevance to the query.

The basic idea of **relevance ranking** is that documents that include a queried term multiple times should be more relevant than those that include it less, we call this **term frequency** ranking. Collection of all words in the dictionary defines a vector space, every document is represented in this space as a point based on the number of times terms appear in it. This approach does not take into account the context of individual words (we call it a **bag-of-words** model), but can still give pretty good results in an efficient manner. To make the vector description even more robust to very frequent terms that are not informative, their influence is reduced by taking into account the term frequency in all documents, the **document frequency**. This way we get the “term frequency inverse document frequency” (tf-idf) scores, we multiply the frequency of a term with its inverse document frequency (logarithm of number of all documents

divided by the number of documents that the term appears in). This way the weight of the term is high if the term is frequent only in a small number of documents, and low if it is rare in a lot of documents or if it appears in almost all documents. The composite weight for a document that was matched by a query that includes multiple terms can be simply term weights for terms in a query summed together.

Comparison of documents can be then represented as a distance between points in the vector space. A frequent and efficient distance measure for comparing sparse description vectors (each point is essentially a vector) in a high-dimensional space is **cosine distance**, which is a dot product of normalized vectors (using Euclidean distance). This way, documents can be compared between themselves or to a query and ordered by their distance. To perform this kind of ranking in a retrieval system we have to store document frequency for each term as well as its term frequency for each document.

Relevance feedback

From the perspective of interaction with a query engine and the document corpus behind it, it is unlikely that the user will be familiar with all the documents in the dataset, therefore he/she does will not know how to specify a specific enough query (e.g. using synonyms, hypernyms, or even spelling errors). There are two approaches of making the database more accessible:

- Global: Expand query to as many possibilities with as many possible terms with error correction, synonyms, etc.
- Local: Based on interaction between the user and the system, the user is given a result and an option to select relevant and irrelevant documents from the list and improve the query this way. This approach is also called **relevance feedback** and is based on vector space similarities between documents.

In relevance feedback all documents are represented in a vector space, we know the query as well as which returned documents are relevant and which not. The task of the improvement algorithm is to find documents that are maximally similar to relevant results and minimally similar to irrelevant results. This estimate can then be improved in a feedback loop with several iterations. An example of an algorithm that does this is Rocchio algorithm. The algorithm is actually just a formula that takes a vector of the original query, a sum of vectors of relevant and sum of irrelevant documents and sums them together with predefined weights. The result is then used as a new query vector.

Relevance feedback loop can also be used to improve queries **without user interaction**, we call this **pseudo or blind relevance feedback**. In this scenario we use the default retrieval method method to find the most relevant documents, we assume that the K highest ranked documents are relevant and mark them as such, then we compute relevance feedback (e.g. with **Rocchio algorithm**) and improve the query results. Entire process can be repeated several times. Experiments have shown that this approach improves the relevance of the results even though the user is not involved in the process at all.

Evaluating retrieval systems

Objective evaluation of retrieval systems helps in understanding which approaches work and which do not. The main question that we are trying to answer in evaluation is how many of the retrieved documents are relevant? To evaluate a retrieval system we need a representative dataset that is annotated, i.e. we already know which answers we want for which query. For each query we can therefore look at results and count the number of documents among the returned documents that were expected (True Positives) and ones that were not (False Positives). We can also look at the remaining documents and check which documents were not returned, but should be (False Negatives) and which are correctly left behind (True Negatives). With this in mind we define two main measures:

- **Precision**: Percentage of relevant documents among retrieved documents, i.e. TP / (TP + FP)
- **Recall**: Percentage of returned relevant documents with respect to all relevant documents, i.e. TP / (TP + TN)

Precision and recall are related measures, without a significant improvement to the retrieval algorithm, we cannot improve both, but usually sacrifice one aspect for the other. Precision typically falls if the number of retrieved documents is increased, while recall increases if the number of retrieved documents is increased. To summarize both measures in a single score we use an **F-measure** (also called F1 score) which is a harmonic mean of both quantities (2 * Precision * Recall / (Precision + Recall)).

