ECE1783H – Design Tradeoffs in Digital Systems
Assignment 1

This assignment (and other succeeding ones) is to help students to grasp the basic fundamentals of digital video compression. The exercises will help you to construct some building blocks and to integrate them forming an abstract video codec system

1. Chroma up-sampling, YUV pixel manipulation, YUV-RGB CSC
    a. Read the YUV 4:2:0 video sequence(s), and upscale it to 4:4:4. Raw YUV files contain pixel data without headers. For example, for a 4:2:0 video sequence of 352x288 resolution, there are 101376 bytes corresponding to the luma plane (one byte per pixel, raster order), followed by 25344 bytes for the first chroma plane (notice that 25344 = 101376/4), followed by another 25344 bytes for the second chroma plane. The second frame then starts at byte 152064, and so on
    b. Convert the 4:4:4 YUV content to corresponding sequence(s) of RGB .png files and visually compare – Notes:
        i. You may use the following formulas for CSC:

$$\begin{bmatrix} Y \\ U \\ V \end{bmatrix} = \begin{bmatrix} 0.299 & 0.587 & 0.114 \\ -0.169 & -0.331 & 0.5 \\ 0.5 & -0.419 & 0.081 \end{bmatrix} \times \begin{bmatrix} R \\ G \\ B \end{bmatrix} + \begin{bmatrix} 16 \\ 128 \\ 128 \end{bmatrix}$$

$$\begin{bmatrix} R \\ G \\ B \end{bmatrix} = \begin{bmatrix} 1.164 & 0 & 1.596 \\ 1.164 & -0.392 & -0.813 \\ 1.164 & 2.017 & 0 \end{bmatrix} \times \begin{bmatrix} Y - 16 \\ U - 128 \\ V - 128 \end{bmatrix}$$

        ii. To convert from 4:2:0 to 4:4:4, you can assume one chroma sample corresponds to four co-located luma samples, with no filtering (duplicate each sample in both the X and Y dimensions)

    c. Repeat (a) and (b) but adding random noise to the Y, U, V, R, G, and B components (one at a time) and observe the impact on subjective quality versus the original YUV

2. Basic block-based operations on video content and quality assessment metrics
    a. Read the Y-component of 4:2:0 video sequences, and dump it into corresponding *Y-only* files of each sequence
    b. Read every Y-only file, and apply to it the following operation(s):
        i. Split each frame into (*i*×*i*) blocks (where (*i*) takes the values 2, 8, and 64)
        ii. Use "padding" if the width and/or height of the frame is not divisible by (*i*). Pad with gray (128). If padding is necessary, padding pixels should be placed at the right and/or bottom of the frame.
    c. Calculate the average* of the sample values within each (*i*×*i*) block
    * Use efficient rounded division by (*i*×*i*) while calculating the approximated average

d. Replace every (*i*×*i*) block with another (*i*×*i*) block of identical elements of this average value to generate Y-only-block-averaged file(s)
e. Subjectively compare* every original Y-only file with its corresponding Y-only-block-averaged one
   * In addition to the frame to frame comparison, you can use simple frame differencing (multiplied by an arbitrary factor to magnify the deltas)
f. Using average PSNR and SSIM among frames, compare every original Y-only file with its corresponding Y-only-block-averaged one
g. Plot graphs that relate objective quality, with (*i*) for the tested sequences

3. Basic Motion Estimation/Compensation:
   a. Repeat parts a and b of exercise 2
   b. Using integer pixel full search, find the best predicted block for every (*i*x*i*) current block; Except for frame boundary blocks that need special handling, use a search range of +/- *r* pixels in the horizontal and vertical dimensions (centered around the collocated block of *reconstructed* previous frame) . Use (*r* = 1, 4, and 8)

      i. While predicting the first frame, assume a hypothetical reconstructed reference frame with all samples having a value of 128

      ii. Use Sum of Absolute Difference (SAD) as the ME assessment metric. In case of a tie, choose the block with the smallest motion vector (L1 norm: |x|+|y|, not Euclidean). If there is still a tie, choose the block with smallest y. If there are more than one, choose the one with the smallest x.

