Design of Multimedia Applications

Part 1: Error Concealment in Digital Video

September 26, 2014

1 Introduction

Streaming of video to different devices over different types of networks has been a market of substantial growth during the past few years [1], [2], with devices ranging from connected televisions to smartphones and networks ranging from wired networks to wireless networks making use of Long Term Evolution Advanced (LTE Advanced). Many research and engineering efforts are currently directed toward optimizing the network transport of video streams. Indeed, present-day network technology is mostly IP-based, only offering a best-effort delivery service. No guarantees are for instance given about the timely delivery of packets from one network node to another network node.

Three common problems may occur during the network transport of video, namely bit errors, burst errors, and packet loss. The first two errors are caused by disruptions in the transport channels. The third error is typically caused by excessive data traffic, a problem that is also referred to as network congestion.

Compressed video only contains a minimum amount of redundant data, amongst others due to the elimination of spatial and temporal redundancy. This holds particularly true when making use of the newest standards for video compession, like H.264/AVC (Advanced Video Coding; [3], [4]) and H.265/HEVC (High Efficiency Video Coding; [5], [6], [7]). As a consequence, minor network errors can cause severe problems for the video sequence streamed [8]. For example, if (part of) an intra-coded frame is lost, the error may propagate throughout all inter-coded frames that refer to the damaged frame. If an error occurs during the transmission of crucial parameters (the resolution used, the entropy codec used, and so on), the entire video sequence may be lost.

In this lab session, we will study a number of state-of-the-art techniques for the reconstruction of lost information in video sequences, as used by visual content applications like video conferencing, video telephony, and live video broadcasting.

2 Compression of digital video

This section contains explanatory notes regarding the compression of digital video. The target audience are those people that have little to no knowledge of video compression. Note that some parts of this section have been simplified in order to allow for a quick understanding of the basic concepts of digital video compression.

2.1 Reasons for compression

An image is represented by a two-dimensional array of pixel values. The value of each pixel p can be represented as a vector $p[x, y]$, with x denoting a row of pixels and y denoting a column of pixels, describing the color of pixel p at the location $[x, y]$. As shown in Table 1, still images and (uncompressed) video sequences need a lot of storage and bandwidth. To mitigate storage and bandwidth requirements, image and video compression is used.

Two types of compression can be distinguished: lossless and lossy. Lossless video compression allows reconstructing a mathematically identical video sequence after decoding. This is a requirement often encountered in the area of computer graphics, medical imaging, digital cinema, and archiving. A disadvantage of lossless video compression is the low compression ratio.

Compared to lossless video compression, lossy video compression allows achieving higher compression ratios, meaning fewer bits are needed to represent the original video sequence. In this case, the decoded video sequence will not be exactly the same as the original video sequence. In most cases, however, human observers are hardly able to notice this difference, since the Human Visual System (HVS) is highly resilient against information loss.

2.2 Codec

A device or application that compresses a signal (two-dimensional in the case of still images, three-dimensional in the case of video) is called an encoder. A device or application that can decompress this compressed signal is called a decoder. The combination of encoder and decoder is generally denoted as a codec.

Table 1: Storage and bandwidth needs for images and video. Note that the peak download rate offered by LTE and LTE Advanced is about 300 Mbps and 3 Gbps, respectively.

Type	Resolution	Bits per pixel	Uncompressed size (B = byte)	Bandwidth (bps = bits per sec)
Image	640 x 480	24 bpp	900 KiB	
Video	640 x 480 (480p)	24 bpp	1 min video, 30 fps 1,54 GiB	221,18 Mbps
Video	1280 x 720 (720p)	24 bpp	1 min video, 30 fps 4,63 GiB	663,55 Mbps
Video	1920 x 1080 (1080p)	24 bpp	1 min video, 30 fps 10,43 GiB	1492,99 Mbps
Video	3840 x 2160 (2160p)	24 bpp	1 min video, 30 fps 41,71 GiB	5971,97 Mbps
Video	7680 x 4320 (4320p)	24 bpp	1 min video, 30 fps 166,85 GiB	23887,87 Mbps

2.3 Redundancy

Codecs are designed to compress signals containing statistical redundancy. An encoder takes symbols as input, and outputs encoded symbols that are referred to as code words. For example, the characters e and n occur more frequently in the Dutch language than the characters y and q. If a text file needs to be compressed, an encoder could represent the most frequent characters with a shorter code word than the least frequent characters (as in Morse code). Such a codec is called an entropy codec, and where the entropy of an image denotes a lower bound for the smallest average code word length using a variable length code. A low entropy for instance means that few bits are needed to code the image.

Still images and video sequences are difficult to compress by only making use of entropy coding. The latter is only effective when the input symbols are uncorrelated (that is, when the input symbols are statistically independent). This is hardly the case for still images and video sequences. It should for instance be clear that neighboring pixels in a still image or video frame have substantial visual resemblance. This type of resemblance is typically referred to as spatial redundancy. Accordingly, video sequences also contain temporal redundancy. Indeed, consecutive images in a video sequence often have large parts that are highly similar. Further, the HVS is more sensitive to low spatial frequencies than to high spatial frequencies. Therefore, high-frequency components can be removed from an image without the viewer noticing this. This is called spectral redundancy.

As a summary: video compression exploits statistical, spatial, temporal, and spectral redundancy in order to represent a video sequence with as few bits as possible.

2.4 Color spaces

A video sequence consists of a series of consecutive images. As explained above, each pixel is represented by a vector that holds color values. The RGB color space (Red-Green-Blue) is one of the most well-known color spaces. However, in this lab session, we will make use of the YUV color space, which is widely used in the area of video coding [9]. The YUV color space consists of three components: a luminance component (Y) and two chrominance components (U and V, also referred to as Cb and Cr, respectively).

Compared to the RGB color space, the main advantage of the YUV color space is that the chrominance components can be represented at a lower resolution, given that the HVS is less sensitive to changes in chrominance than to changes in luminance. Figure 1 visualizes the most common sampling formats.

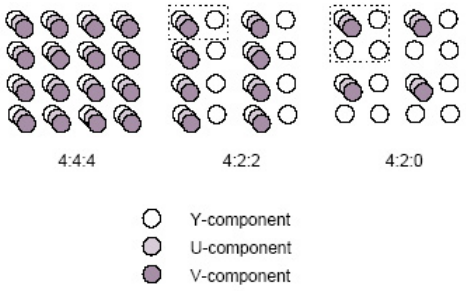


Figure 1: Sampling formats.