To extend the evaluation from unordered query results to relevance ranked query results we introduce a threshold on document similarity. This threshold decides how similar a document must be to a query to consider it relevant. Depending on the threshold we get different precision and recall values which can then be plotted as a plot in 2D space, a precision-recall curve. We can also compute Average Precision (AP) as an average of precision over multiple thresholds and Mean Average Precision (MAP) as an average of AP for multiple queries.

Another way to look at the performance of a retrieval system is using a Receiver operating characteristic curve (ROC) with respect to acceptance threshold. The curve displays true positive rate (sensitivity, TP / (TP + FN)) in relation to false positive rate (negative specificity, FP / (FP + TN)). The best operation point for the system (best compromise for a threshold) is the point on the curve with the minimum distance to point (0, 1). As an overall performance we can compute the **area under the curve** (AUC) which gives us an average over all thresholds.

Image retrieval

In comparison to text documents, where most tokens can be reasonably easily extracted from a text, images (as well as video and audio) are more challenging because extraction of **meaningful atomic units** is not an easy task. Many image retrieval systems therefore resort to

either **text metadata** accompanying the image (which is often incomplete) or **user annotations** (which are unreliable and scarce). Describing image content directly from the image requires not only decomposition of individual objects, represented by the image, but also their individual actions and relationships. A big problem is also perceptual relevance, In reality images are still described using simple low-level descriptions, e.g. color distribution, texture, or shape, while semantic approaches are still largely in the domain of academic research.

Before we review some primitive description techniques, let's review the basic idea of an image retrieval system. The concept is largely inspired by text retrieval. For each image that is entered to the database we extract a high-dimensional feature vector of its features, either basic or more complex. A query is also represented as an image and is also reduced to a vector description. Then all images in the database are compared to the query vector and ranked according to similarity, the best N of them are returned as a result. This is of course the most basic idea, a more complex system would also use indexed terms, which requires a more sparse semantic representation of content to work efficiently.

Primitive image descriptors

Source material: A. del Bimbo, Visual Information Retrieval

Primitive descriptions of image content are very close to pixels that compose it, we are talking about color of individual pixels, local texture, shape. The goal is to bring these low-level modalities to a **vector space** so that individual documents can be compared.

One of the simplest low-level representations is a distribution of **color**. Color of an image can be described as a **distribution** in color space, but more generally it is described with a color histogram, a non-parametric representation of the distribution that can be easily interpreted as a vector by normalizing the histogram (so that it is not dependent on the size of the image). Color histograms are also robust, they are **invariant** to change in size or rotation as well as partial occlusions. The downside of color histograms is that they do not contain any **spatial information** which makes them useless to encode relationships. Color histograms are also **sensitive to illumination changes**. This can be to some degree addressed by using a color space where chroma is separated from the luminosity and only comparing the chromatic components.

The **texture** is another property that can be easily converted to a vector description. Because of a lack of exact definition of texture there are many ways to describe it. We know that texture is loosely a description of the spatial arrangement of colors or intensities in an image or a selected region of an image, which means that a texture descriptor has to reflect this arrangement in some way. Another question regarding texture is the **level-of-detail**. Looking close enough in the scene we can find that most attributes that are considered texture are in fact shape properties. Texture can be described using spatial relationships of individual pixels, using

frequency analysis and using “high level” perceptual properties, such as periodicity, coarseness, dominant orientation, and complexity.

- **Cooccurence matrix** is a matrix that encodes how many times does pixel of value V1 appear next to pixel of value V2 in a specific pre-defined relationship. This matrix can be computed for regions of different sizes which means that an image is in fact a collection of cooccurence matrices. These matrices can be quite big so images are usually first quantized to a lower number of values per pixel. Various features can be computed for each matrix, e.g. energy, entropy, contrast, homogeneity, correlation. These quantities, taken from multiple matrices, then compose a description vector.
- **Local Binary Patterns** are another low-level texture descriptor. In this case each pixel gets a LBP value based on which of its eight neighbor pixels has a higher or lower value than itself (having eight neighbors means that the binary value obtained by comparison is again 8-bit). These LBP values are then summarized similar to a color, using a histogram.
- **Autocorrelation** is a measure of self similarity of an image patch, shifting pixels and comparing them to the original patch using normalized scalar product results in a response that can clearly show if the patch contains a repetitive structure as self-similarity of shifted patches results in multiple strong peaks.
- **Fourier transform** encodes image as a set of sinusoidal basis functions. In texture descriptors we are primarily interested in the energy of the frequency spectrum. The features that are extracted from the spectrum are usually sums of areas in frequency space, i.e. features are sensitive to certain parts of the spectrum.

