Practicum 2: Image Compression

This assignment is about implementing an image compression method, which works in a similar way as the one JPEG uses. The width and height of an input image are both multiples of four (e.g. a 1024x512 image). For this exercise we only consider gray-scale images, with 8 bits per pixel, stored in a raw binary format. Your compression routine will need to perform the following steps:

- Subdivide the input image into blocks consisting of 4x4 pixels.
- Transform each block using the Discrete Cosine Transform (DCT), about which you can find extra information at the end of the document. The result of this transformation should be output in a clear way in a log file when specified (more about a log file later).
- Quantize these transformed blocks, using a quantization matrix which will be specified through a configuration file. This will be a 4x4 matrix which specifies the quantization factor for each value of the DCT transformed input block. Note that the quantization factor may differ for each of the 4x4 values. Again, if a log file is specified, the result of this step should be shown in it in a clear way.
- Perform a zig-zag scan (similar to the way JPEG does this) to store these quantized coefficients in a list.
- If specified in the configuration file, you should be able to apply a run-length encoding (RLE) step on this list. You may limit this step to the zero coefficients at the end of the list.
- Write the result of all this, the compressed image, in binary form to an output file. Below, you can find more information about the format to use.
- This output file must contain all information needed to recreate an uncompressed image. For example the quantization matrix and an indication if RLE is used should be stored in the file as well.

You will again need to use the Bitstream related classes you already encountered in the previous assignment. Do not use more bits than necessary when writing your compressed file. For example, indicating if RLE is used in the compressed image can be done using only one bit. Or if certain numbers always lie in the range [0,63], use no more than six bits.

Apart from writing a program that creates a compressed file using the steps above, you must also write a program that performs the necessary steps to decompress such a file. The output of this program must again be a raw grayscale image using 8 bits per pixel. How the quality of this decompressed image relates to the quality of the original input image will depend on the quantization matrix used.

The final result of your programming efforts should consist of two executables: an encoder and a decoder. Both should be command line programs which take exactly one argument: the name of a configuration file. No other input should be required.

The original input file always has the extension **.raw**, while the compressed file always uses **.enc**. The decompressed file is a raw image again and uses the

extension **.raw** again. The name of the configuration file is specified on the command line, and will be something like ‘settings.conf’. This file needs to specify the parameters listed in the example below, obviously with possibly different values and not necessarily in this order. The same file will be used for the decoder but only the name of the encoded file and the decoded output file may be used in that case. All other parameters need to be stored in the encoded file itself. An example ‘settings.conf’ file:

```
rawfile=input.raw
encfile=encoded.enc
decfile=decoded.raw
width=512
height=512
rle=1
quantfile=matrix.txt
logfile=log.txt
```

The ‘quantfile’ option specifies the text file containing the 4x4 quantization matrix, in this example called ‘matrix.txt’. It should contain an integer 4x4 matrix, as in the following example:

```
2 4 8 16
4 4 8 16
8 8 32 64
16 32 64 128
```

The ‘logfile’ specified in this configuration file is the log file mentioned earlier. If specified, information about the transformations involved should be written to it. To turn off the use of a logfile, specify the line
`logfile=`
in the configuration file.

Make sure that the following conditions are met:

- The config file should specify NO other parameters than the ones above.
- On the command line, NO other arguments may be specified.
- Your program may NOT ask for any user input during its execution.
- Output on the screen is allowed but should not be excessive: do not write out the information about every block on the screen.
- Names like ‘settings.conf’ and ‘matrix.txt’ are of course only examples, the real name of the settings file is specified on the command line while the name of the quantization matrix is specified in the configuration file.
- Both the settings file and the file containing the quantization matrix are text files and can use both UNIX (‘\n’) and DOS (‘\r\n’) line separation characters. Your program MUST be able to handle both.

