Design of Multimedia Applications

Assignment 3: Video coding

Deadline: April 24 2020 23:59

1 Introduction

1.1 Compression of still images

Figure 1 gives an overview of the different steps in the compression process for a simple codec for still images. These phases are fairly common and occur in almost every modern block-based image coding algorithm [1]. The most important phases that are followed during compression are intra prediction, the transformation, quantization, scanning and entropy coding. In a first step, intra-prediction will use the similarity of surrounding pixels to form a prediction for each block. The purpose of the transformation phase is to localize spatial and spectral redundancy in the pixel values. In the quantization phase, the transformed coefficients will be represented with less precision and using a smaller number of bits.

[1] Note that with older compression algorithms, such as JPEG, there is no intra-prediction step yet.

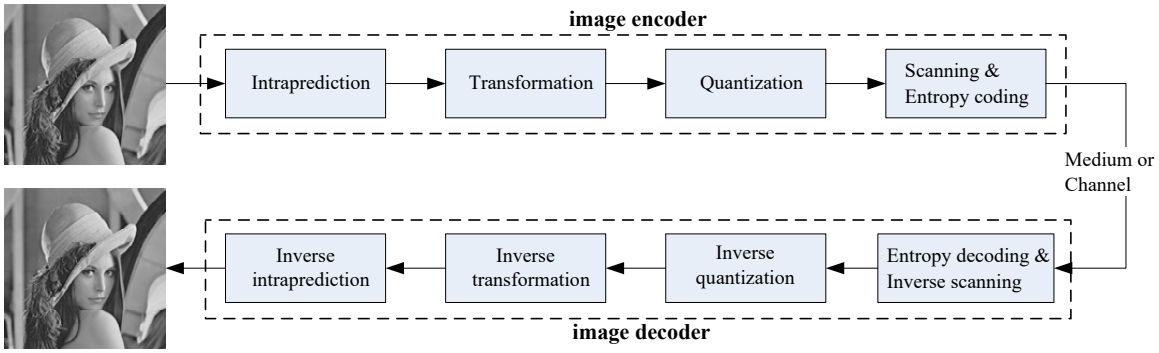


Figure 1: Blockscheme of a still image codec.

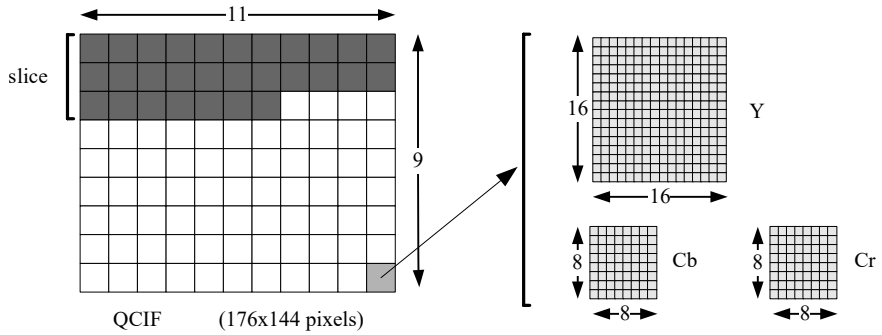


Figure 2: Subdivision of an image in slices and macroblocks.

Note that the transformation step itself does not provide compression; deleting coefficients during quantization does. Since the quantization step is lossy, the quality degrades. The scanning will convert the 2-dimensional representation of the quantized image into a one-dimensional representation. The final phase is the entropy coding phase. This ensures a further general lossless compression. Entropy coding will reduce redundancy between the different quantized values and generate a bit stream. Commonly used entropy coders are the Huffman coders, arithmetic coders and simple run-length coders. The different phases of the coding process are discussed in the following paragraphs.

1.1.1 Subdivision of an image

A video sequence is made up of a series of images consisting of three matrices with pixel information, one for the luminance component and two for the chrominance components. Each image is further subdivided into a series of macro blocks. A macro block consists of a matrix of 16x16 luminance samples and two matrices with chrominance samples. The number of chrominance samples depends on the sub-sample format used. Macro blocks are grouped into slices such that each macro block belongs to just one slice. In Figure 2 this partitioning is illustrated for one QCIF image (176x144 pixels). The image is subdivided into 99 macro blocks. The structure also shows the structure of one of these macroblocks.