Shape of objects is tightly connected to the concept of texture. Shapes are initially edges or binary masks that are encoded with a numeric representation, such as moments or differential codes. In some cases shape descriptors can be compared using normal vector similarity, another option is to compare it using transformation distance, e.g. the amount of transformation required to convert query shape to the given shape.

Since low-level representations, usually encoded as a histogram, do not convey spatial information well enough, the image is divided into multiple regions, a local histogram is computed for each region alone and the histograms are combined together using concatenation. A more advanced form of this technique is called **spatial pyramid**, a full image histogram description is concatenated by multiple levels of sub-divided image histogram descriptions. This is a compromise that encodes spatial information as well as the global location-invariant one.

Towards semantic description

A big step towards more sparse and semantic description of images came with the introduction of sparse local features that are clustered to combine their appearance variance with fixed dictionary size. The approach is generally called **bag-of-words**, local features are detected and

assigned to their corresponding words. Then the frequency of these words in the image gives us a descriptor vector that can be used similarly to the one in text retrieval. The advantage of this approach over primitive low-level features is that the process of acquiring (learning) the visual words gives us basic semantic information, some basic frequent parts emerge (e.g. wheels on a car) because they appear in the training images so often.

One of the early bag-of-words approaches used to detect stable regions, e.g. corners and blobs and describe their normalized (invariant to rotation and scale) local neighborhood with a **Scale invariant feature transform** (SIFT) descriptor. This descriptor divides the region into 16 sub-regions and computes histograms of quantized edge histograms in each of them for 8 orientations; a joint histogram has a standard size of 128 bins (16 x 8) and is normalized. An **unsupervised** approach is used to get a dictionary of words, descriptors are acquired from a set of training images and clustered using **K-means clustering**. This results in K word prototypes, when an image is presented to the system, the features are acquired and matched to the closest word. Then the words are counted in a form of a K-dimensional histogram.

Another approach to the construction of semantic features is to hierarchically combine edge features, detected using Gabor filters. These features are learned using their cooccurence in a training dataset. This way a multi-layer hierarchy is established that can (at top layer) describe very complex parts. Because the number of parts is again finite, a histogram of these features can be computed at top or multiple layers.

The last large conceptual jump in semantic image retrieval came in the form of end-to-end learning, which means that both the feature and a classifier are trained at once in a single framework that directly processes image pixels and outputs a high level decision about an image. This kind of learning required conceptual improvements, large quantities of data, and sufficiently powerful hardware. The conceptual improvements that were required for this leap came in the form of advanced and efficient artificial neural networks that were a well known method since the 1960s with their roots in biological systems. Recently these methods were upgraded with a convolutional approach to neuron weight sharing that improved performance in image classification tasks due to spatial invariance and reduction of the number of parameters. A convolutional neural network consists of a set of layers, each of them is processing the responses from the previous layer, the top layer is subjected to a cost function that compares its responses with the desired values that usually present a high-level concept, e.g. a category. The training of such a network is reduced to backpropagation of cost function errors down the layers, correcting the weights so that they would produce a response that is closer to the desired one. To make the learning process efficient, there are multiple additional layers that process responses in other ways, e.g. max-pooling, dropout, due to the brevity of this description we will not talk about them. The configuration and ordering of these layers is called a network architecture. Once such a network is trained for a sufficient amount of time, the weights start resembling local semantic parts.

The first use case for convolutional neural networks was object categorization and detection, however, high-level feature responses can also be used as vectors in retrieval systems. Additionally, a multi-object detector network can be used to index images using semantic words based on visual categories, e.g. by detecting objects on an image and use them in Boolean search scenario to speed up search, the index of categories provides an initial subset of candidate images that are then ranked based on vector similarity. Recently deep-learning approaches have also been used in even more ambitious retrieval scenarios, e.g. describing spatial relationships of objects and describing scenes as a whole.