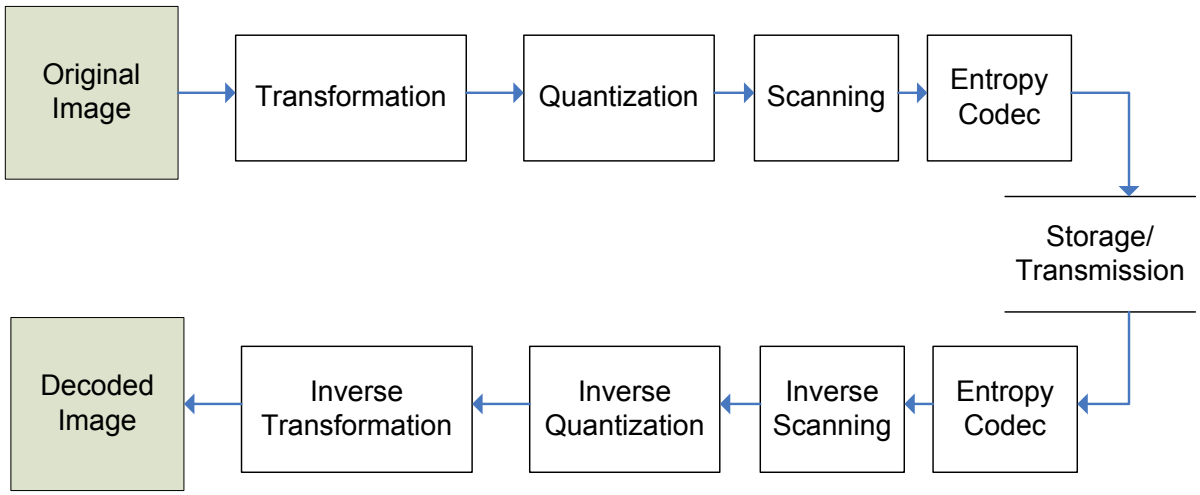


Figure 2: Scheme for encoding and decoding of still images.

- YUV 4:4:4: both the luminance and chrominance components are used at full resolution.
- YUV 4:2:2: only one U and V value is used for every two pixels.
- YUV 4:2:0: only one U and V value is used for each block of four pixels.

In this lab session, we make use of the (lossy) YUV 4:2:0 sampling format. This sampling format is frequently used in the context of consumer video.

2.5 Compression schemes for still images

Figure 2 shows a simple scheme for compressing still images. The most important steps are as follows: transformation, quantization, scanning, and entropy coding.

In the transformation step, the original image is transformed from the spatial domain to the frequency domain. This allows better localizing spatial and spectral redundancy, thus making it easier to remove redundant information.

Quantization represents the transformed coefficients with less precision, and thus with a smaller amount of bits. The quantization step is lossy. As a result, quantization will lower the image quality.

Scanning transforms the two-dimensional matrix representation of a quantized image into a one-dimensional vector representation.

The final phase, entropy coding, further compresses the video data. Specifically, the statistical redundancy between the different quantized values is found and a bitstream is generated that is suitable for storage or transmission. Entropy coding commonly makes use of Huffman codes, arithmetic codes, LempelZivWelch (LZW) compression, or simple run-length codes.

Each step of the encoding process is discussed in more detail in the following sections.

2.5.1 Image structure

A video sequence is a series of images that consist of three matrices containing pixel information. A first matrix holds luminance values, whereas the two remaining matrices hold chrominance values. Each image is further divided into a series of macroblocks. A macroblock consists of one matrix of 16x16 luminance samples and two matrices with chrominance samples. The number of chrominance samples is dependent on the sampling format used (e.g., 4:4:4, 4:2:2, or 4:2:0). Macroblocks are grouped into slices, and each macroblock can only belong to one slice. Partitioning an image into slices helps increasing the robustness against errors (among other functionalities). Indeed, an error in a slice cannot influence the decoding of the other slices of the image under consideration.

Figure 3 shows an example partitioning for a QCIF image (176x144 pixels). The image is divided into 99 macroblocks. The structure of one of these macroblocks is also shown in Figure 3. The sampling format used is 4:2:0, implying that the matrices holding chrominance values consist of 64 elements (8x8).

2.5.2 Prediction: DPCM (Differential Pulse-Code Modulation)

Spatial redundancy can be exploited by predicting a pixel value from one or more neighboring pixel values (rather than encoding each pixel value separately). Figure 4 shows how this is done for pixel X. One way to realize prediction is to simply take the value

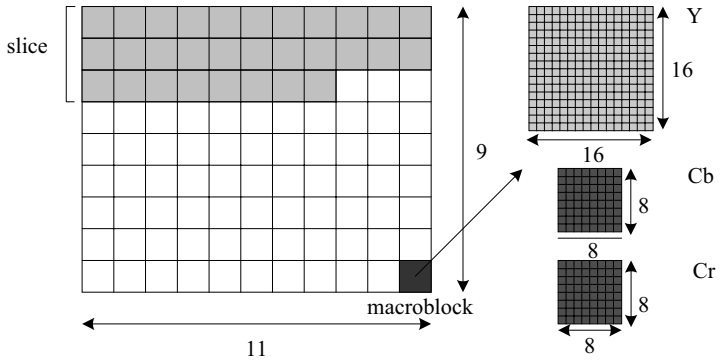


Figure 3: Division of an image into slices and macroblocks.

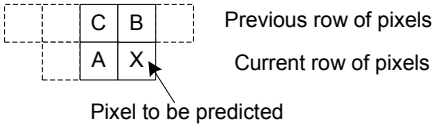


Figure 4: Prediction of a pixel value.

of the previously encoded pixel (pixel A). However, more effective prediction can typically be achieved by taking a weighted average of the values of multiple neighbor pixels that have been previously encoded (pixels A, B, and C). The original value of pixel X is subsequently subtracted from its predicted value. The resulting difference (i.e., the prediction error) can then be compressed effectively. Indeed, given the observation that prediction errors are typically small thanks to the presence of spatial correlation in an image, high compression can be achieved by representing small prediction errors with short code words and large prediction errors with long code words (as the former occur more frequently than the latter).

2.5.3 Transformation: DCT

Spatial correlation can also be removed by applying a two-dimensional Discrete Cosine Transform (DCT), transforming pixel values (or difference values) from the spatial domain to the frequency domain. To that end, an image is first divided in square blocks of pixels. Typical block sizes are 8x8 and 16x16. A DCT is then applied to each of these blocks, representing the content of these blocks as a linear combination (of a fixed set) of base functions. That way, the content of each block can be represented by a small number of transform coefficients that are visually important and a large number of transform coefficients that are visually less important. Typically, the coefficients that are visually

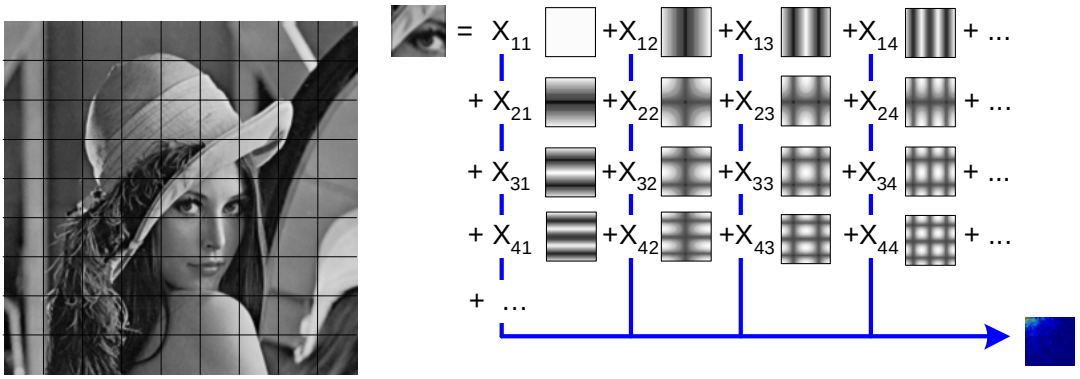


Figure 5: (left) Division of an image into macroblocks. (right) DCT: transformation of a macroblock into a linear combination of DCT base functions. Note that the X_{ij} represent the DCT coefficients (X_{11} denotes the DC coefficient).