1.1.2 Intra prediction

A possible method of exploiting spatial dependencies in an image is not to code each pixel value separately, but to predict it from the neighboring pixels. Each pixel is then predicted based on one and/or multiple previously encoded pixels. Suppose we want to predict the pixel value of pixel X in Figure 3. The simplest form of prediction is to take

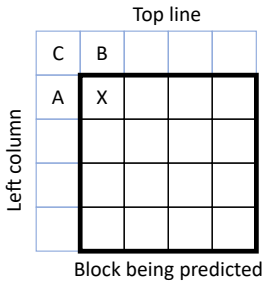


Figure 3: Pixel prediction

the pixel value of the previous coded pixel (pixel A). A more advanced prediction can be obtained by taking a weighted average of neighboring, already coded pixels (e.g. pixels A, B and C). The predicted value of the pixel (X) is then subtracted from the actual value. This difference (the prediction error) is used further in the compression scheme. Due to spatial correlation, this prediction error will be small. As already stated above, efficient compression can be obtained by presenting (small) common values with short code words and (large) rare values with longer code words (Huffman entropy coding). Since the differences obtained are often small, these can be properly compressed during entropy coding. A more advanced form of intra-prediction will be studied in this assignment.

1.1.3 Transformation

One way to remove the spatial correlation from the image is to use the 2-dimensional Discrete Cosine Transform (DCT). The pixel values are transformed from the spatial to the frequency domain. For this the image is first divided into discrete square blocks. Typical block sizes are 8x8 and 16x16. A DCT is performed on each of these blocks. It is important to note that a DCT itself does not cause compression (after performing an inverse DCT, the original coefficients are recovered). It can be said that an image is a linear combination of 2-dimensional DCT basic functions. A DCT transformation ensures that an image is a linear combination of these basic functions. Due to the characteristics of the DCT, the image will be resolved after transformation into a small number of visually important coefficients, while the vast majority of coefficients present visually less important information. In other words, the visually important information of a macroblock is contained in a small number of coefficients. For typical images, these coefficients are located at the top left. The coefficient in the upper-left corner of the transformed macroblock is called the DC coefficient, the other coefficients are called the

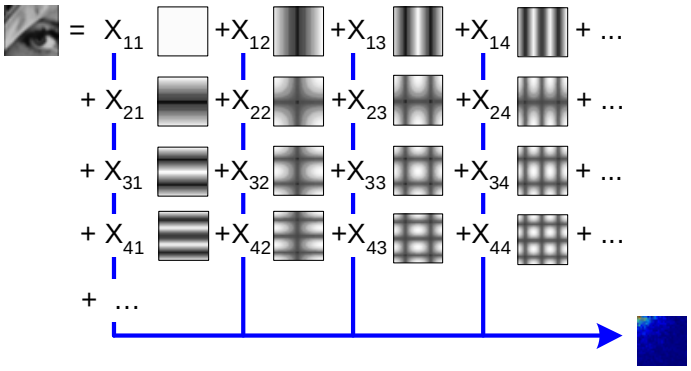


Figure 4: DCT: macroblock is a linear combination of basic DCT functions, X_{ij} are DCT coefficients.

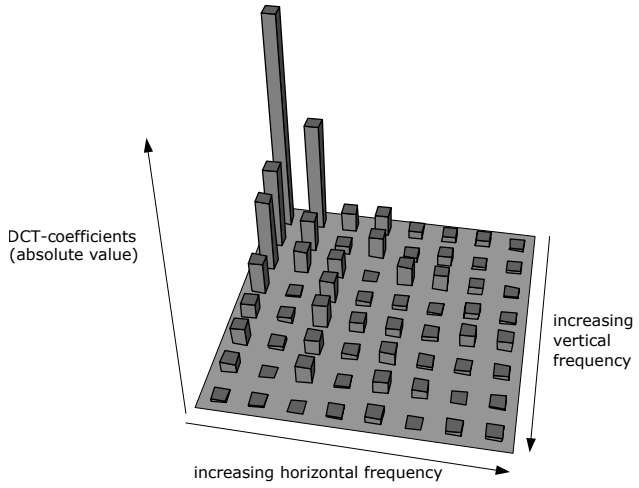


Figure 5: Example of a DCT co-efficient matrix (in 3D) associated with an 8x8 block; the numerical values of the different coefficients are given in the left table of Figure 6. The largest (absolute) values are in the upper left corner. The farther away from this region, the higher the spatial frequencies.

"AC" coefficients. This is illustrated in Figures 4 and 5.

1.1.4 Quantization

As already mentioned, the DCT transformation does not cause compression. The purpose of quantization is to remove the visually less important coefficients or to reduce the accuracy. The quantized coefficients can be encoded with fewer bits due to their lower entropy. With scalar quantization, this is applied separately for each coefficient. With vector quantization, this is applied to a group of coefficients. In Figure 6 this principle is illustrated for scalar quantization where each coefficient is shifted to the right by 2 bits (division by 4). Note that this process is lossy, after inverse quantization, the original

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 6: (left) Original DCT coefficients; (middle) coefficients after quantization; (right) coefficients after inverse quantization and transform

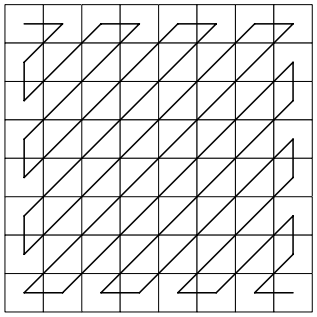


Figure 7: Zigzag scanning approaches to convert the 2-dimensional representation into a 1-dimensional representation.