Segmentation in retrieval

One of the important aspects of semantic image retrieval is correct decomposition of images. If we are searching for images with a certain semantic category, describing an entire image with a single descriptor makes this task hard due to the noise in the descriptor (background, other objects). We can describe only parts of an image, but we do not know in advance the number or shape of the regions. As we have mentioned in the previous section, we can use object detectors to detect objects of specific categories, but this can only be used for pre-trained categories.

Segmentation decomposes images into a set of regions. The division of regions can be based on simple low-level properties or can have some semantic qualities. The classical approach to segmentation uses unsupervised learning techniques to establish division, and the image is considered a set of elements with different properties that are clustered together. The key idea is to describe pixels or patches with attribute vectors that can then be used in clustering. These features can be very simple, e.g. color of a pixel or more complex descriptors of local texture.

- **Color** - each pixel is represented as a point in a color space, colors are then clustered.
- **Cooccurence** - cooccurence matrix was already described in previous section, the local vectors of matrix properties can also be used
- **Textons** - textons are descriptors that are learned from data, each pixel described with responses to a bank of filters (e.g. 24 filters). Then response vectors are clustered, the clusters are then representing individual textons (similar to words in bag-of-words idea). A local texture can be described with a histogram of textons - these histograms are descriptions that are clustered for segmentation.

When working with appearance information clusters will unlikely be spatially coherent, if this is desired then location information has to be encoded as a part of the description. This is the case in over-segmentation, i.e. superpixels, where the idea is to divide an image to small local units of multiple pixels that follow the structure of an image and can then be used for high-level processing (e.g. segmentation) instead of raw pixels.

For clustering features for segmentation, multiple methods can be employed. Generally the question is if we know the number of clusters in advance or if the method should decide on their count based on some kind of similarity threshold parameter. Some notable methods are:

- **K-means**: fixed number of clusters, defined with K. Cluster centers are randomly initialized, feature vectors are assigned to the closest cluster. Then the centers are computed again as the mean of all assigned feature vectors. This approach converges if the distance measure is Euclidean.
- **Mean-shift**: cluster consists of all feature vectors that converge to the same modus using mean-shift mode-seeking. Cluster number is determined automatically, but kernel bandwidth and type have to be provided. The kernel defines an attraction field, i.e. region in space where all points will have the same modus. Unfortunately the algorithm does not scale well to a high number of dimensions.
- **Affinity propagation**: the key feature of the algorithm is that it can work with an arbitrary similarity measure (affinity) between features. The affinity does not have to abide triangle inequality and does not even have to be defined for all feature pairs. The number of clusters is defined automatically, clusters are defined with exemplary features and not with mean feature vectors. The algorithm works in an iterative manner with messages passing between nodes in a graph.

The segmentation described so far could in theory segment images in different objects if their appearance was different, however, the segments were still just groups of similar pixels. In semantic segmentation groups are already assigned semantic categories. The most common view of semantic segmentation is pixel-wise categorization, i.e. each pixel is assigned a category. A classical example of this is using texton features together with a classifier that is trained to classify histograms of textons (bag-of-textons) into semantic categories.

As with classical object categorization, semantic segmentation was significantly improved with the use of convolutional neural networks. Instead of training for a single top-level decision, these networks are designed for per-pixel classification. The key idea is not to use any fully connected layers or treat image features as vectors, but to retain the spatial information until the top of the network. Because of this, segmentation networks are also called fully convolutional.

An issue that has to be addressed with respect to semantic segmentation with convolutional neural networks is how to obtain a full resolution segmentation if modern network architectures rely heavily on pooling layers where an entire response field is subsampled (reduced in spatial resolution) and only a maximum response in a local region is retained. These layers reduce parameter count and increase spatial robustness. By default the resulting segmentation mask is therefore very coarse. A simple solution to this is to use interpolation to increase it and potentially to use post-processing like Markov Random Field to improve local accuracy. Novel solutions also include dilated convolution layers that do not decrease resolution while still increasing spatial robustness and deconvolution layers that actually increase the spatial resolution. Deconvolution layers are used in encoder-decoder architectures where convolutional layers first encode visual information in high-dimensional, but low resolution vectors; deconvolutional layers then decode this information from high-dimensional feature space to per-pixel categorization.