The ‘ImageMagick’ set of programs contains an executable called ‘convert’ that you can easily use to transform any input image to an 8 bit raw grayscale image. The reverse operation is also possible: read a raw 8 bit grayscale image and produce a PNG image, which you can easily open in a standard image viewer. You MUST use PNG for this since this does not introduce any artefacts of its own (as JPG does).

To convert an input photo to a 512x512 raw grayscale image, you can use the following command:

```
convert foto.jpg -resize 512x512! gray:foto.raw
```

To convert a raw 512x512 grayscale image of 8 bits per pixel to a PNG:

```
convert -size 512x512 -depth 8 gray:foto.raw fotogray.png
```

Make sure you test your code using various input images (real photos, cartoons, artificial patterns, …) and think about the validity of the result. Also make sure to test using both large and small quantization values.

Some final remarks:

- It is NOT allowed to use extra quantization factors, only the ones in the matrix file should be used. Make sure you do not introduce any extra multiplications or divisions in your code.
- The encoder will be started as follows (Windows/Linux):
  - **encoder.exe settings.conf** or **encoder settings.conf**
- The decoder will be started as follows (Windows/Linux):
  - **decoder.exe settings.conf** or **decoder settings.conf**
- You will need to reuse your code in the next assignment about video compression, where many more frames will need to be processed. So mind the memory usage and beware of memory leaks!
- These executable names must be used, to allow automating evaluation tests.
- Writing an entire encoder first and then creating an entire decoder is not a good strategy to use. It is much better to work in an incremental way, for example:
  - start by creating an encoder which simply reads the input image, subdivides it in blocks and writes these to an output file. Also create a decoder which can read the blocks from this file and can reconstruct a raw image from them
  - then add forward and inverse DCT operations
  - next, add quantization and inverse quantization
  - …
- Do not focus on the file size too early in the project, make sure everything works first. A very small file that cannot be decoded correctly has no value whatsoever.

You should submit a zip-file using Blackboard, containing:

- The code of the encoder and decoder programs, including a Makefile or project file or anything else needed to build the executables.
- Compiled executables (including dependencies), built in Release mode.
- A Readme.txt file in which you describe which features were implemented and which you could not implement. Mention any artefacts that you noticed during testing but for which you could not find the reason!

After the deadline (which is mentioned on Blackboard), a demonstration of your code will be needed as well. Typically you will need to run your code on

input files and using settings provided by us. More information about this will follow later.

Do not underestimate this assignment: you'll find that errors are typically not straightforward to track down and that debugging takes up a lot of time.

Some additional information about a DCT (from 'H.264 and MPEG-4 Video Compression: Video Coding for Next Generation Multimedia' by Iain E. G. Richardson)

3.4.2.2 DCT

The Discrete Cosine Transform (DCT) operates on **X**, a block of $N \times N$ samples (typically image samples or residual values after prediction) and creates **Y**, an $N \times N$ block of coefficients. The action of the DCT (and its inverse, the IDCT) can be described in terms of a transform matrix **A**. The forward DCT (FDCT) of an $N \times N$ sample block is given by:

$$\mathbf{Y} = \mathbf{A}\mathbf{X}\mathbf{A}^{\mathrm{T}} \tag{3.1}$$

and the inverse DCT (IDCT) by:

$$\mathbf{X} = \mathbf{A}^{\mathrm{T}}\mathbf{Y}\mathbf{A} \tag{3.2}$$

where **X** is a matrix of samples, **Y** is a matrix of coefficients and **A** is an $N \times N$ transform matrix. The elements of **A** are:

$$A_{ij} = C_i \cos\frac{(2j+1)i\pi}{2N} \qquad \text{where } C_i = \sqrt{\frac{1}{N}}\ (i=0), \qquad C_i = \sqrt{\frac{2}{N}}\ (i>0) \tag{3.3}$$

