Institute of Parallel and Distributed Systems
University of Stuttgart
Universitätsstraße 38
D–70569 Stuttgart

Study Thesis Nr. 2360

A Just-Noticeable-Distortion Based Perceptually Lossless Image Compression Codec

Kai Liu

Course of Study:	Computer Science
Examiner:	Prof. Dr. Sven Simon
Supervisor:	M. Sc. Zhe Wang
Commenced:	December 1, 2011
Completed:	June 1, 2012
CR-Classification:	E.4, I.4.2

Contents

List of Figures

List of Tables

List of Algorithms

1 Introduction

Motivation

Digital pictures are present in every modern application. They do not only have a cosmetic function but often have practical uses like visualizing important information or clarifying facts.

Uncompressed images, especially in large dimensions, use up a lot of disc space and it is not convenient to transfer those images, especially in situations where only a limited bandwidth is available. Also, loading big files is too memory demanding and time consuming, depending on the device and transfer speed. This is unacceptable whenever time is a critical factor. Over the last decades, many different methods were developed to reduce the file size as much as possible while the original picture is still recognizable.

So-called lossy or near-lossless compression algorithms are used which remove and/or quantize regions of the image to get a smaller file size. This is sufficient for most applications where pictures and images are present to serve as a design choice. But there are also more demanding fields like medicine in which a reduction of picture information is not acceptable as they may contain vital information. Here, lossless compression is of particular interest as it allows to reduce the file size while a perfect reconstruction of the original image is possible.

Because of the higher demand in picture recontructability, lossless compressed images have worse compression ratios than near-lossless compressed images. Thus many attempts and a lot of research was and is being conducted to get the best possible quality while minimizing the file size. This study thesis implements an extension for the JPEG-LS compression codec so a Just-Noticeable-Distortion (JND) measurement can be used to dynamically adjust the quantization step size to the human perception.

Structure

Chapter 2 - Perceptional Coding: Introduces perceptional coding, its principles and explains the JND idea.

Chapter 3 - JPEG-LS: Describes the baseline JPEG-LS algorithm whereas only the regular mode will be treated in more detail.

Chapter 4 - Just-Noticeable-Distortion Calculation: Explains how the Just-Noticeable-Distortion is being calculated and what problems occurred.

Chapter 5 - Integration of the JND-Calculation with near-lossless JPEG-LS: Describes how the JND-Measurement is integrated into near-lossless JPEG-LS and how the codec has to be adapted.

Chapter 6 - Evaluation and Results: Defines visual quality measurement methods and presents the results

Chapter 7 - Summary: Shows an overview of this work and gives an outlook for further approaches.

Scope of work

In this study thesis a JND (Just-Noticeable-Distortion)-Measurement will be implemented on top of JPEG-LS while only considering a grayscale bit depth of 8 Bit. This is sufficient to show a proof of concept of combining the JND approach with JPEG-LS. JPEG-LS is a widely used and relatively simple coding mechanism for lossless and near-lossless image compression.

The JND measurement will be defined, implemented and integrated into JPEG-LS. Therefore a modified approach of [CL95] will be used. The quantization step size (QSS) will be adapted dynamically according to the JND value so that the compression ratio compared to a standard JPEG-LS implementation can be improved. By that, a near-lossless variable bit-rate (VBR) is introduced into the coding flow.

The encoder and decoder are implemented in MATLAB and after defining the visual quality criteria the results are evaluated and analysed in matter of compression quality and performance. Two test picture sets will be used and the perceptual quality of the codec will be evaluated by an Mean-Opinion-Score (MOS) and Multi-Scale Structural Similarity (MS-SSIM) test.

2 Perceptional Coding

Perceptional coding aims to reduce the compressed file size by analyzing certain parts of the image and taking advantage of the human perception. This chapter will introduce some terms and definitions and also explain the general principle of perceptional coding. Finally, the JND idea is briefly explained.

2.1 Disambiguation

The word *codec* is often used in conjunction with multimedia compression. A codec is a portmanteau of the words **co**der and **dec**oder. The term coder itself is an abbreviation to "compressor" and the term decoder respectively to "decompressor".

In this work only grayscale pictures with a bit depth of 8 Bit will be used. This means that only gray shades in the source image will be accepted and processed while there can be exactly $2^8 = 256$ different gray nuances.

The *compression ratio* roughly describes by a number how well a method compresses the source image. For that, the size of the compressed file and the size of the uncompressed file are compared. It serves as a quick reference in how well a codec performs.