Retrieval in video

Conceptually, retrieval in video can be very similar to retrieval in single images. Each frame is a document, while the entire video is a corpus. Additionally, video does not only contain still objects and object categories, but also short term actions and long term temporal relationships between entities. At the moment this kind of semantic data is still a matter of research and is not extensively used in retrieval.

Since the video is composed of many quite similar frames, it is much more efficient if we only index a subset of representative frames that we call keyframes. Keyframes can be detected based on predetermined shot boundaries, by taking their first, last or middle frame. Or can be obtained from “raw” video using clustering or subsampling. To ensure temporal consistency, features detected in keyframes can be tracked over several frames to ensure that they are consistent. Spatial consistency can also be employed in cases when a specific object is searched for (features should appear close to one another).

MPEG-7

In contrast to most other MPEG standards, the MPEG-7 does not describe video and audio codecs, but rather a multimedia content description standard. It is complementary to MPEG-4 and is meant to enable efficient access and manipulation of multimedia content by standardizing the elements of text-less object retrieval. It therefore standardizes the following components:

- **Object descriptors** - what and how can be described
- **Description schemas** - structure and semantics of the relations between its components
- **Description query** - how to search for content

The video and audio content can be described at a level of objects that are in a certain semantic relationship. The main problem of this idea is that the standard does not address the problem of semantic-gap, the large divide between low-level visual features and the meaningful high-level concepts that humans operate with.

Interactivity

One of the main concepts in modern multimedia systems is interactivity. Being able to communicate with a computer system and influence its behavior is compelling, but also poses certain additional requirements to the computer system in terms of speed and latency.

In this chapter we will look at two aspects of interactivity in a more narrow sense, we will focus on future emerging technologies. We will look at **augmented reality** as a challenging field of recognizing external environments and presenting additional content in it. We will also look at

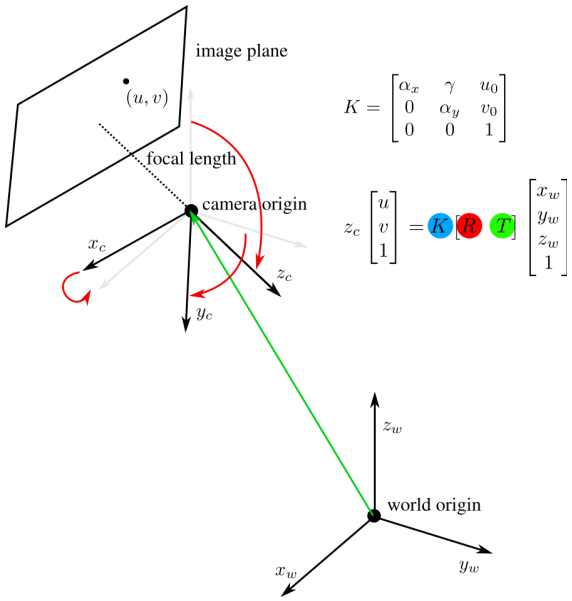
interactive surfaces as a way of providing new interactive experiences with display technologies.

Augmented reality

We perceive the world through our **senses**: sight, hearing, smell, haptics, balance. Since influencing sensory information changes our understanding of the world, we can say that reality is subjective. Augmented reality aims to **augment sensory information** so that the fused information is beneficial to our understanding of the world (or that it offers some other value, e.g. entertainment).