      For boundary blocks, do not consider vectors that would make the block fall partially or totally outside the previous frame.

   c. Dump the x and y components of every successful MV into a text (and/or binary) file using an arbitrary format that preserves the coordinates of the block

   d. An (*i*×*i*) residual block will be generated by subtracting the predicted block from the current block

   e. Generate an approximated residual block by rounding every element in the (*i*×*i*) residual block to the nearest multiple of 2^n (for *n* = 1, 2, 3)

   f. Dump the (*i*x*i*) approximated residual values into a text (and/or binary) file using an arbitrary format that preserves the coordinates of the block

   g. Add every (*i*×*i*) block of approximated residuals to the corresponding (*i*×*i*) predictor block, generating an (*i*×*i*) reconstructed block, which should be used in the prediction process of blocks in the following frames

   h. Repeat for all (*i*x*i*) blocks in the frame (processed in raster order, starting from the top left block of each frame, ending with the bottom right block of the frame)

   i. A Y-only-reconstructed file will be constructed from the reconstructed (*i*×*i*) blocks

j. Compare every original Y-only file with its corresponding Y-only-reconstructed file. Use subjective and objective quality metrics. Highlight cases to showcase the impact of content type, resolution, as well as the values of *i*, *r*, and *n*

k. Decoder

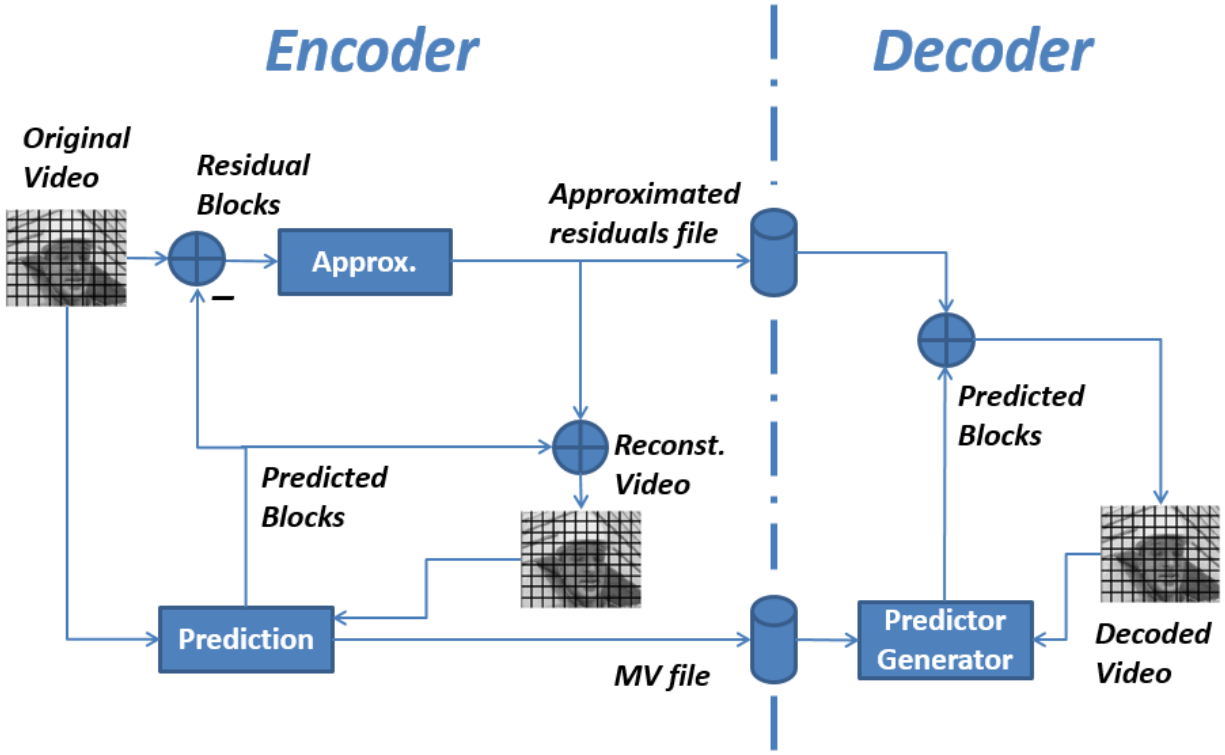
   i. Construct a decoder that uses the approximated residual file as well as the MV file (that were generated in the encoding process) to generate a Y-only-decoded file