In general, one can roughly categorize compression algorithms into lossless and lossy algorithms. *Lossless algorithms* compress a source image in a way so it can be perfectly reconstructed. Only redundant, unneeded information is removed and thus the best possible picture quality is assured and all picture information can be reobtained. *Lossy or near-lossless algorithms* on the other hand accept and allow a certain loss in image quality and image information to be able to compress further than lossless methods. Because of these different quality expectations and requirements, lossy algorithms can achieve a better compression ratio [Pla96]. Actually in lossless image coding the signal to noise ratio relative to the source input should not change in any case. But as long as an average viewer cannot perceive under normal viewing conditions any difference between the source picture and the compressed one, the compression system is said to be *visually lossless* [Gle93].

The *human visual system (HVS)* is an expression to describe not only the receptors, the eyes, but also the psychological processing of images in the human brain. The HVS is very special as it is highly non-linear which is why it is difficult to find good measurements to evaluate the quality of a compressed image by objective algorithms or mathematical formulas [Pla96].

Perceptional coding tries to minimize the perceivable errors. It actually is not limited to still images as its principles, which will be explained in the next section 2.2, may be used in audio and video processing too. Two of the most prominent perceptional coding examples at the moment are the Moving Picture Experts Group MP3 codec [ISO98] and also the H.264 codec [IT11] in video compression which is used in BluRay Discs [Blu10] among others. Perceptional coding can also be used in 3D modelling [CSBY06] or 3D stereoscopic images [LWZ+11].

2.2 Principle

The addition of perceptional coding by considering the human perception in regular coding mechanisms is also called *second generation image coding*. *First generation image coding* algorithms are for example Huffman encoding [RMB94] or the Pulse-Code Modulation (PCM) [KIK85].

In still image processing one way to reduce the file size while maintaining a lossless image is to remove all redundant information. Redundant data is a characteristic which is related to numerous factors such as predictability, randomness and smoothness of image data [Pla96]. Another factor for perceptual redundancies in a picture is the inconsistency in the sensitivity of the human visual system to varying levels of contrast and luminance changes in the spatial domain. Finding the perceptual threshold where an average viewer in normal light circumstances cannot perceive the difference between the original and the compressed image is one of the challenges which have to be overcome.

Because the visually lossless perception is difficult to measure objectively, different approaches exist on how to determine the thresholds. This in turn results in many different ideas because the Just-Noticeable-Distortion can be calculated in many ways depending which factors are considered. Understanding the human visual system is important and thus is often the focus of studies. For example, when looking at a picture, it is divided into different regions by the brain, coarsely and most simply by categorizing a background and foreground. There exist attempts to take advantage of that by having a larger *quantization step size (QSS)* for the background and a smaller one for the foreground and other details. The challenge therein is to find the right measurements and methods to differentiate and categorize the corresponding sections of an image so that the objective is met to finally encode the image visually lossless.

The whole idea of perceptional coding is to effectively adapt the coding scheme to the human perception in a way that allows further removal of perceptually redundant information. This should result in a high quality picture while minimizing the bit-rate. To reduce the bit-rate even more, minimally perceptible image elements can be removed. This usually results in slightly visible visual distortions and is called *Minimally-Noticeable-Distortion (MND)* [JJS93].

Depending on the source material different approaches are more promising to deliver a better compression ratio. There are region-, edge-, mesh- and 3D model-oriented source

models which have their own elements they use as a foundation to compress the data [Pla96]. As this is not the focus here, this work will not go into more detail.

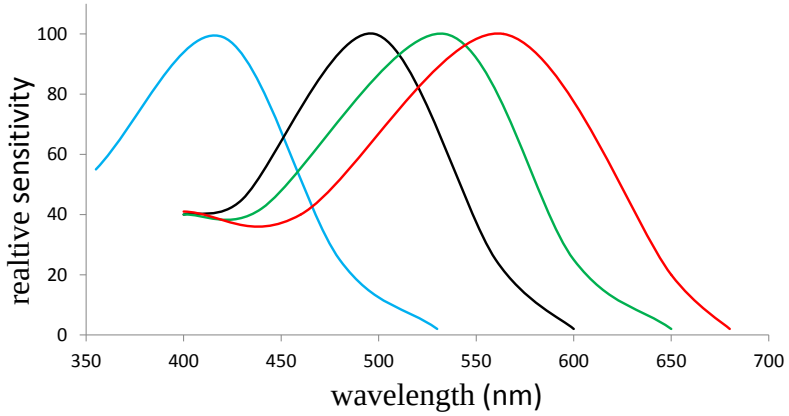


Figure 2.1: Retina response for different wavelengths (source: own sketch after [Wik])

To understand more clearly why the human visual system is so complex and why it is practical to take advantage of its features, it has to be investigated in more detail. The human eyes are ultimately the receivers of almost every processing system. They represent the sensor of visual signals and transfer the information to the human brain and can be roughly compared to a video camera. The lens of the eye though is not perfect even for people who are not near-sighted or far-sighted whatsoever.