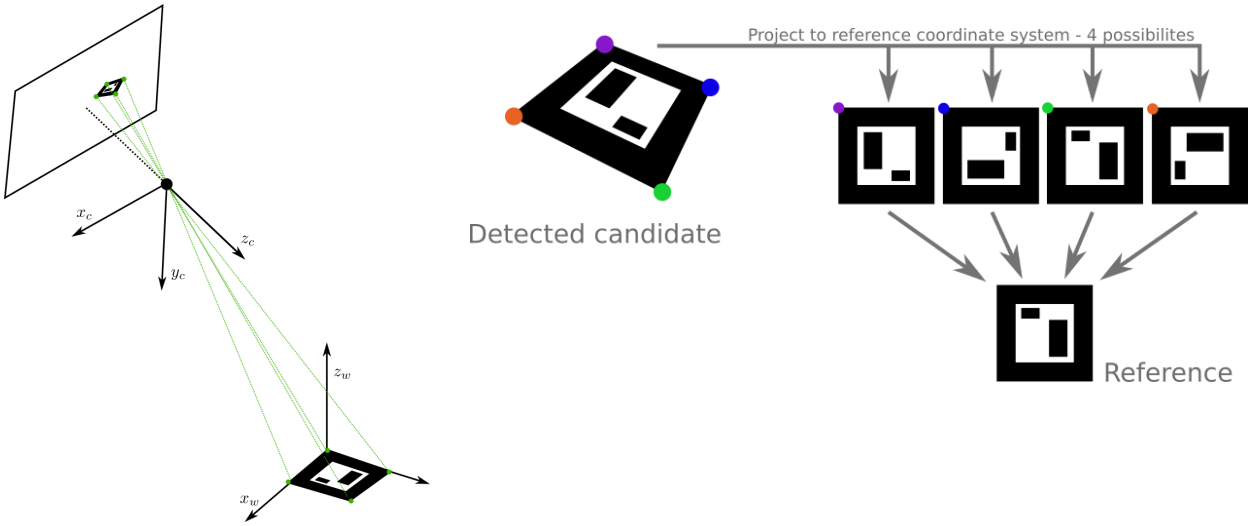
Even though humans possess several senses, we primarily rely on sight, which is perhaps also why augmentation is most developed in this direction. A basic visual augmented reality system has to determine the position of the camera and screen in the space in real-time to maintain the illusion that the augmented information is inserted into that space. There are many options how to achieve that from a camera alone, from computationally less intensive and more robust to more complex, but also more general. Additionally, camera positioning systems can also be aided by other sensors, such as GPS, WiFi positioning, IMU.

To position the camera in space we have to determine its extrinsic parameters: rotation and translation. This can be achieved by solving a linear equation system that includes reference data about the points in the world coordinate system and their projections in the image space, we call these pairs correspondences. Obtaining reliable correspondences can be a difficult task in general, this is why there exist different approaches that either increase reliability or know how to handle potentially unreliable correspondences.



Using markers

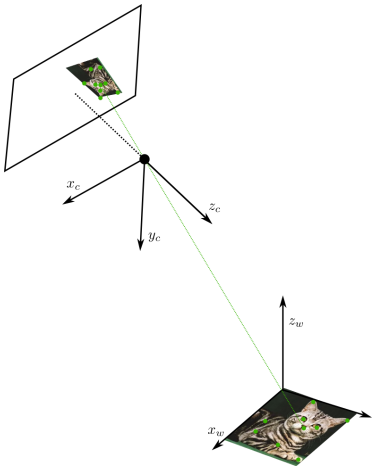
The simplest form of visual positioning that we will discuss is based on artificially created black and white markers that can be recognized with some basic image processing. Most commonly, these markers are square with a black border and an identifying interior that enables their recognition as well as disambiguates rotation. One such marker is enough to estimate camera position, but more can be used to improve robustness to occlusion and view direction.

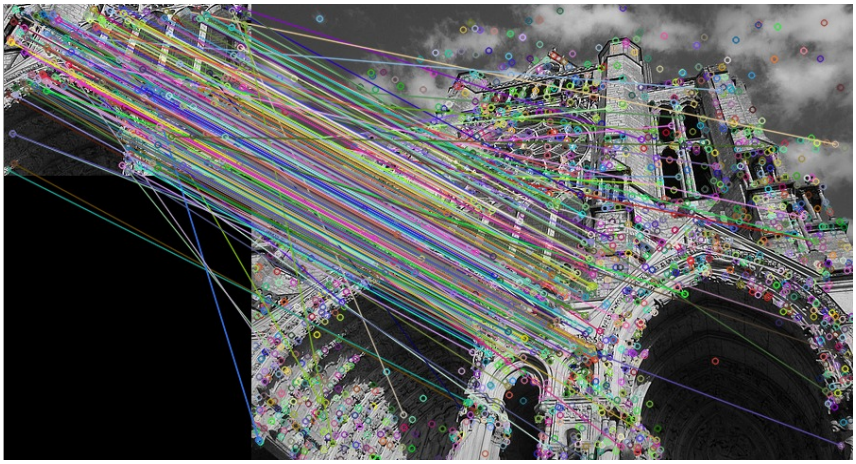


The problem of using markers (besides the fact that they have to be inserted into the scene) is that the algorithms that are used for their detection are not robust to occlusion.