DCT coefficients are no longer reproduced.

1.1.5 Scanning

This step ensures the conversion of the 2-dimensional representation into a 1-dimensional representation that can be given to the entropy coder. This can easily be done by traversing the matrix obtained after the DCT transformation. The intention is to place the most significant coefficients at the front of the 1-dimensional representation. This allows for more effective coding in the next step.

A DCT transformation usually uses the zigzag scan (see Figure 7) because it first finishes the entire top left corner where the most important coefficients are normally located.

The 1-dimensional representation of the quantized macroblock from Figure 6 using zigzag scan is then:

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,

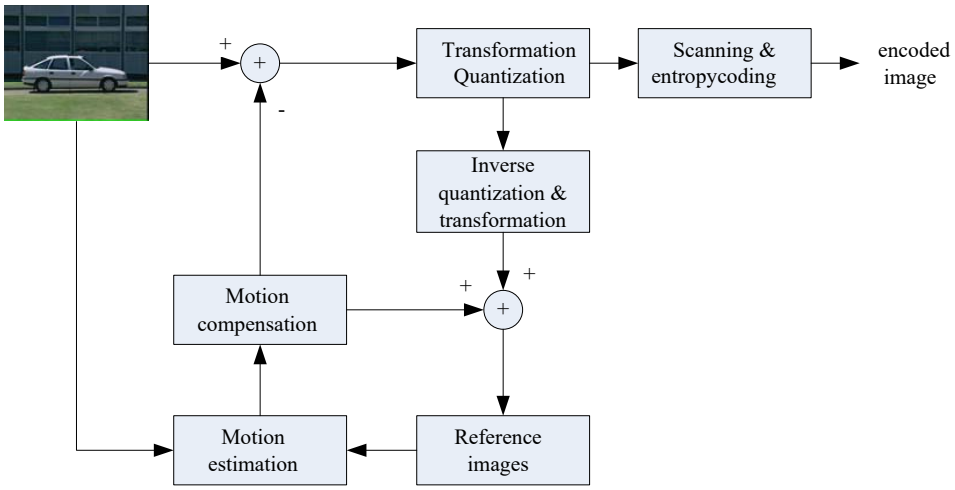


Figure 8: Video coder with motion estimation and motion compensation

Figure 9: Prediction using a reference image

0, 0, 0, -1, 0, -1, 0, 0, 1, 0, 1, 0, -1, -1, 0, -1, 0, 0, 0, 0, 0, 0

1.1.6 Entropy coding

After transformation and quantization, a macroblock will consist of a small number of significant coefficients. These non zero coefficients can be efficiently coded using statistical methods. In this step, the redundancy between the quantized values is detected and exploited to generate a compact bit stream.

1.2 Compression of moving images

1.2.1 Motion estimation and compensation

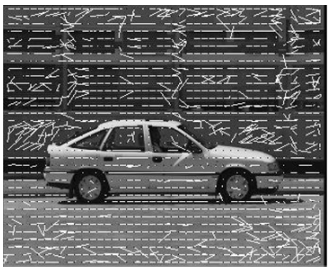
A commonly used way for motion compensation is block-based motion compensation. The current image is first subdivided into rectangular blocks of MxN pixels. These often correspond to macro blocks (16x16 pixels) or smaller partitions. Then, for each of these blocks, there is a check to see which block in the previous image is a good estimate of the current block:

- First the reference images are searched with the intention of finding a good prediction for the current block. To do this, the current block is compared with some or all of the possible MxN blocks in the corresponding search window (this usually corresponds to an area centered around the current position) and the block that best matches the current block is selected. A frequently used criterion here is the energy present in the residue from the difference of the current block and the candidate block in the reference image. The candidate block that results in the lowest energy is then chosen as the best prediction for the current block. This process is known as motion stimulation.

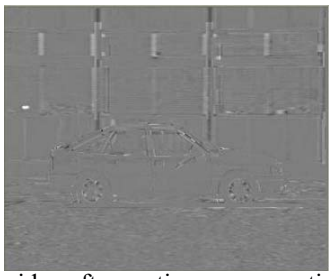
- Thereafter, the selected candidate block is used as a prediction for the current block and is subtracted from the current block to form the residue in this way. This is also referred to as motion compensation.