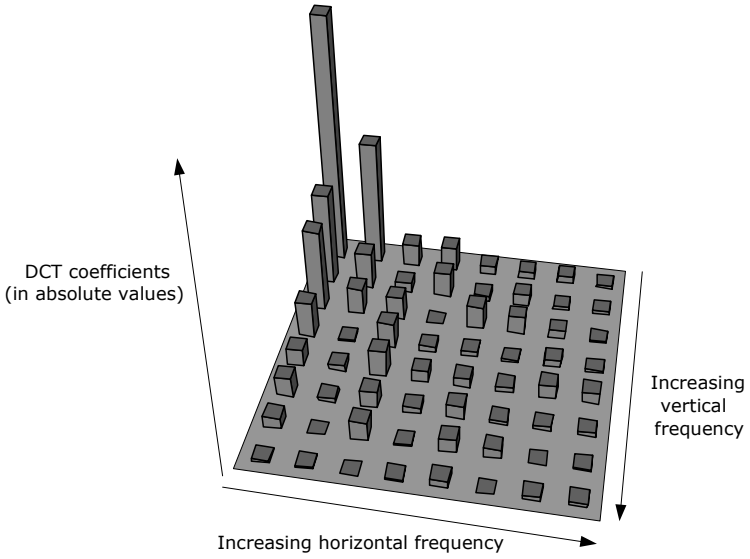


Figure 6: Example of a matrix of DCT coefficients (in 3-D) computed for an 8x8 block. The numerical values of the different DCT coefficients are given in the left table of Figure 7. The most and largest (absolute) values can typically be found in the upper left corner. The further away from this region, the higher the spatial frequencies (the latter are visually less important).

126	-49	43	-19	9	-10	6	-1
-65	19	-14	-1	3	2	0	-1
12	5	-12	13	-14	9	-10	0
-13	13	0	-3	6	3	1	1
5	3	-12	3	-5	-7	7	-4
-4	-6	9	1	-3	2	-5	0
4	-2	-4	-4	7	2	0	2
-1	-2	1	1	-6	-2	1	-2

31	-11	10	-4	2	-2	1	0
-16	4	-3	0	0	0	0	0
3	1	-3	3	-3	2	-2	0
-3	3	0	0	1	0	0	0
1	0	-3	0	-1	-1	1	-1
-1	-1	2	0	0	0	-1	0
1	0	-1	-1	1	0	0	0
0	0	0	0	-1	0	0	0

124	-44	40	-16	8	-8	4	0
-64	16	-12	0	0	0	0	0
12	4	-12	12	-12	8	-8	0
-12	12	0	0	4	0	0	0
4	0	-12	0	-4	-4	4	-4
-4	-4	8	0	0	0	-4	0
4	0	-4	-4	4	0	0	0
0	0	0	0	-4	0	0	0

Figure 7: (left) Original DCT coefficients; (middle) Coefficients after quantization; (right) Coefficients after inverse quantization.

the most important are located in the upper-left corner. The coefficient in the upper-left corner is called the DC coefficient, whereas all other coefficients are referred to as AC coefficients. The use of the DCT transform is illustrated in Figure 5 and in Figure 6.

2.5.4 Quantization

A DCT transforms pixel or difference values from the spatial domain to the frequency domain. The goal of quantization is to either remove transform coefficients that are visually less important or to reduce the precision of the aforementioned coefficients (by reducing the number of bits used to represent the values of the transform coefficients).

Two types of quantization can be distinguished: scalar and vector quantization. Scalar quantization treats each coefficient independently. This is in contrast to vector quantization, which is applied to a group of coefficients.

Figure 7 shows the principle of scalar quantization. Each coefficient is shifted two bits to the right (this is, each coefficient is divided by four). Note that quantization is lossy: inverse or backward quantization does not necessarily allow recovering the original DCT coefficients.

2.5.5 Scanning

Scanning processes all (quantized) transform coefficients according to a certain pattern. This pattern determines the order of the coefficients in the one-dimensional representation of a macroblock. The aim is to place the most significant coefficients in front of the one-dimensional representation, as well as to group zero-valued coefficients. This increases the effectiveness of run-level coding (see Section §2.5.6). A zigzag scan is commonly used (Figure 8).

Using a zigzag scan, the one-dimensional vector representation of the quantized macroblock shown in Figure 7 is as follows:

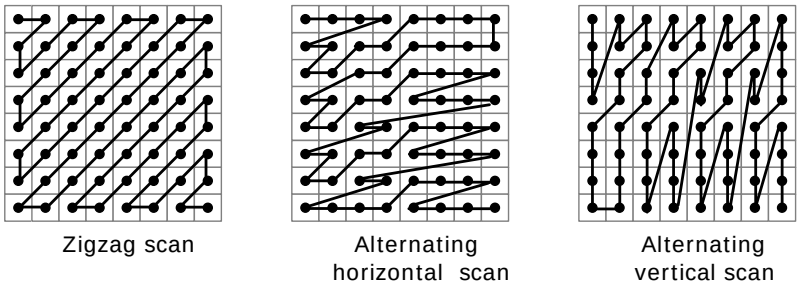


Figure 8: Possible methods to convert a two-dimensional matrix of DCT coefficients into a one-dimensional vector representation.

31, -11, -16, 3, 4, 10, -4, -3, 1, -3, 1, 3, -3, 0, 2, -2,
0, 3, 0, 0, -1, 1, -1, -3, 0, -3, 0, 1, 0, 0, 2, 1,
0, 2, 0, 0, 0, -1, 0, -1, 0, -2, 0, 0, 0, -1, 0, -1,
0, 0, 1, 0, 1, 0, -1, -1, 0, -1, 0, 0, 0, 0, 0, 0.

2.5.6 Entropy coding

After transformation and quantization, a macroblock consists of a small number of significant coefficients. The non-zero coefficients in the one-dimensional vector representation of a macroblock can be efficiently coded by means of statistical methods. We can distinguish two steps:

run-level coding The vector obtained after scanning is sparse, containing a substantial number of zeroes. This sparse vector can be efficiently represented by means of (*run*, *level*)-pairs. A *run* represents the number of consecutive coefficients with a value of zero (preceding the level), whereas a *level* represents the absolute value of a coefficient. The sign of the latter coefficient is coded separately.

The (*run*, *level*)-pairs for the example shown in Figure 7 are as follows (after applying a zigzag scan):
(0,31), (0,11), (0,16), (0,3), (0,4), (0,10), (0,4), (0,3), (0,1), (0,3), (0,1), (0,3), (0,3), (1,2), (0,2), (1,3), (2,1), (0,1), (0,1), (0,3), (1,3), (1,1), (2,2), (0,1), (1,2), (3,1), (1,1), (1,2), (3,1), (1,1), (2,1), (1,1), (1,1), (0,1), (1,1).

entropy coding (*run*, *level*)-pairs are subsequently processed by means of a statistical encoder. The entropy codec uses short code words for the most frequently occurring (*run*, *level*)-pairs, while less frequently occurring (*run*, *level*)-pairs are represented by longer code words.