   ii. The decoder should add the approximated (*i*×*i*) block to the predictor block that can be identified by the corresponding MV to generate the *Y-only-decoded* files

   iii. All *Y-only-decoded* files should be compared against the *Y-only-reconstructed* files of Part (2). They should fully match

   iv. The same as for the encoder, the blocks should be processed in raster order, starting from the top left block, ending with the bottom right block. Also, assume a hypothetical reference frame of all-128-values while decoding the first frame

   v. The given figure shows a high-level view of the encoding/decoding process as described in this exercise



General Notes

- You can download 4:2:0 YUV sequences from this webpage. Make sure to select sequences with various resolutions and types of motion to showcase the impact of the encoder's settings. For practical execution time with high resolution content, use only a subset of the sequence
- To view/playback the raw YUV content, you can use this or this tool, or any similar one. It is also possible to convert them to formats most video players and editors support, for example using FFmpeg: `ffmpeg -s 352x288 -i CIF_sample.yuv -vcodec copy output.avi`
- A recommended practice is to parametrize your code, and to develop your own script to generate the config files, call the executables, collect results, and put them in presentable formats
- In future assignments, it will be required to accelerate the encoder's throughput through frame as well as block-level parallelism. Make sure to develop your code in a platform that allows parallel processing (e.g. multithreading environment, or MATLAB with parallel computing toolbox)

Implementation Notes

In order to verify that your motion compensation code is working correctly, you should visualize the predicted frame, as well as the absolute value of the residual. Here is an example (results will vary depending on parameters):

Reference (previous) frame

Source frame (to encode)

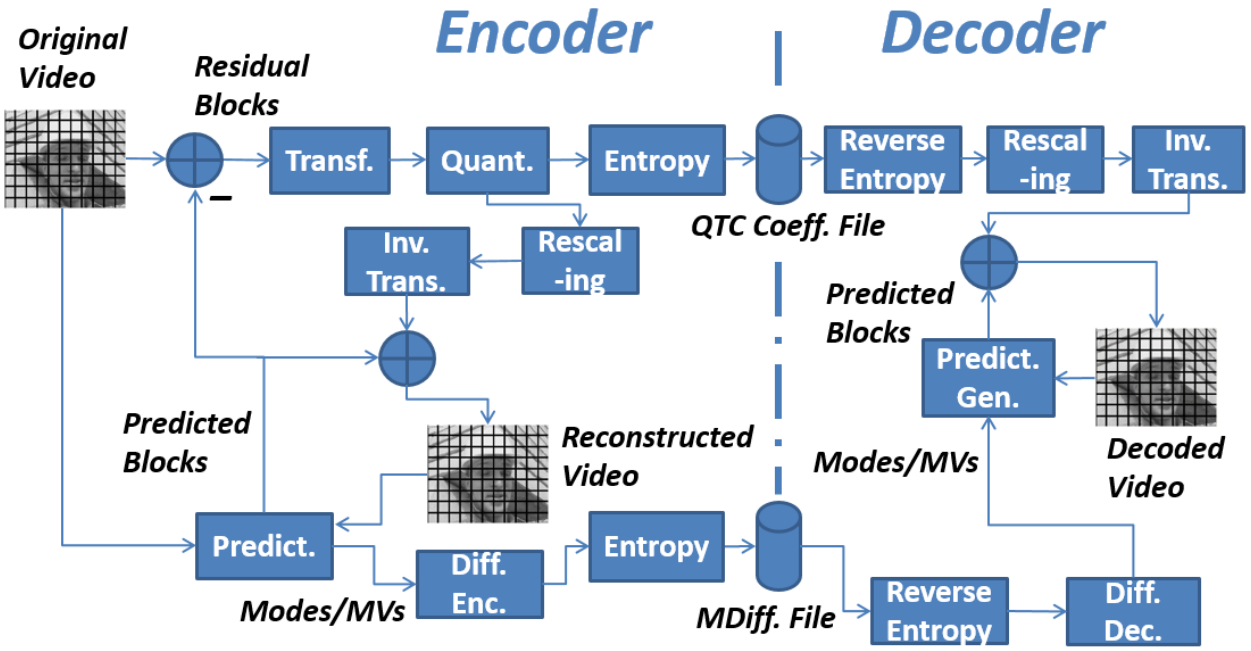
Absolute value of the residual (difference) with no motion compensation