- Finally, the residue is coded using the techniques discussed in Section 1.1, namely transformation, quantization, scanning and entropy coding. Furthermore, the vector representing the difference in coordinates between the current block and the selected candidate block is also signaled in the video stream. This is also referred to as the corresponding motion vector. In Figure 10, the optimal motion vector for each 16x16 block and residue is depicted. It is clear that the residue contains less energy after motion compensation than without motion compensation (see Figure 9).

The decoder uses the received motion vector to reconstruct the selected candidate block. By decoding the corresponding residue and adding it to the reconstructed block, a reconstructed version of the original block is obtained.



motion vectors



residue after motion compensation

Figure 10: Motion vectors as obtained after motion estimation and the associated residue

2 Assignment

In this assignment we will study four functional blocks of a block-based video codec. This particularly concerns the transformation, quantization, intra-prediction and motion estimation. We will discuss these four blocks in more detail in the following sections.

The encoder that is used in this assignment is based on the MPEG-2 Video standard. Since we have opted for a clear and well-arranged framework, the encoder does not fully meet this specification.

2.1 Building blocks

2.1.1 Transformation

During the transformation we use the 2-dimensional Discrete Cosine Transform (DCT) as described in Section1.1.

The traditional DCT of a matrix with dimension NxN is given by:

$$\mathbf{Y} = \mathbf{A}\mathbf{X}\mathbf{A}^{\mathbf{T}} \tag{1}$$

and the inverse DCT (IDCT) by:

$$\mathbf{X} = \mathbf{A}^{\mathbf{T}}\mathbf{Y}\mathbf{A} \tag{2}$$

where $\mathbf{X}$ is the matrix with the original or residual data with dimensions NxN, $\mathbf{Y}$ is the DCT-transformed from $\mathbf{X}$ with dimensions NxN and

$$A_{ij} = C_j cos\frac{(2i+1)j\pi}{2N} \qquad met\ C_j = \sqrt{\frac{1}{N}}\ (j=0), \qquad C_j = \sqrt{\frac{2}{N}}\ (j>0) \tag{3}$$

In the MPEG-2 Video standard, an image is first split into blocks of 8x8 pixels, after which the 2-dimensional DCT is performed on each of these blocks. We will adopt this method in this assignment. As a result, each macroblock is split into six blocks of 8x8 pixels: four for the luminance component and one for each chrominance component. These six blocks are the input for the 8x8 DCT.

2.1.2 Quantization

After the transformation, the coefficients are quatized. As already mentioned in Section 1.1.4, a distinction can be made between scalar and vector quantization. With scalar quantization, each coefficient will be rescaled separately, while with vector quantization the rescaling will take place on a group of coefficients. The MPEG-2 Video standard uses a scalar quantizer. Furthermore, this quantization is made up of two steps: a forward quantization in the encoder and an inverse quantization in the decoder. Since a coefficient after quantization and inverse quantization will not necessarily get its original value, this step is irreversible and will cause compression.

The standard quantization is defined as follows:

$$Z_{ij} = round\,(Y_{ij}/Qstep) \tag{4}$$

The corresponding inverse quantization is obtained by:

$$Y'_{ij} = Z_{ij} \cdot Qstep \tag{5}$$

with Y_{ij} a coefficient from matrix $\mathbf{Y}$ obtained after teh DCT transform. $Qstep$ is the stepsize of the quantization. Z_{ij} is the quantized coefficient and Y'_{ij} the value after inverse quantization.

The *round* function does not necessarily rounds to the nearest integer.

The larger the step size of the quantizer, the lower the quality obtained. This is because more input values will be mapped to the same value, as a result of which the corresponding quantization error will also increase. On the other hand, a larger step size will compress the video more because the range of the quantized coefficients is smaller and therefore efficient coding can be done. Since fewer bits will be needed, a lower bit rate will be achieved. In case the step size is small, the values obtained after quantization and inverse quantization will be a better approximation than with a large step size. However, this small step size also means that compression will be less efficient.

In Figure 6, the following quantization has been used:

$$Z_{ij} = \lfloor Y_{ij}/Qstep \rfloor \qquad with \quad Qstep = 4 \tag{6}$$

A disadvantage of this quantizer, however, is that all values that belong to the same interval are converted to the lowest value. With this quantizer, for example, all values belonging to [4,8[will be rescaled to 1. After inverse quantization, they will all get the value 4, the value of the smallest element in the interval. Consequently, the average quantization error will be large. A better solution is to reconstruct the values within an interval on the average value of this interval. The following quantizer is an example of this:

$$Z_{ij} = sign(Y_{ij}) \lfloor abs(Y_{ij})/Qstep + 0.5 \rfloor \tag{7}$$

The average error made during quantization for the same step size will be smaller, resulting in a higher quality.