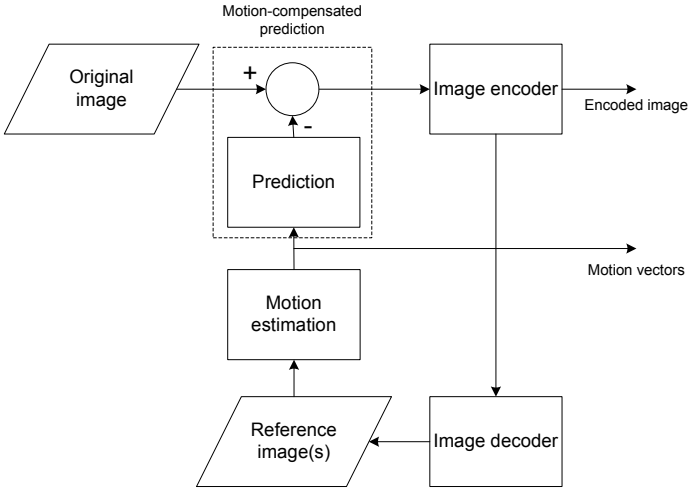


Figure 9: Video codec with motion estimation and motion compensation.

2.6 Compression schemes for moving images

A video sequence consists of a series of consecutive images. As described in Section §2.5, each image can be compressed separately. This is referred to as intra coding. However, higher compression rates can be achieved by taking advantage of information present in previous and following images (this is, by eliminating temporal redundancy). This is referred to as inter coding.

When making use of inter coding, the current image is first predicted based on reference images. Next, the current image is subtracted from the predicted image. The resulting difference image is then further processed by an intra codec. This is illustrated in Figure 9.

2.6.1 Motion Estimation (ME)

Reference images are images used for the purpose of prediction. Reference images can be the result of intra coding (intra-coded frames or I frames) or inter coding (predictively-coded frames or P frames). Prediction makes use of decoded images. That way, both the encoder and decoder predict frames based on the same values, thus preventing drift between the encoder and decoder (this is, preventing the introduction of additional prediction errors). Note that both previous and following images can be used for the purpose of prediction (bidirectionally-coded frames or B frames). This is shown in Figure 10.

For each block in the current image, the block most similar to the current block is sought in one or more reference images. This is the block that minimizes the differences with the current block. The position of the block found (x', y') is subtracted from the

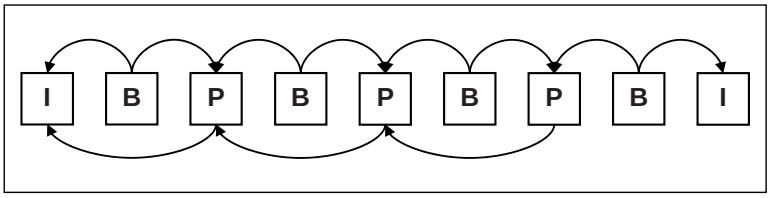


Figure 10: Temporal dependencies in a compressed video sequence.

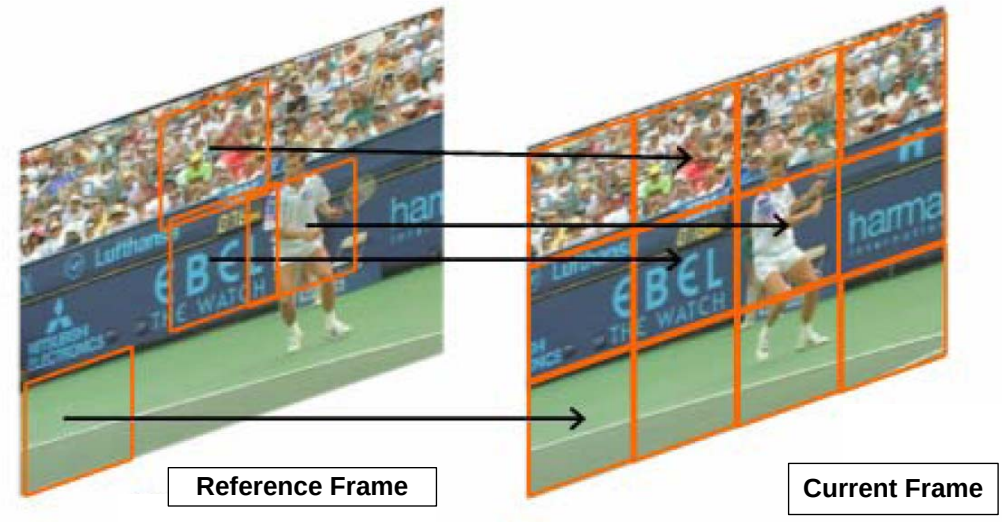


Figure 11: Motion estimation for P frames.

position of the original block (x, y). $(dx, dy) = (x, y) - (x', y')$ is called the motion vector of the current block. This principle is shown in Figure 11.

2.6.2 Motion Compensation (MC)

Using the motion vectors and the reference image, a prediction can be made of the current image. The difference between the original and the predicted image represents the prediction error, and this difference is further referred to as a difference image or a residual image. It should be clear that each inter-coded image is represented in a coded bit stream by means of motion vectors and an intra-coded difference image.

3 Error Resilience and error concealment

In the next sections, we discuss a number of straightforward spatial and temporal error concealment (or reconstruction) techniques. We also provide a non-exhaustive summary

of more advanced concealment techniques. The latter are usually able to obtain results that are visually more pleasing, but they come at the cost of a higher computational complexity (time and/or memory). This cost may for instance be prohibitive in the context of real-time video conferencing or live video broadcasting. For a recent overview of error concealment techniques, we would like to refer the interested reader to [10].

3.1 Active, passive, and interactive error concealment

Techniques for error concealment can be divided into three groups: active, passive, and interactive.

By choosing different coding configurations, possibly based on the network characteristics, an encoder can represent images in a more robust way. This is an *active* approach toward error resilience. A disadvantage of this approach is that it comes at the cost of an increased bit rate as robustness is typically facilitated by introducing redundancy.

Passive error concealment is done at the side of the decoder. Here, the decoder tries to reconstruct missing information.

Finally, when *interactive methods* are used for mitigating the impact of network errors, an encoder and decoder collaborate to optimize the quality of the video. For example, the decoder may send information to the encoder about the state of the network and packets lost. The encoder can subsequently use this information to alter the encoding process or to retransmit parts of the video sequence sent.