Absolute value of the residual after motion compensation (if it is working correctly, it will be smaller than before motion compensation)

Predicted frame after motion compensation. Note i-sized blocks visible around mouth area.

In order to avoid discontinuities in the residual (which is important especially for the next part), use a signed data type with appropriate range (for example, 16-bit short). You want to avoid integer wraparound.

4. The goal of this exercise is to turn the encoder/decoder system that was developed in Exercise 3 to a more realistic one such as shown in the following figure.



The following is the list of changes:

a. Transform/Quantization/Rescaling/Inverse Transform
   i. Apply (ixi) 2D DCT transform to every (ixi) block. Input to the 2D DCT block is the (ixi) block of signed residuals, while the output is an (ixi) block of signed transformed coefficients (rounded to the nearest integer). You are allowed to integrate ready-made libraries of 2D DCT into your system or to develop your own implementation based on the mathematical formula(s) provided in lecture. You are encouraged (but not requested) to test the *separablility* feature of DCT by replacing the 2D DCT block by two cascaded 1D DCT blocks (and a transpose operation)
   ii. Same applies to the 2D Inverse DCT (IDCT) block at both encoder/decoder, which generates the (ixi) block of signed residuals from the (ixi) block of signed integer transformed coefficients
   iii. Use the following formulas for the quantization operation of every element (x,y) in the (ixi) transform coefficients (TC) matrix:

For every x and y from 0 to (i-1)

QTC[x][y] = round(TC[x][y]/Q[x][y])

Where Q is defined as follows:

If (x+y < i-1) then Q[x][y] = 2^{QP}
Else If (x+y == i-1) then Q[x][y] = $2^{(QP+1)}$
Else then Q[x][y] = $2^{(QP+2)}$

QP is a Quantization Parameter that may take any value between 0 and $\log_2(i)+7$ (inclusive) in order to determine the coarseness level of the quantization operation (with "zero" corresponding to lowest quantization/highest quality).
For example, for (i=2) and (QP=0), Q will be defined as:

Q = [1 2]
[2 4],

While, for (i=4) and (QP=2), Q will be defined as:

Q = [4 4 4 8]
[4 4 8 16]
[4 8 16 16]
[8 16 16 16]

Obviously, QP needs to be communicated to the decoder to obtain Q matrix and use it for rescaling, by multiplying every QTC by its corresponding element in Q.

As a *side test*, you are encouraged to transform/quantize/rescale/inv. transform every (ixi) block of some sample *original* image(s) (not the *residual* ones for this specific side test) and to witness the effect. For example, below is the reconstructed image after applying transform/quantization/rescale/inv. trans. to the first frame of Foreman CIF using the following parameters (i=8 and QP=6).

b. Prediction
    i. One of the adjustable configurations that should exist in the config file of the encoder is the I_Period (number of P frames between every two I frames, plus one of the I-frames). First frame of a sequence is always an I-frame
    ii. Every (ixi) block in an I frame will be encoded using intra prediction, as follows:
        1. Horizontal or (Vertical) intra predictors (at encoder and/or decoder) are obtained by propagating the left-i (or top-i) border *reconstructed* samples (use a value of 128 for absent samples at left-/top-frame borders). Note that you will use previously decoded values from the same/current frame
        2. The mode that gives lowest SAD cost is selected (e.g., "0" for horizontal and "1" for vertical)