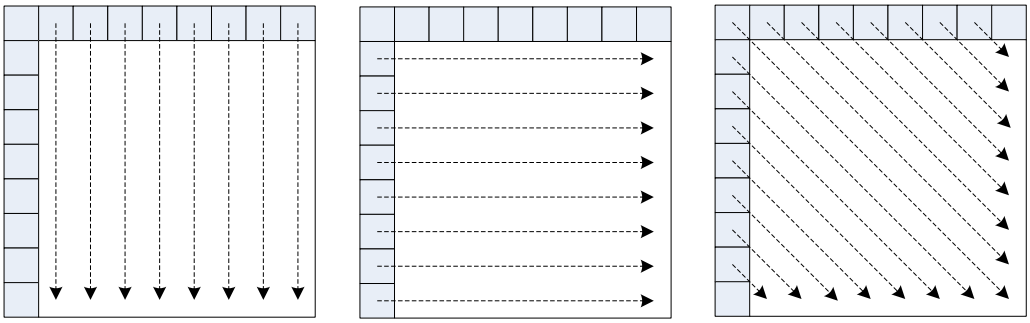
In the MPEG-2 Video standard and more recent video specifications, the encoder goes one step further. After the transformation step (discussed in Section 1.1.4), the important coefficients will mainly be in the upper-left corner of the matrix, while less significant values are more in the lower-right corner. This is taken into account in the quantization step by making the quantization location-dependent. This means that the step size used for more important coefficients will be smaller than for less significant values.

2.1.3 Intraprediction

In the H.264/AVC standard there is the option of performing intraprediction with variable block size. A distinction can be made between intra-prediction at 4x4, 8x8, or 16x16 level. Up to nine prediction modes are available for intra 4x4 and intra 8x8 prediction.

In this assignment we will study four prediction modes, namely DC prediction (mode 0), vertical (mode 1), horizontal (mode 2) and diagonal prediction (mode 3). These four prediction modes will be applied to 16x16 blocks for the luminance component and to 8x8 blocks for the chroma component. With DC prediction, the average value of the pixels on the left and above is calculated (a total of 32 pixels for luma and 16 pixels for chroma). This average is subtracted from the current block; the residue is further processed by the transformation, quantization and entropy coding.

In the vertical, horizontal and diagonal modes, surrounding pixel values are extrapolated in the respective directions to predict the current block. A copy of the edge pixel value is used, so the prediction is the same for all locations on the same row / column / diagonal line. These modes are illustrated in Figure 11 for 8x8 blocks. The prediction for 16x16 blocks is constructed in an similar way.



(a) Vertical intra prediction (mode 1)

(b) Horizontal intra prediction (mode 2)

(c) Diagonal intra prediction (mode 3)

Figure 11: Intra prediction applied to 8x8-blocks

The algorithm for deciding which prediction mode is used is not standardized. When searching for the best prediction mode for intra prediction, different metrics can be used to calculate the "energy" present in the residue after prediction. The formulas below describe four ways to calculate this energy: SSE, MSE, SAE and MAE. The block size here is NxN pixels; C_{ij} and R_{ij} correspond to the pixel values of the current block and the block formed by intra prediction, respectively.

$$1.\, Sum\, of\, Squared\, Errors: \qquad SSE = \sum_{i=0}^{N-1}\sum_{j=0}^{N-1}(C_{ij} - R_{ij})^2 \tag{8}$$

$$2.\, Mean\, Squared\, Error: \qquad MSE = \frac{1}{N^2}\sum_{i=0}^{N-1}\sum_{j=0}^{N-1}(C_{ij} - R_{ij})^2 \tag{9}$$

$$3.\, Sum\, of\, Absolute\, Errors: \qquad SAE = \sum_{i=0}^{N-1}\sum_{j=0}^{N-1}|C_{ij} - R_{ij}| \tag{10}$$

$$4.\, Mean\, Absolute\, Error: \qquad MAE = \frac{1}{N^2}\sum_{i=0}^{N-1}\sum_{j=0}^{N-1}|C_{ij} - R_{ij}| \tag{11}$$

These formulas can be calculated only on the luminance component or on both the luminance and the chrominance component of the blocks to be compared. This choice is

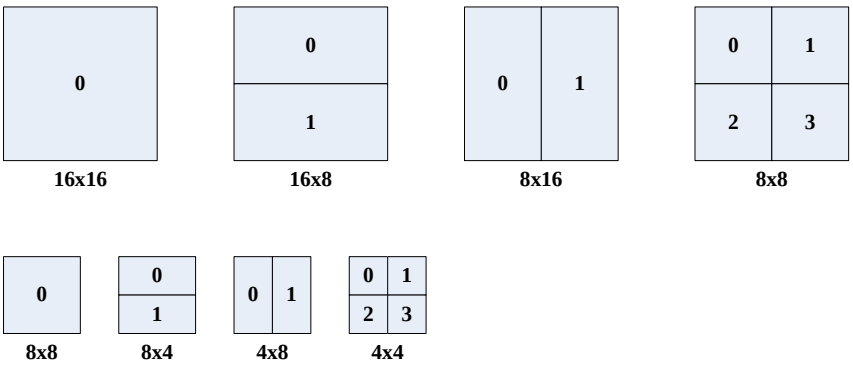


Figure 12: Motion estimation and compensation with block partitions in H.264/AVC

left to the developer. In the majority of existing encoders, only luminance is considered to limit the computational complexity.