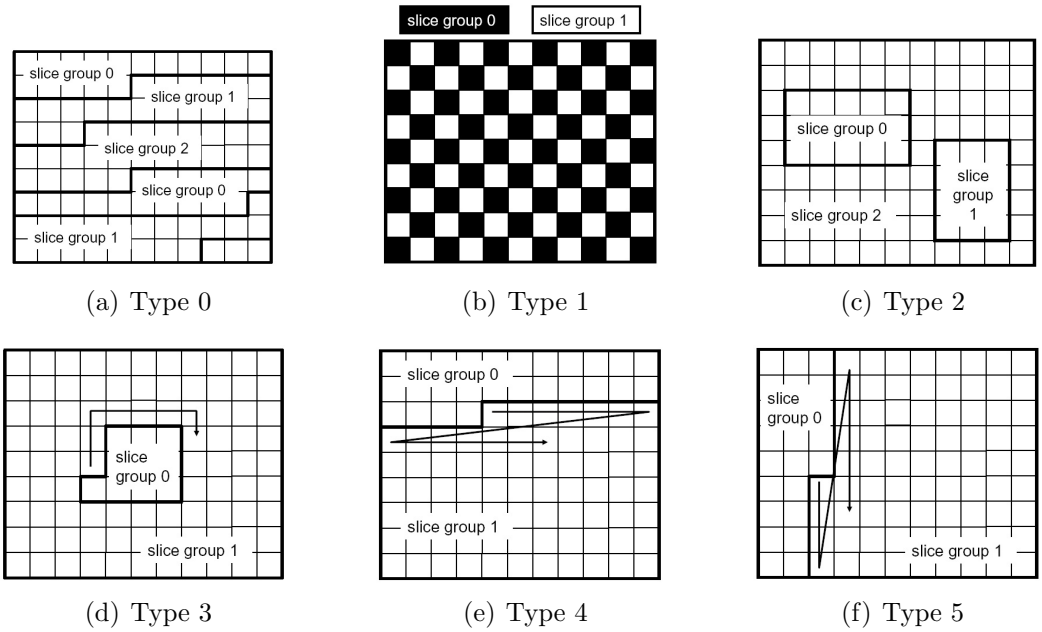
The main problem with interactive methods is the need for a communication channel. If the communication channel is slow or not reliable, the encoder may take incorrect decisions and even decrease the video quality.

In this lab session, we focus on passive error concealment. This approach is commonly used for dealing with errors: there is no need for additional communication channels and the encoder can optimally compress the video data.

Techniques for reconstructing lost parts make use of spatial and/or temporal information. The best results are generally obtained by techniques that combine both types of information in an adaptive way.

3.2 Flexible macroblock ordering

Many techniques for error concealment assume that neighboring macroblocks are available during the reconstruction of a missing macroblock. These techniques generally fail



(a) Type 0 (b) Type 1 (c) Type 2

(d) Type 3 (e) Type 4 (f) Type 5

Figure 12: Different types of FMO. Each macroblock is uniquely assigned to a so-called slice group by means of a macroblock allocation map (this map is transmitted from the encoder to the decoder as part of the header information of the compressed video sequence). By partitioning each slice group into several slices, and by subsequently transmitting each slice from the encoder to the decoder by means of a different network packet (among other possible packetization strategies), a higher level of robustness can be achieved against packet loss.

when connected macroblocks are lost. Tools like *Flexible Macroblock Ordering (FMO)* make it possible to code and transmit macroblocks in an order that is different from the conventional raster scan order used for coding and transmitting macroblocks, increasing the probability that connected macroblocks are still available after packet loss [11]. Figure 12 visualizes the different types of FMO that can for instance be found in the Baseline Profile and the Extended Profile of the widely used H.264/AVC standard.

3.3 Spatial error concealment

In order to reconstruct a lost macroblock, techniques for spatial error concealment make use of information present in neighboring (non-lost) macroblocks within the same frame.

3.3.1 Spatial interpolation

The most simple spatial reconstruction technique consists of interpolation based on the pixel values of the four surrounding macroblocks. Figure 13 shows how missing pixels can be reconstructed by means of spatial interpolation. The pixels at the border of the

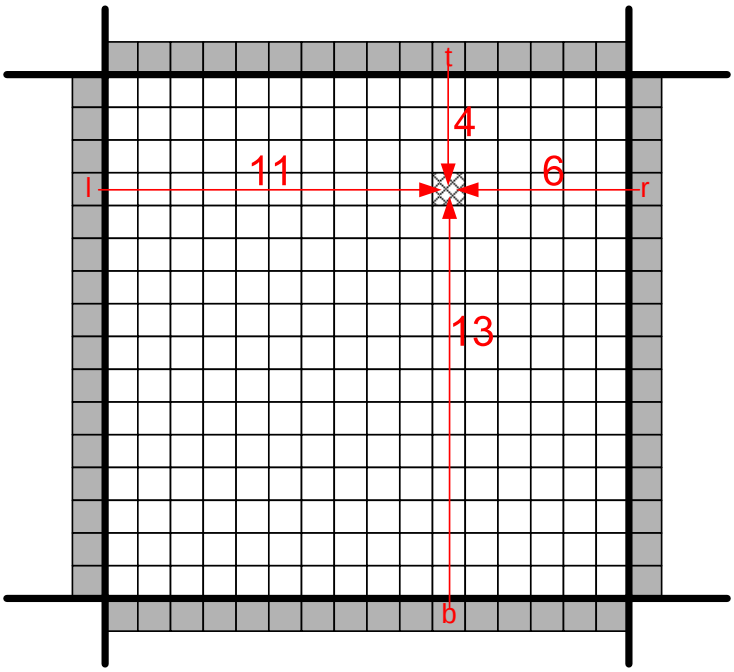


Figure 13: Simple spatial reconstruction using interpolation.

known macroblocks are called l, r, t, en b. The marked pixel can then be found using the following formula:

$$p = \frac{(17-11)l + (17-6)r + (17-4)t + (17-13)b}{(17-11) + (17-6) + (17-4) + (17-13)}.$$

3.3.2 More advanced techniques

More advanced spatial reconstruction algorithms often make use of edge detection. The edges in the surrounding blocks are calculated and used for the reconstruction of the content of the missing macroblock. Figure 14 shows this technique. Possible disadvantages of edge detection are the limited accuracy and the high computational complexity.

3.4 Temporal error concealment

In order to reconstruct a lost macroblock, techniques for temporal error concealment make use of information present in previous or following images. Additionally, motion information can be used to further enhance the error concealment.

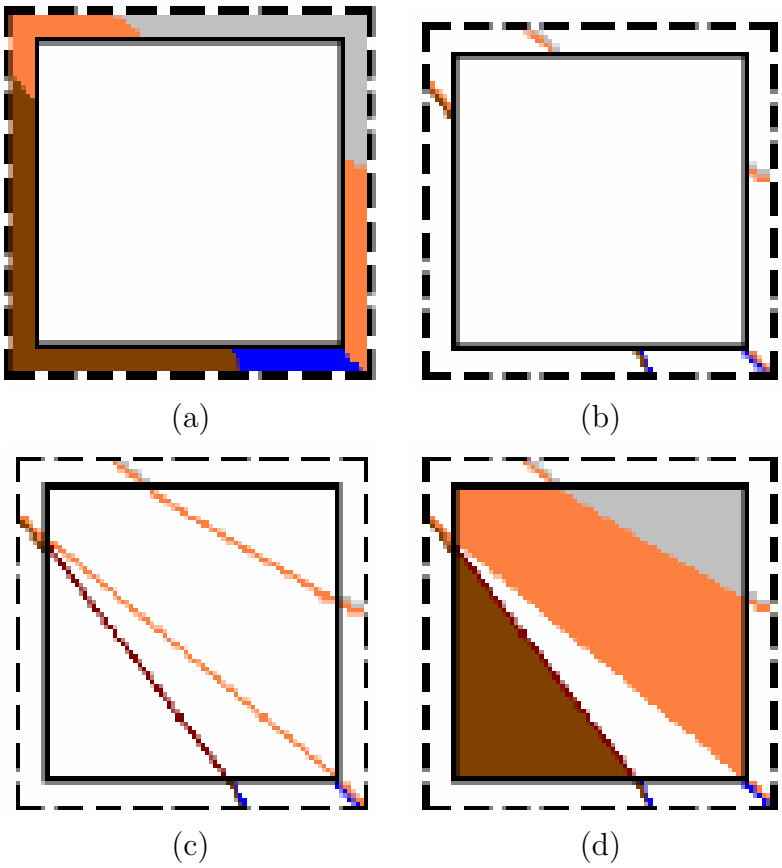


Figure 14: Spatial reconstruction using edge detection: (a) the border pixels surrounding a missing macroblock are analyzed, (b) an edge detection technique is applied to the border pixels, (c) the edges found are consecutively extended throughout the missing macroblock, (d) taking into account the edges found, spatial interpolation is performed.