Using templates

Instead of artificial markers, any textured surface can also be used for camera positioning. The concept also relies on detection of correspondences, however, the chance of having wrong correspondences in the set is much larger. Correspondences are detected by matching feature points (their local area description) from the reference template image to the query camera frame. To robustly estimate the camera extrinsic parameters from such a set, the RANSAC algorithm is used.





Building scene map

The most advanced approach for camera positioning that does not require any predefined markers is to build the 3D reference from the actual scene that we are tracking. This is also called VSLAM (Visual simultaneous localization and mapping) and can be used in many scenarios besides augmented reality (mobile robotics, 3D reconstruction). In principle, this kind of approach is computationally expensive since both camera location and map have to be updated. The approach is adapted to real-time augmented reality scenarios in the PTAM (Parallel tracking and mapping) algorithm that uses two threads, one for real-time responsive camera localization and the second one for slower map updating.

Interactive surfaces

Since the advent of the first iPhone, touch technology has become the de-facto way of interacting with multimedia systems. We will look at the technologies that make touch interfaces work and how different setups are applied to different usage scenarios. An interactive surface is composed of two components: a display and a sensor. These two components can be integrated together (e.g. smartphones) or placed at appropriate positions (e.g. projector and depth camera).

Designing a tabletop touch interface presents multiple challenges, from ergonomic issues (surface size and position, can lead to neck muscle strain or back problems) to usability issues (visibility and reachability in multi-user scenarios, use-cases, added value).

Resistive sensors

Resistive sensors are made from two clear layers coated with transparent conductive substances and an insulating layer between the both layers. A force applied to the outer layer squeezes the insulation and brings it closer to the opposite layer which can be detected as a changed resistance property of the system. The controller alternates between the layers, driving electric current on one and measuring the current on the other layer. This way detection of

horizontal and vertical position is then combined into full information. Resistive sensors are low-cost and have low power consumption, but can reduce the display quality of the overlaid display.

Capacitive sensors

There are two capacitive sensor approaches. In case of **surface capacitive** sensors, an uniform transparent conductive coating is placed on a glass panel, electrodes are positioned in each corner of the panel. The electrodes generate an **uniform electric field** across the conductive layer. When a finger (or other conductive object) touches the surface, electrical current will flow from the four corners through the finger. Ratio of the electrical current flowing from the four corners will be measured to detect the touched point. The measured current value will be inversely proportional to the distance between the touched point and the four corners. This approach has high positional accuracy, but it is difficult to detect multiple touches.

An alternative approach is called a **projected capacitive** sensor. In this case the surface is divided into a sensor **grid of electrodes**, each electrode covering only a small part of the surface. This approach enables accurate detection of multiple touch occurrences and has a high positional accuracy, but it is not suited for large panels because of slower transmission of electrical current over large distances. It is, however, the dominant technology on modern **mobile phones** and touch-capable laptops.

IR cameras

Since sensors based directly on electric phenomena do not scale very well, researchers have used other means of detecting touch, primarily using optics. A common approach is to use IR light since it can be detected by a (dedicated) camera and does not interfere with our visual perception. The diffused Illumination approach is the direct example of this, the semi-transparent projecting surface is illuminated from the rear with IR light sources. If a finger (or any other object) is placed on the surface, the IR light is reflected from it and is detected by the IR camera.

The FTIR (Frustrated Total Internal Reflection) improves touch reliability since it eliminates wrong touch detections where a finger is only close to the surface. The IR light is applied from the sides of an acrylic sheet and is trapped in the material due to TIR (total internal reflection). This effect is interrupted when an object touches the sheet from the side which causes the light to reflect from the touching object sidewise and is again detected by an IR camera.

Depth cameras

Another approach using optics that is increasingly used for HCI are depth cameras. Most of the cameras use IR light in some way to estimate depth from the sensor either from a projected pattern or using time-of-flight. Another option is using a stereo camera. Touch detection using depth information is more computationally intensive. The algorithm has to detect surfaces,

hands, fingers. However, this approach also offers additional HCI opportunities beyond finger touch detection, e.g. gesture recognition in 3D space above the interaction surface and object detection.