2.1.4 Motion estimation and compensation

The size of elementary blocks on which motion estimation is performed depends on the video standard used. In older standards, this step was mainly performed on block sizes of 16x16 pixels, which corresponds to the size of a macroblock. More recent standards will, however, divide these macro blocks into smaller blocks such as, for example, 16x8, 8x16, 8x8, 8x4, 4x8 and 4x4 pixels. The advantage is that an optimal division can be selected depending on the contents of the macroblock. Each of these blocks then has a corresponding motion vector. In H.264/AVC, the partitioning goes as small as 4x4 pixels as shown in Figure 12.

Dezelfde maten als hierboven (SSE, MSE, ...) kunnen gebruikt worden om de energie na predictie te berekenen. In het geval van bewegingsestimatie stellen C_{ij} en R_{ij} in formules (10)-(13) respectievelijk de pixelwaarden van het huidige blok en van het kandidaatblok voor.

The same measures as above (SSE, MSE, ...) can be used to calculate the energy after prediction. In the case of motion estimation, C_{ij} and R_{ij} in formulas (10) - (13) represent the pixel values of the current block and of the candidate block, respectively.

With motion estimation, a distinction is made between the algorithms used to find the best prediction. *Full Search* algorithms will take into account all the blocks in the search window, finding the best solution. A disadvantage, however, is that these algorithms are very computationally intensive. In certain applications, it is therefore advisable to use *Fast Search* algorithms. These will only consider a subset of the possible blocks. A lot of

publications on this subject have been published in the literature.

2.2 Assignment

In this practicum the intention is to implement a simple encoder. For this a framework is made available in which a number of components must be supplemented: DCT, quantization, intra-prediction and motion estimation.

- The first block is the transformation. The formulas given in Section 2.1.1 can be used. Formulas (1), (2) and (3) will be used during implementation. Furthermore, blocks of 8x8 pixels are used. This means that for each macroblock six transformations must be performed: four for the luminance component (the 16x16 macroblock is split into four 8x8 'transformation blocks') and one for each chrominance component.

- The second part comprises the quantization in which use formula (7). Pay sufficient attention to rounding numbers as these are different for positive and negative numbers!

- In the third part, the intra-prediction for the four aforementioned prediction modes must be implemented. Do this at 16x16 level for the luminance component and at 8x8 level for the chrominance component (the same mode is used for both the luminance and the chrominance component) . Depending on the location of the macroblock in the image, the surrounding pixels (pixels above, pixels on the left, pixels on the top left) will be available or not. Use the value '128 'for the unavailable pixels. Calculate the DC prediction only for the pixels above and left (32 pixels for luminance, 16 pixels for chrominance), and also use the value '128' for the non-available pixels. To determine the most suitable mode, the cost must be determined for each prediction mode during the prediction. Use the SSE metric to calculate the energy of the residue; only consider the luminance component.

- In the next part, the motion estimation and compensation will be examined, working on blocks of 16x16 pixels for the luminance component (and on blocks of 8x8 pixels for the chrominance component), which corresponds to a macroblock. The search window must first be defined as shown in Figure 13. Potential candidates will only be searched for in this area in the corresponding reference image.

  For blocks located at the edges of the image, the search window will not completely overlap with the image itself. Special precautions must be taken to initialize the

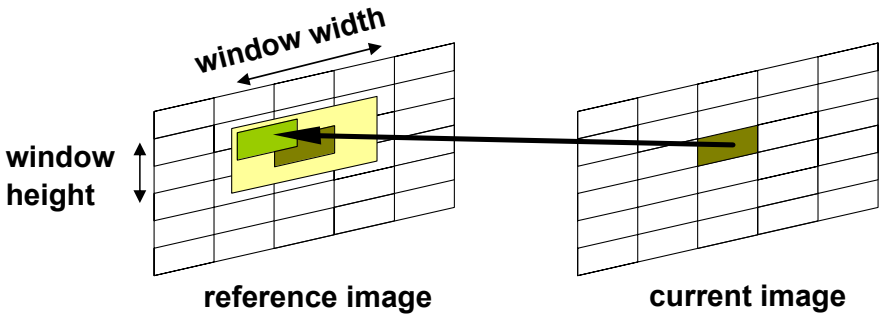


Figure 13: The search window associated with the block to be predicted

pixels in the search window that fall outside the image. Possible solutions are copying the pixels at the edge of the image or initializing these pixels to mid gray (128, 128, 128). Furthermore, it is also possible to prohibit predicting outside the boundaries of the image. Because copying the pixels on the edge of the image gives the best results, we opt for this technique.