3.4.1 Zero motion temporal error concealment

This is a relatively straightforward method that does not make use of motion information. To reconstruct a missing macroblock, a copy is made of the macroblock at the corresponding location in the previous frame.

3.4.2 Boundary matching spatio-temporal error concealment

This technique is more advanced and uses both temporal and spatial information. First, a motion vector is selected by taking advantage of the availability of the motion vectors of the neighboring macroblocks (up, down, left, and right). The selected motion vector is then used for the purpose of motion-compensated reconstruction.

To decide which motion vector is best, the boundary pixels of the surrounding blocks are compared with the pixels within the motion-compensated block. For each border pixel of the latter, the difference is calculated with the neighboring pixel in the surrounding blocks. The best motion vector is thevector that minimizes the sum of these differences.

Remark: to keep the computational complexity limited, the best motion vector is often determined by only making use of luma information.

3.4.3 More advanced techniques

More advanced reconstruction techniques in the temporal domain generally focus on finding the best motion vector for a missing macroblock. For example, using the median or average of the surrounding motion vectors may further improve the visual quality of error concealment. Another option is to use the motion vector of the macroblock at the corresponding location in the previous frame (assuming the co-located macroblock is available). Finally, even more advanced reconstruction techniques can make use of the motion vectors of different consecutive frames in order to define a motion trajectory that can be used to predict the current motion vector.

4 Video quality

4.1 Subjective versus objective quality assessment

Video quality assessment is currently a topic of high research interest and intense development [12], especially given the recent developments in the area of 3-D video consumption.

In order to assess the effectiveness of a video codec, *subjective* experiments can be performed using Mean Opinion Score (MOS) tests. With these test human observers are asked to compare the quality of a decoded video sequence to the quality of the original video sequence ('the proof of the pudding is in the eating'). However, experiments with human observers are often time-consuming. In addition, due to differences in the HVS of each person and decision fatigue, the judgements of human observers may not always be consistent, thus requiring testing for outliers. As a result, there is a strong demand for *objective* quality metrics. Ideally, an objective quality metric can be easily and automatically computed, while at the same time showing a strong correlation with the judgements of human observers [13] (e.g., measured by means of the Spearman or Pearson correlation coefficient). The definition of an objective quality metric that meets the aforementioned requirements is the goal of the *Video Quality Experts Group*, which is part of the *International Telecommunication Union* (ITU).

4.2 Peak Signal-to-Noise Ratio (PSNR)

PSNR is an example of an objective video quality metric, widely used for reasons of simplicity. Its computation is based on determining the Euclidean distance between a reference image and an altered version of the reference image, making this metric a so-called *full-reference* quality metric. The following equation shows how a PSNR value can be calculated (expressed in decibels):

$$PSNR\ (dB) = 10 \log_{10} \frac{(2^n - 1)^2}{MSE} = 20 \log_{10} \frac{(2^n - 1)}{MSE}.$$

The parameter n denotes the number of bits used to represent a sample (typically 8). The Mean Squared Error (MSE) for an image of dimension NxM is given by:

$$MSE = \frac{1}{(N * M)} \sum_{x=0}^{N-1} \sum_{y=0}^{M-1} (f(x, y) - f'(x, y))^2,$$

with f denoting the original image (i.e., the reference image) and with f' denoting the reconstructed image (i.e., the altered version of the reference image). When the MSE is calculated for two identical images, the result is zero. Further, PSNR is measured using a logarithmic scale.

In general, the quality of a video sequence is high if the PSNR is high and vice versa. Unfortunately, given that PSNR does not take into account properties of the HVS, a high PSNR value does not necessarily imply that the subjective quality of the video sequence is high.

4.3 Structural Similarity index (SSIM)

Lately, the SSIM [14] metric is gaining more attention in scientific publications. SSIM is an other objective quality metric which takes into account the perceived quality by humans. This is done by comparing two frames and evaluating the structural similarity between those frames. The structural similarity is considered as the similarity between adjecent pixels in the frame. This allows to differentiate the perceived errors since the similarity between pixels will be different near edges, objects, texture-rich areas, ... Hence, objectively quantifying the structural similarity should allow us to better estimate the effect of compression on the perceived quality.

SSIM is calculated between two windows (x,y) with size NxN and is given by:

$$SSIM(x,y) = \frac{(2\mu_x\mu_y + c_1)(2\sigma_{xy} + c_2)}{(\mu_x^2 + \mu_y^2 + c_1)(\sigma_x^2 + \sigma_y^2 + c_2)}.$$

With μ the average, σ^2 the variance, σ_{xy} the covariance of x and y, and c_1, c_2 are two variables to stabilize the division.

Typically, multiple windows are calculated across a frame. The resulting SSIM values are averaged to obtain the SSIM of the complete frame. To achieve a single SSIM value for a video sequence, the SSIM values are averaged for the complete sequence. If a high SSIM value is obtained, the perceptual quality is higher than when a low SSIM value is obtained.

5 Exercises

5.1 Instructions

The following five files need to be uploaded to Minerva, and in particular to the Dropbox module:

1. The three encoded video sequences (Exercise 1):
   `common_natural_<group number>.enc`, `common_synthetic_<group number>.enc`, and `group_<group number>.enc`.
2. All software, including the project and solution files (Exercises 2 and 3):
   `error_concealment_src_<group number>.zip`.
3. The report, in PDF or MS Word format (Exercise 4):
   `error_concealment_report_<group number>.pdf` or
   `error_concealment_report_<group number>.doc(x)`.

Deadlines

- **23 October 2014, 16h00 (Thursday)**: encoded video sequences and software (Exercises 1, 2, and 3).
- **30 October 2014, 16h00 (Thursday)**: report (Exercise 4).

Remarks

- We recommend solving the exercises in the correct order.
- Only the files requested need to be uploaded to Minerva. There is for instance no need to upload uncompressed video sequences.
- Please make sure that all files have been named correctly, and that all files have been uploaded to the Dropbox on Minerva.
- Grading takes into account the structure (e.g., usage of helper functions), readability, and documentation of the code written. Note that code needs to be documented in English.
- Pay close attention to the correctness of the computation of the macroblock and pixel indices (wrong indexing may easily result in the decoder crashing).

5.2 Background information

In this lab session, we will implement and evaluate some of the reconstruction techniques previously discussed. Section 5.4.1 to Section 5.4.3 deal with spatial reconstruction techniques, whereas Section 5.5.1 to Section 5.5.3 deal with temporal reconstruction techniques. Once the necessary techniques have been implemented in the test provided framework, experiments need to be performed in order to investigate and compare the effectiveness and efficiency of the different reconstruction techniques (see Section 5.6.2).