The retina is the receptor which transforms the incoming light into electrical signals that are sent to the brain. Two types of photocells are distributed over the retina: rods and cones. Roughly 130 million rods and 6.5 million cones exist on a retina. The rods cannot perceive colors but are sensitive to shapes and do not need a lot of light. In contrast to that, cones need more light to be able to perceive colors and be able to distinguish details. There are three different kinds of cones, each one is capable to perceive one spectral distribution of a color. Figure 2.1 is a sketch which illustrates the relative sensitivity to different wavelengths of rods and cones whereas rods are illustrated as black [Enc7]. This is one of the reasons why the HVS is more sensitive to luminance changes and shapes than changes in color and exact details. This behaviour can be used to remove more redundant information from the picture and maintain a visually lossless quality [KIK85].

Actually not all information the eye catches is transferred to the brain. As the bandwidth of the optical nerve is limited, the eye compresses the signal before it is finally sent to the brain [New06] [KIK85]. There also exist further psycho-visual properties which were investigated through extensive experiments [Pla96]. Weber's law for example indicates that the HVS is more sensitive to luminance contrast than absolute luminance [KIK85] [JJS93] [Pla96].

As mentioned before perceptional coding is also used in other areas of multimedia. For example the JND/MND approach are already successful in compressing audio signals. For

audio, frequencies which an average listener cannot hear are cut. By using a threshold, most simply explained the bit-rate, more or less frequencies are cut [Pla96]. Using a bit-rate that is too low results in perceptible bad quality. For example, compressing a recording of falling rain crackling against a window or a clapping audience in a theatre can not be distinguished from static noise if the parameters are set too low.

2.3 Just-Noticeable-Distortion

As mentioned before, perceptional coding is considered a second generation coding method. Those methods can be categorized into two general categories where the first one is *local operators based* and the second *contour-texture model based*. The proposed JND measurement used in this work is part of the first category as it uses the local environment of a sample to determine the JND value. Chapter 4 goes into more detail, this section focuses on introducing the idea of the Just-Noticeable-Distortion approach used in this work.

It is not sufficient to only remove the redundant information in a picture to achieve a good compression ratio. Just-Noticeable-Distortion also tries to eliminate the perceptually insignificant components of the image. Ideally the picture is subjectively visually lossless while having the lowest possible bit-rate. Because of that, the actual coding is always near-lossless, even though the use of the JPEG-LS codec might suggest otherwise by name.

Noise in an image is perceived as distortion. If the distortion level is below the JND value, a visually lossless image can be achieved but at the expense of a higher bit-rate. Vice versa a distortion level above the JND yields a better compression ratio but the distortion is perceptible. The idea is that the distortion is distributed in a way so that it is masked by the original image signal. This is also called *distortion masking*. This masking effect results from the inability of the human visual system to distinguish properly between the noise and the input signal even though it is possible to objectively measure it with mathematical formulas. That is why the noise can be not perceptible and the concept of JND is realizable. The right calculation of JND marks the boundary value which is the limit the coder should try to match as good as possible [JJS93].

It is often not possible to maintain the JND threshold, in those situations a *Minimally-Noticeable-Distortion (MND)* is accepted. The noise may be slightly visible but should appear to be uniformly distributed over the whole picture [JJS93]. In many applications a MND is sufficient, especially when the picture has small dimensions. The perception of noise depends on various factors like luminance changes and viewing distance to the picture.

JND models themselves can further be categorized into two branches. The first one is the *pixel-wise JND model* which model comes from the image domain. The second is called *sub-band JND model* which model comes from the transformation domain like wavelet and Discrete Cosine Transformation. This work uses an approach from the pixel-wise JND model category as JPEG-LS is also processing an image in that way. The used methods and performed calculations are explained in chapter 4 in more detail.

JPEG-LS can perform both lossless and near-lossless (lossy) coding. The maximum allowed deviation between the original image value and compressed image value is set by the parameter $NEAR$. For more information about variables and processes please consult chapter 3 or the ISO/IEC 14495-1 : 1998(E) CCIT Recommendation T.87 [IT98].

Native JPEG-LS only supports a fixed non-adaptive quantization step size where the integration of JND allows to adapt the $NEAR$ value and thus the quantization step size where

$$QSS = 2 \cdot NEAR + 1$$

is true for all JPEG-LS coding steps. Without the JND adaption the compression efficiency is limited.

Summarized, the goal is to find a way to adapt the QSS to the contents of the source image and maximize the quantization step size while the resulting picture is still visually lossless.