Furthermore, the intention is to locate the optimal candidate block in the corresponding reference image on the basis of a Fast Search algorithm. **The algorithm that will be used for this is a fast algorithm that runs in 3 steps (3 step search algorithm, see Figure 14)**. The version of the algorithm that we are going to implement consists of a fixed search window that is searched as indicated in the figure. First, the current position of the block and the eight positions at a distance of 8 pixels around the current position are checked. Thus, an SSE is calculated nine times between the original block and the shifted block in the reference image (SSE 1 MV = (- 8, -8), SSE 2 MV = (- 8,0), SSE 3 MV = (- 8 , 8), SSE 4 MV = (0, -8), ...). The position with the lowest cost (lowest SSE) is used as the starting point for the next step. In the example of Figure 14, the lowest cost appears to be found at MV = (0.0). Around this point (MV = (0.0)) the optimum location is searched for in the same way with a distance of 4 pixels (8 SSE evaluations). In this example, MV = (4.0) now appears to give the lowest SSE. In a third step, a full search is done with a distance of 2 pixels (24 SSE evaluations).

You may use the SSE metric for the evaluation. In the case of motion estimation, only luminance coefficients must be taken into consideration. Once the motion vector is found, the residue must be determined. One motion vector is stored per macroblock. This will be used in the compensation for both the luminance and chrominance coefficients. Do not forget that the latter are under-sampled, which means that the motion vector must be rescaled. Use a division for this resizing.

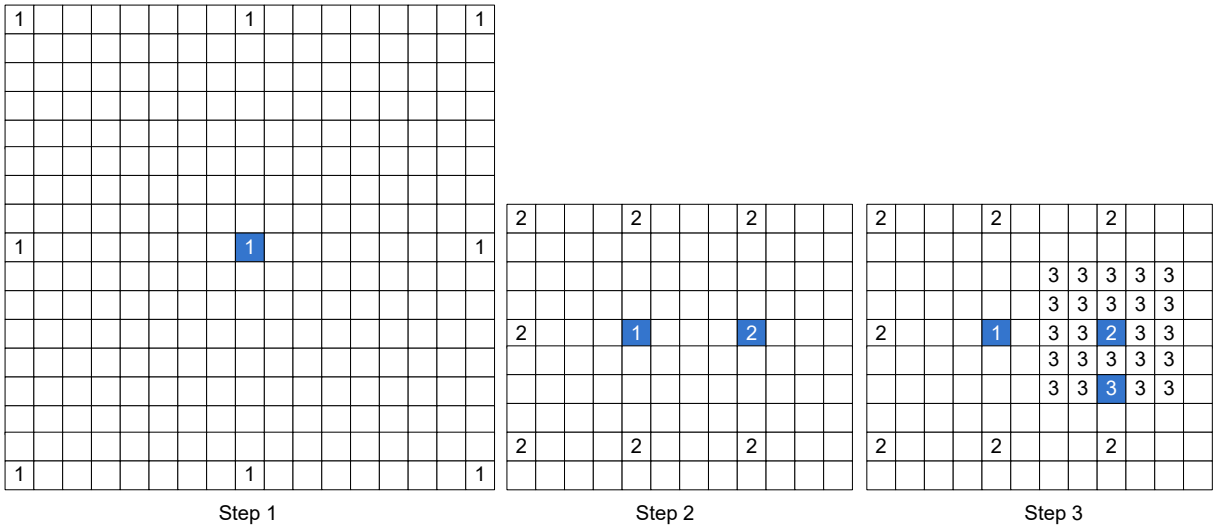


Figure 14: The 3 steps of the fast search algorithm (3 step search).

- In the last part, the motion estimation and compensation is expanded with partitioning. For the partitioning, a macroblock is divided into four 8x8 partitions. The encoder can choose per macroblock to use partitioning or not. If we do not use partitioning, only one motion vector will be encoded. Four motion vectors are transmitted when partitioning.

  It is important to make the right choice regarding the use of partitioning. This determines to a large extent the coding efficiency of the encoder. To make a correct assessment, we compare the distortion D of both possibilities (use the SSE metric for this). Only if the accumulated distortion D across the four 8x8 partitions is smaller than the distortion with only one partition, macroblock partitioning should be used. Note that in an optimized encoder the signaling cost of the motion vectors will also be taken into account.