To keep the test framework simple and comprehensive, we have chosen not to simulate transmission errors by damaging the encoded bit stream. In that case, the damaged bitstream would not comply any more with the standardized syntax, preventing simple codecs from successfully decoding the video stream. In principle, a decoder can be created that takes this into account. However, the design of such a decoder is not trivial and is beyond the scope of this series of exercises. Therefore, transmission errors are simulated at the level of the decoder itself.

Given a correct bit stream, every frame is fully decoded. Next, transmission errors are simulated by removing certain macroblocks. The resulting frame is then corrected by making use of a particular reconstruction technique. Finally, the reconstructed frame is stored and used as a reference frame for the next inter-coded frame.

Transmission errors are simulated by means of an error pattern, reflecting the use of a particular type of FMO. This error pattern can be found in a text file, containing a line for each image. The first number on each line corresponds to the frame number. The subsequent numbers, if present, denote macroblock numbers. Each macroblock number listed corresponds to a macroblock lost.

The syntax of the decoder software is as follows:

```
decoder.exe <inputfile> <outputfile> <error_pattern> <conceal_method>
```

where `inputfile` refers to the encoded video file, `outputfile` is the name of the decoded YUV file, `error_pattern` denotes the error pattern (the decoder does not check the correctness of the file name of the error pattern!), and `conceal_method` is a number between 0 and 5, indicating which reconstruction technique needs to be used.

The API of the test framework, needed to solve the exercises, can be found in Figure 15.

class: **Frame**		
	public methods	
	int getWidth()	Returns the width of the frame (in macroblocks)
	int getHeight()	Returns the height of the frame (in macroblocks)
	int getNumMB()	Returns the number of macroblocks in the frame
	bool is_p_frame()	Returns *true* if the frame is a P frame, *false* for an I frame
	Macroblock* getMacroblock(int index)	Returns the macroblock with macroblock number *index* (in raster scan order)
class: **Macroblock**		
	public attributes	
	pixel luma[i][j]	Value of the luma component of the pixel at row *i* and column *j*
	pixel cb[i][j]	Value of the chroma (blue) component of the pixel at row *i* and column *j*
	pixel cr[i][j]	Value of the chroma (red) component of the pixel at row *i* and column *j*
	MotionVector mv	Motion vector corresponding with the macroblock (only for P frames)
	public methods	
	int getMBNum()	Returns the macroblock number (index in raster scan order) of the macroblock in the frame
	int getXPos()	Returns the column number of the macroblock in the frame (in terms of the number of macroblocks)
	int getYPos()	Returns the row number of the macroblock in the frame (in terms of the number of macroblocks)
	bool isMissing()	Returns *true* if the macroblock is not available, *false* if the macroblock is available
	void setConcealed()	Marks the macroblock as being reconstructed by changing the value of the flag isMissing from *true* to *false* (so the macroblock is again available for further use)
struct: **MotionVector**		
	public attributes	
	int x	Horizontal component of the motion vector
	int y	Vertical component of the motion vector

Figure 15: API of the test framework, needed to solve the exercises. All indices, macroblock numbers, and row and column numbers start from zero. A `pixel` is defined as an `int` (through the C++ `typedef` operator). A motion vector with an example value of (-2, 4) represents an offset that is valid for all pixels in the macroblock the motion vector belongs to: 2 to the left, 4 to the bottom.

5.3 Creation of bitstreams

5.3.1 Exercise 1: Creation of bitstreams

A simple encoder has been made available for the encoding of the original video sequences.

The syntax of the encoder is as follows:

```
encoder.exe <inputfile> <input_width> <input_height> <qp> <I-interval>
<output_file>
```

`Inputfile` is the original YUV file. `Input_width` and `Input_height` are the width and height of the video, expressed in terms of the number of macroblocks, respectively. Note that a macroblock typically consists of 16x16 pixels. Next, the `QP` denotes the Quantization Parameter. A high QP means that the video is quantized strongly, resulting in a lower bit rate (and perceptual quality). `I-interval` determines the number of inter-coded frames between each intra-coded frame. Finally, `output_file` is the file name of the encoded video.

Each group has to encode three bitstreams:

- A first *common bitstream* for all groups, containing natural content. This bitstream must be named
  `common_natural_<group number>.enc`.
- A second *common bitstream* for all groups, containing synthetic content. This bitstream must be named
  `common_synthetic_<group number>.enc`.
- A *specific bitstream* for each group. This bitstream must be named
  `group_<group number>.enc`.

Consult the Documents module on Minerva in order to find the uncompressed video sequences, the error patterns, and the coding parameters to be used.

5.4 Spatial error concealment

5.4.1 Exercise 2.A: Simple spatial reconstruction

Complete the method `conceal_spatial_1` in `ErrorConcealer.cpp` by implementing the spatial reconstruction technique described in Section 3.3.1. Provide support for timing the execution of this method.

For this exercise, you may assume that no two neighboring macroblocks have been lost at the same time. In other words, all neighboring macroblocks (top, bottom, left, and right) are completely available during the reconstruction of the current macroblock. Note that this does not hold true for macroblocks belonging to the borders of the corrupt frame under consideration.

This method can be tested by invoking the decoder with parameter `conceal_method` equal to `0`. For this exercise, use error pattern `error_pattern_simple` (FMO Type 1).

5.4.2 Exercise 2.B: General spatial reconstruction

Complete the method `conceal_spatial_2` in `ErrorConcealer.cpp`. For this method, which extends the previous method, it can no longer be assumed that the neighbors of each macroblock lost are available. Provide support for timing the execution of this method.

This method can be tested by invoking the decoder with parameter `conceal_method` equal to `1`. For this exercise, use error pattern `error_pattern_complex` (FMO Type 0).

5.4.3 Exercise 2.C: General spatial reconstruction using edge information

Complete the method `conceal_spatial_3` in `ErrorConcealer.cpp`. The error concealment technique implemented in this method, which builds on top of the previous two methods, needs to take advantage of edge information. How to take advantage of edge information and how to deal with the different problems encountered is at your discretion. More advanced methods will result in better quality and will also be graded higher. Provide support for timing the execution of this method.

This method can be tested by invoking the decoder with parameter `conceal_method` equal to `2`. For this exercise, use both error pattern `error_pattern_simple` (FMO Type 1) and error pattern `error_pattern_complex` (FMO Type 0).

5.5 Temporal error concealment

5.5.1 Exercise 3.A: Zero motion temporal reconstruction

Complete the method `conceal_temporal_1` by implementing zero motion temporal error concealment (as described in Section 3.4.1). Provide support for timing the execution of this method.

This method can be tested by invoking the decoder with parameter `conceal_method` equal to `3`. For this exercise, you can use both error patterns available, given that this method does not use information from surrounding macroblocks in the corrupt frame under consideration.

5.5.2 Exercise 3.B: Simple spatio-temporal boundary matching reconstruction

Complete the method `conceal_temporal_2` in `ErrorConcealer.cpp` by implementing the spatio-temporal boundary matching technique discussed in Section 3.4.2. It can be assumed that no two neighboring macroblocks have been lost simultaneously. Provide support for timing the execution of this method.