Equation 3.1 and equation 3.2 may be written in summation form:

$$Y_{xy} = C_x C_y \sum_{i=0}^{N-1}\sum_{j=0}^{N-1} X_{ij} \cos\frac{(2j+1)y\pi}{2N} \cos\frac{(2i+1)x\pi}{2N} \tag{3.4}$$

$$X_{ij} = \sum_{x=0}^{N-1}\sum_{y=0}^{N-1} C_x C_y Y_{xy} \cos\frac{(2j+1)y\pi}{2N} \cos\frac{(2i+1)x\pi}{2N} \tag{3.5}$$

For a 4x4 DCT, this results in:

Example: $N = 4$

The transform matrix **A** for a 4×4 DCT is:

$$\mathbf{A} = \begin{bmatrix} \frac{1}{2}\cos(0) & \frac{1}{2}\cos(0) & \frac{1}{2}\cos(0) & \frac{1}{2}\cos(0) \\ \sqrt{\frac{1}{2}}\cos\left(\frac{\pi}{8}\right) & \sqrt{\frac{1}{2}}\cos\left(\frac{3\pi}{8}\right) & \sqrt{\frac{1}{2}}\cos\left(\frac{5\pi}{8}\right) & \sqrt{\frac{1}{2}}\cos\left(\frac{7\pi}{8}\right) \\ \sqrt{\frac{1}{2}}\cos\left(\frac{2\pi}{8}\right) & \sqrt{\frac{1}{2}}\cos\left(\frac{6\pi}{8}\right) & \sqrt{\frac{1}{2}}\cos\left(\frac{10\pi}{8}\right) & \sqrt{\frac{1}{2}}\cos\left(\frac{14\pi}{8}\right) \\ \sqrt{\frac{1}{2}}\cos\left(\frac{3\pi}{8}\right) & \sqrt{\frac{1}{2}}\cos\left(\frac{9\pi}{8}\right) & \sqrt{\frac{1}{2}}\cos\left(\frac{15\pi}{8}\right) & \sqrt{\frac{1}{2}}\cos\left(\frac{21\pi}{8}\right) \end{bmatrix} \tag{3.6}$$

The cosine function is symmetrical and repeats after 2π radians and hence $\mathbf{A}$ can be simplified to:

$$\mathbf{A} = \begin{bmatrix} \frac{1}{2} & \frac{1}{2} & \frac{1}{2} & \frac{1}{2} \\ \sqrt{\frac{1}{2}}\cos\left(\frac{\pi}{8}\right) & \sqrt{\frac{1}{2}}\cos\left(\frac{3\pi}{8}\right) & -\sqrt{\frac{1}{2}}\cos\left(\frac{3\pi}{8}\right) & -\sqrt{\frac{1}{2}}\cos\left(\frac{\pi}{8}\right) \\ \frac{1}{2} & -\frac{1}{2} & -\frac{1}{2} & \frac{1}{2} \\ \sqrt{\frac{1}{2}}\cos\left(\frac{3\pi}{8}\right) & -\sqrt{\frac{1}{2}}\cos\left(\frac{\pi}{8}\right) & \sqrt{\frac{1}{2}}\cos\left(\frac{\pi}{8}\right) & -\sqrt{\frac{1}{2}}\cos\left(\frac{3\pi}{8}\right) \end{bmatrix} \tag{3.7}$$

or

$$\mathbf{A} = \begin{bmatrix} a & a & a & a \\ b & c & -c & -b \\ a & -a & -a & a \\ c & -b & b & -c \end{bmatrix} \quad \text{where} \quad \begin{aligned} a &= \frac{1}{2} \\ b &= \sqrt{\frac{1}{2}}\cos\left(\frac{\pi}{8}\right) \\ c &= \sqrt{\frac{1}{2}}\cos\left(\frac{3\pi}{8}\right) \end{aligned} \tag{3.8}$$

Evaluating the cosines gives:

$$\mathbf{A} = \begin{bmatrix} 0.5 & 0.5 & 0.5 & 0.5 \\ 0.653 & 0.271 & -0.271 & -0.653 \\ 0.5 & -0.5 & -0.5 & 0.5 \\ 0.271 & -0.653 & 0.653 & -0.271 \end{bmatrix}$$