c. Differential Encoding/Decoding
    i. Inter (MVs) or intra (modes) need to be differentially encoded at the sender and decoded at the receiver. A simple subtraction with the last block's MV (or intra mode) is sufficient. For first block of every row of (ixi) blocks, assume a hypothetical previous MV of (0,0), or previous mode as (Horizontal). In practice, for the intra predictor, a differential value of 0 will indicate "same as before", whereas differential values of 1 or -1 will indicate a change

d. Entropy Encoding/Decoding
    i. Every frame will have a type marker (1: I-Frame; 0: P-frame, specified even for the first frame). Then every block will need the differential prediction information (one value for Intra, two for Inter, as described in section c). To encode prediction info, use Exponential-Golomb Coding with extension to negative numbers as described here. The (ixi) quant. trans. coefficients (the result of applying the 2D DCT to the residual, and the quantization operation to the resulting coefficients) will go through scanning, RLE, and then coding stages as described below.

Assuming the quant. trans. coefficients are in matrix form and (0,0) is the lowest frequency one (DC), reorder them in diagonal order of increasing frequency, like so: (0,0); (0,1); (1,0); (0,2); (1,1); (2,0); For example, if they look like this:

-31	9	8	4
-4	1	4	0
-3	2	4	0
4	0	-4	0

Reorder them like so: -31, 9, -4, 8, 1, -3, 4, 4, 2, 4, 0, 4, 0, 0, -4, 0. After this, encode them using a form of RLE where you prefix a number that indicates whether a run of non-zero or zero numbers are coming: if non-zero terms are coming, encode –(number of non-zero terms) followed by the terms. If zero terms are coming, encode +(number of zero terms). If there are no more non-zero terms, encode a 0 to indicate that the rest of the elements in the array are

all zeros. Continuing the example above, the reordered matrix will be encoded as follows:
-10, -31, 9, -4, 8, 1, -3, 4, 4, 2, 4, 1, -1, 4, 2, -1, -4, 0.

Like for the frame type and differential prediction information, use Exponential-Golomb Coding with extension to negative numbers as described here to code the runs/prefixes and the coefficients.

Side Note: Sometimes, it may be cheaper not to encode all zeros as runs, since a run will need at least 3 bits, whereas a single 0 needs only 1 bit. Accordingly, you can optionally optimize your encoder not to use zero runs when it would cost more. For example, the array above would be represented as: -15, -31, 9, -4, 8, 1, -3, 4, 4, 2, 4, **0**, 4, **0**, **0**, -4, 0. Notice that the encoder deliberately decided to treat the **underlined 0's** the same as non-zero coefficients, because it is more efficient in terms of bitcount.

For example, an 8x8 inter block with the same MV as the previous block and all-zero quantized transform coefficients should use 3 bits in total (three zeros: two for the MV difference, and one to mark that all 64 elements of the quant. trans. coefficients array are zeros). The prediction info and the quantized transform coefficients maybe dumped into a single file as described here (which is more realistic), or for simplicity, they may be dumped into two separate files as described in the first diagram. The files maybe binary ones (which is more realistic), or they may simply be text files to ease the debugging. During development, you may dump intermediate data into binary and/or text files and visualize them for easy debugging.

Obviously, this reordering, RLE, and coding operations should be fully reversed (without any loss) at the decoder side.

General Comments on Exercise 4

- Use PSNR as your metric to assess *quality*
- Use the sum of binary digits (bitcount) in QTC Coeff. and MDiff. files as your metric of *bitrate*
- For easier development and testing, use only the luminance (Y) component of the sequence
- Insert necessary reconstruction parameters (e.g., block size and QP) in the bitstream before the first frame for ease of use (header information/sequence parameter set)
- If Exponential-Golomb Coding significantly slows down the *batch* testing, you may calculate bitcount (the number of bits required to encode a value $v \neq 0$ is $3+2\times\text{floor}[\log2(|v|)]$, and $v=0$ needs 1 bit) while dumping into the file(s) the non-entropy coded values instead of the entropy-coded stream. Showing that Exponential-Golomb works will still be credited
- At both the encoder and decoder, make sure that the pixel values of reconstructed video are always clipped to be in the range (0-255)

Deliverables of Exercise 3

In your written report, at a minimum, you will want to include at least:

- A per-frame SAD graph for varying i with fixed r=4 and n=3, another for varying r with fixed i=8 and n=3, and another for varying n with fixed i=8 and r=4. You can use any sequences you want, but the first 10 frames of Foreman CIF (352x288) are a requirement, plus at least one other sequence of different dimensions (number of frames at your discretion).
- For the i=8, r=4 and n=3 case, include a Y-only reconstructed file of the first 10 frames of Foreman CIF (this file should be exactly 1,013,760 bytes in size). Include also a text file with the found motion vectors (the format of this file is arbitrary, but it should contain 15,840 x/y pairs).
- For at least two i cases of your choice, the residual before and after motion compensation, as well as the predicted frame before reconstruction, similar to what is shown in the **Implementation Notes** section. You can choose r, n, and the frame number as you see fit (but do not choose the first frame – the predicted frame must not be all gray)

Answer the following questions in your report:

- What is the effect of the i and r parameters on residual magnitude and encoding time? Provide a table that shows how encoding time varies.
- What is the impact of each of the parameters (i, r, and n) on the final quality of the image (measured as average SAD between original and reconstructed frames)? Explain the results you are seeing.

Deliverables of Exercise 4

In your written report, include the following:

- For a fixed set of parameters (i = 8 and 16; and search range = 2), create R-D plots where the x axis is the total size in bits of a test sequence, and the y axis is the quality/distortion measured as PSNR. Draw 3 curves: one for an I_Period of 1 (GOP is IIII…), another for an I_Period of 4 (GOP is IPPPIPPP…), and finally another for an I_Period of 10 (IPPPPPPPPPIPP…). You get the different points of a curve by varying the QP parameter (0.. $\log_2(i)+7$ – if this is too much, you can skip several and do 0, 3, 6 and 9 for the 8x8 block size, and 1, 4, 7 and 10 for 16x16, but always include these four). Use the first 10 frames of Foreman CIF for testing. You can add more sequences of your choosing, but present them separately. Since you will be running 12×2 (or more) instances of the encoder, it is highly recommended that you write script(s) to run the experiments, collect the results, and put them in a tabular format that easily maps into the desired curves. Record (and plot) the execution times too.
- For (i=8, QP=3) and (i=16, QP=4) experiments, plot the bit-count (on-y-axis) vs. frame index (on x-axis) curves. Show plots for three different values of I_Period (1, 4 and 10).

Encoder R-D performance can be valued during the assignment assessment.

Answer the following questions in your report:

- Which kind of frames (Intra or Inter) typically consume more bitrate? Why? Is this true across the entire QP range?
- For (i=8, search range = 2, and QP=3), what is the compression ratio of the 10 frames of Foreman CIF compared to the uncompressed Y component? What is the average PSNR?

Instructions and Deliverables (For exercises #3 & #4)

- Note: you only have to present exercises 3 & 4, but you will have to use parts of exercises 1 and 2 in order to complete exercise 3 & 4, so it is recommended you do all 4 exercises in order.
- This assignment is to be done in groups of 3 people. You are encouraged to use the Discussion Board in the Portal to search for teammates.
- You can use C, C++, C#, Java, or MATLAB. If you want to use another language, please ask beforehand. As mentioned in the General Notes, the language of your choice should support parallelism, so you should keep in mind if you choose a language such as Python, where some implementations might not expose shared-memory parallel processing in a straightforward manner.
- The deliverables are the source code you developed, two generated files described above, and a report, introducing the problem, explaining how it was approached, showing evidence that the system is working, mentioning the observations/challenges, highlighting results to showcase the impact of different encoder's settings, and proposing ideas/thoughts.
- The due date will be **October 23** before class. By the deadline, the deliverables should be submitted in Quercus (only once per group).
- The clarity and conciseness of your report will be assessed. You should try to make a convincing case that your encoder is working as it should, explaining and reasoning about the results.
- If you have any questions about the assignment you can make use of the teaching assistant's email and/or office hours (Wednesdays 4PM-6PM, Engineering Annex room PT306). For the latter, it is also recommended that you email first to make an appointment.