Remark: when the lost macroblock belongs to an intra-coded frame, no motion vectors are available. In this case, zero motion temporal reconstruction can be applied.

This method can be tested by invoking the decoder with parameter `conceal_method` equal to `4`. For this exercise, use error pattern `error_pattern_simple`.

5.5.3 Exercise 3.C: General temporal reconstruction

Complete the method `conceal_temporal_3` in `ErrorConcealer.cpp`. This implementation extends the previous method. In this case, it can no longer be assumed that neighboring macroblocks are available. Provide support for timing the execution of this method.

The way in which to solve this exercise can be chosen freely. Again, more intelligent (content-adaptive) methods will result in better quality and will also yield better grades.

This method can be tested by invoking the decoder with parameter `conceal_method` equal to `5`. For this exercise, use error pattern `error_pattern_complex`.

5.6 Evaluation and report

5.6.1 Quality measurement

Measuring the quality of a decoded video sequence can be done by means of the VQMT tool available on Minerva. This tool generates both the PSNR and SSIM, and is used as follows:

```
VQMT.exe [OriginalVideo] [ProcessedVideo] [Height]
                [Width] [Frames] [Output] PSNR SSIM
```

`OriginalVideo` refers to the original uncompressed video sequence, `ProcessedVideo` denotes the decoded video sequence, and `width` and `height` refer to the width and height of the video sequence (in pixels), respectively. `Output` specifies the file to which the PSNR and SSIM of each frame is writen (resp. output_psnr.csv and output_ssim.csv). Averaging the PSNR and SSIM values will result in a single PSNR and SSIM value for the sequence. For more information on efficient processing (e.g.: Python) of these files, we refer to the practical sessions of the courses Multimedia and Multimedia Technologies.

5.6.2 Exercise 4: Evaluation and report

Use the VQMT tool to evaluate the effectiveness of the different reconstruction techniques implemented. To accomplish this, calculate and compare the PSNR and SSIM for the reconstructed video sequences by using both the simple and complex error patterns. Obviously, the complex error pattern can only be used by the methods supporting it. For these measurements, you should use both the common bitstreams and the bitstream specifically assigned to your group. Similarly, use the time measurements to evaluate the time complexity of the different reconstruction techniques implemented.

The results obtained should be described in a clear report containing:

1. Pseudocode and explanatory notes for the methods that have been implemented in Exercises 2.B, 2.C, and 3.C, paying particular attention to the way edge information and content-adaptivity were leveraged.
2. The rationale of why these methods were chosen, and a discussion of their advantages and disadvantages.
3. The results of all quality and time measurements (provide information about the PC configuration used in order to put the time measurements in context).
4. A comparison of the different results, as well as an explanation of the relation between the results.
5. Conclusions regarding the comparison of the different results (for the spatial and temporal techniques separately, as well as for the comparison of both approaches).
6. An answer, with explanation, to the following questions:
   (a) Can a clear winner be found among all techniques and groups of techniques?

- If so, what would be a possible reason for *not* choosing the best candidate (thus, choosing another technique)?
- If not, which factors decide which technique is best?

(b) When you do a visual inspection of the reconstructed sequences, do you get the same results?

- If so, what conclusions can be drawn from this experiment?
- If not, what are the main differences and what can be concluded?

7. What would be the advantages and disadvantages of client-side buffering?

8. Given the complex error pattern, what is the highest PSNR value obtained for the entire *Bourne Ultimatum* video sequence (calculated as the average of all frame PSNR values), for Exercise 2.B, Exercise 2.C, and Exercise 3.C? Plot the different PSNR values in a **summarizing table**.

9. Is there a clear correlation between the PSNR and SSIM; and does the SSIM better match the perceived quality.

Make sure that the report is well-structured and well-written, and that the report answers all of the questions listed above. Note that the use of bullets may be helpful in structuring the answers to the questions listed above.

The number of pages is limited to six (a seventh page will not be read!). So, keep the report focused and concise. Note that supportive screenshots can be added as an appendix (this is encouraged), and that this appendix does not add to the page count (e.g., screenshots illustrating the positive and/or negative impact of edge information usage).

References

[1] Cisco Visual Networking Index: Forecast and Methodology, 2013-2018. Technical report, Cisco, Available on `http://www.cisco.com/c/en/us/solutions/collateral/service-provider/ip-ngn-ip-next-generation-network/white_paper_c11-481360.pdf`, 2014.

[2] YouTube. YouTube Statistics. Available on `http://www.youtube.com/yt/press/statistics/`, 2014.

[3] Thomas Wiegand, Gary J. Sullivan, Gisle Bjøntegaard, and Ajay Luthra. Overview of the H.264/AVC Video Coding Standard. *IEEE Trans. Circuits Syst. Video Technol.*, 13(7):560–576, July 2003.

[4] Gary J. Sullivan and Thomas Wiegand. Video Compression - From Concepts to the H.264/AVC Standard. *Proc. the IEEE, Special Issue on Advances in Video Coding and Delivery*, 93(1):18–31, January 2005.

[5] Joint Collaborative Team on Video Coding (JCT-VC). Official HEVC Website. Available on `http://hevc.info/`, 2012.

[6] Gary J. Sullivan, Jens-Rainer Ohm, Woojin Han, and Thomas Wiegand. Overview of the High Efficiency Video Coding (HEVC) Standard. *IEEE Trans. Circuits Syst. Video Technol.*, pages 1–19, December 2012.

[7] Jens-Rainer Ohm and Gary J. Sullivan. High Efficiency Video Coding: The Next Frontier in Video Compression [Standards in a Nutshell]. *IEEE Signal Processing Magazine*, 30(1):152–158, January 2013.

[8] Thomas Wiegand and Gary J. Sullivan. The Picturephone Is Here. Really. *IEEE Spectrum*, 48(9):50–54, September 2011.

[9] Gary J. Sullivan and Stephen Estrop. Recommended 8-Bit YUV Formats for Video Rendering. Technical report, Microsoft Corporation, Available on `http://msdn.microsoft.com/en-us/library/windows/desktop/dd206750(v=vs.85).aspx`, 1998.

[10] Martin Fleury, Sandro Moiron, and Mohammed Ghanbari. Innovations in Video Error Resilience and Concealment. *Recent Patents on Signal Processing*, 1(2):124–134, December 2011.

[11] Peter Lambert, Wesley De Neve, Yves Dhondt, and Rik Van de Walle. Flexible Macroblock Ordering in H.264/AVC. *Journal of Visual Communication & Image Representation*, 17:358–375, January 2006.

[12] Zhou Wang and Alan C.Bovik. Mean Squared Error: Love It or Leave it? A New Look at Signal Fidelity Measures. *IEEE Signal Processing Magazine*, 26(1):98–117, January 2009.

[13] Rosario Feghali, Filippo Speranza, Demin Wang, and Andre Vincent. Video Quality Metric for Bit Rate Control via Joint Adjustment of Quantization and Frame Rate. *IEEE Transactions on Broadcasting*, 53(1):441–446, March 2007.

[14] Zhou Wang, Alan C. Bovik, Hamid R. Sheikh, and Eero P. Simoncelli. Image Quality Assessment: From Error Visibility to Structural Similarity. *IEEE Transactions on Image Processing*, 13(4):600–612, April 2014.