  Consider a signaling cost of the block as being J. Only if the accumulated cost J over the four 8×8 partitions is smaller than the cost of only one partition, partitioning should be used. The coding cost $J = D + \lambda R$ depends on the distortion D of the residual signal (use the SSE metric for this) and the signaling cost for the motion vectors R (expressed in bits). In principle, the complete entropy coding of the motion vectors must be performed to know R. To keep the implementation simple, however, you may assume the cost per motion vector to be constant. The Lagrange multiplier λ serves to maintain the balance between distortion and signaling costs and should be determined empirically.

2.3 Framework

For this assignment, a framework is made available that is written in C ++. The only blocks that are missing in this framework are the DCT, quantization, intra-prediction and motion estimation and compensation. In the code these can be found in the files *DCT-Transform.cpp*, *Quantiser.cpp*, *IntraPredictor.cpp* and *MotionCompensator.cpp*. We recommend solving the assignment in this order (however, this is not an absolute necessity).

Various coding parameters can be found in the *Config.h* file, including *DO_MC*, *DO_DCT*, *DO_QT* and *DO_INTRA*. These four parameters indicate whether or not to perform these steps in the encoding and decoding process. An encoder will work optimally if all these steps are performed. However, it is also possible to test your code based on these parameters. For example, by only activating *DO_DCT*, only the transformation will be performed. Since these parameters are also passed on to the decoder, it will be aware of which inverse steps must be performed. If only the DCT is implemented within the encoder, the decoder will only be allowed to perform the inverse DCT, and not inverse quantization and motion reconstruction.

Operations will be performed at the macroblock level within each coding step. The coefficients on which the calculations are performed can be found via *mb→luma*, *mb→cb* and *mb→cr*. Since the result after calculation in the next step will be used as input, the results must be placed back in these structures.

The encoder can be called via the command prompt as follows:

```
Encoder.exe <input_file> <input_width> <input_height>
 <qp> <I-interval> <output_file>
```

- `<input_width> <input_height>`

  The given YUV files use the YUV 4:2:0 format and have a resolution of 352x288 pixels.

- `<qp>`

  The quantization parameter (the *qp* value) is used to calculate the *QStep* from formula (9) ($qp \geq 0$). To avoid a division by 0 in the quantization, *QStep* corresponds to $qp + 1$ in this assignment.

- `<I-interval>`

The *I-interval* parameter specifies between how many images an intracoded image must be inserted. This means that this image is not predicted on the basis of previous images and, therefore, motion estimation is not required. This is used to enable random access and / or to prevent propagation of errors. As long as motion estimation is not implemented in your code, it is recommended to set the I interval to 1 so that all images are intracoded.

2.4 Some remarks and hints

- To test the different encoding steps, the parameters in *Config.h* can be used. By decoding the encrypted file, you can see the result. Furthermore, it is also possible to use the inverse methods in the program itself whose syntax you can find in the header files.

- To further simplify testing the code, it is recommended to work in phases. In the case of intra-prediction, instead of looking for the optimal mode, you could take the same mode for all blocks (for example, first DC). As a result, each block is compressed with the DC mode and after decoding it can be checked whether this went well. In the next step you decide in the software that the vertical mode must always be used. Decoding then teaches whether this mode is written without errors, and so on. This can also be done with motion estimation. First set the motion vector to (0,0) without searching and see if the decoder makes a good reconstruction.

- Initialization of surrounding macroblocks in intra-prediction:
  The pixel values of surrounding blocks must be retrieved during the intra-prediction step. Although the mb_up and mb_left pointers (Macroblock class) seem to qualify for this, they will not always produce the desired result. Define your own pointers to the surrounding macro blocks.

2.5 Evaluation

Different aspects of the project will be evaluated independently of each other. The following is a small list that must be taken into account when submitting the project.

First and foremost, the submitted project is compiled as supplied by you. So make sure that all project files are returned unaltered as they were offered in the assignment. The code will be compiled in Visual Studio 2019. Attention will be paid to extra warnings that are generated during compiling. So be careful while programming and avoid warnings.

The correctness of your code will be checked both manually and automatically. Ensure that logic is not repeated in the code by copying and pasting it. Think about which structure you find in the code and try to exploit it. Make the code compact and readable and certainly not cryptic.

For the automatic correcting of the operation of the encoder, the video files offered are encoded with different configurations. Therefore, make sure that the software does not fail during the following configurations. First, a full intra compression is performed on the video images followed by compression using P images. To check whether the quantization has been implemented correctly, 4 different QP values in the range of 2 to 52 will be used. Visual inspection will be performed on these images and most of the problems will already become noticeable. So look at the decoded result of **every** sequence offered with at least 2 extreme QP values.

During the execution of the code, attention will also be paid to the memory usage of the encoder. Not so much the constant amount of memory, but rather the leaking of memory will be judged negatively.