3 JPEG-LS

JPEG-LS is a codec which allows to compress a source image lossless or near-lossless (lossy). In comparison to other coding techniques, it uses a relative to other coding techniques simple approach to effectively reduce the file size of the output file. JPEG-LS is based on the work of HP in "The LOCO-I lossless image compression algorithm" [WSS00]. The following section introduces the general application flow which is important to understand the baseline JPEG-LS [IT98] coding and decoding mechanisms.

3.1 General application flow

A source image is used as the input for the encoder. Images containing colors usually have a number of components. Components themselves are two dimensional arrays which each represent a certain spectrum of a color palette. Through combining all components the color information is mixed together so that the original color distribution is restored. As this thesis only considers grayscale images, solely one component is being considered.

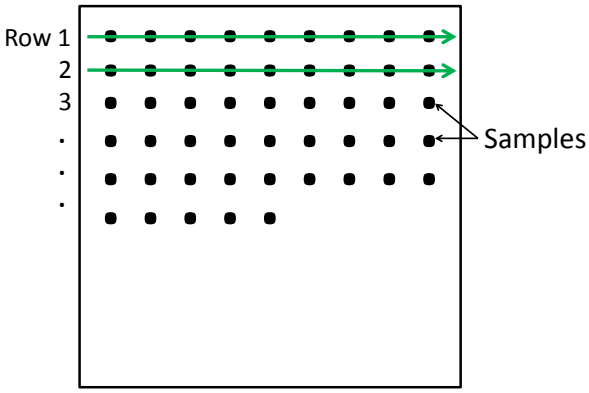


Figure 3.1: Raster scan method (source: own figure after [IT98])

Through a predefined scan order each pixel or sample from the source is processed and written into an output file by the encoder. The scan order is from top row to bottom row whereas each row's column is scanned from the left-most column to the right-most column. This method is called *raster scan*. Figure 3.1 visualizes this principle in which the green arrows symbolize the scan sequence. It is important that encoder and decoder process the information in the same way so that the values are read in the right order. Having this convention makes it needless to convey that information in the bitstream of the compressed image. A picture has a total of X columns and a total of Y rows which determine the dimensions of the source image. Figure 3.2 (a) shows the layout of a component with its orientations.

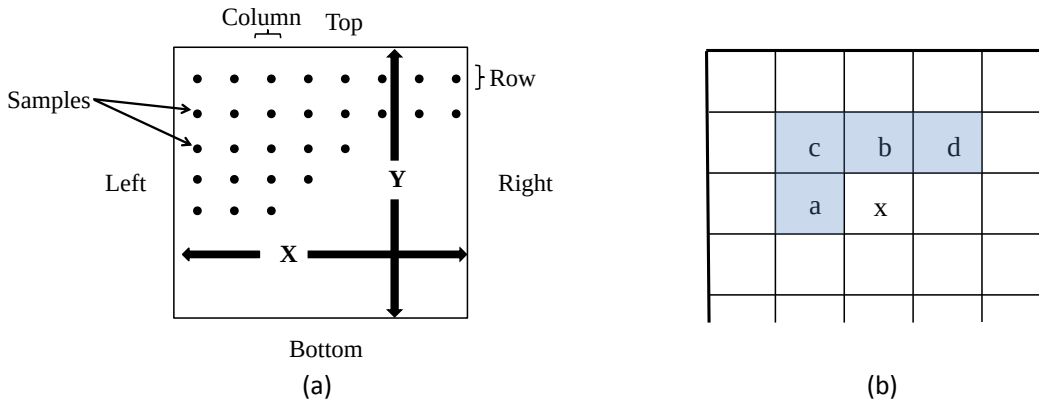


Figure 3.2: Component layout and the causal template (source: own figure after [IT98])

NEAR is a variable which determines by how much the reconstructed pixel may differ from the source image. The quantization step size (QSS) is determined by $QSS = 2 * NEAR + 1$ whereas NEAR is a positive integer with the smallest value of zero. Through setting the NEAR value the encoding mode between lossless ($NEAR = 0$) or near-lossless ($NEAR > 0$) can be set. The value for a sample Ix from the source image is also an integer with a range $2^P - 1$ whereas $P = 8$ for 8 Bit bit-depth. In this case this results in possible integer values between 0 and 255. If NEAR is set to zero, no difference between Ix and the reconstructed value Rx may occur. This is why lossless reconstruction is possible. A value of $NEAR = 3$ for example means that the sample Ix is 127, Rx may be between 124 and 130 and consequently near-lossless encoding/decoding is performed. The NEAR variable is essential for the codec as many other variables adapt to its value.

To convey the image properties to the decoder, the encoder inserts several markers. These markers or header segments are already partly defined in [IT93] but are extended or modified. A compressed image file basically consists of two parts: The header which contains various parameters required to decode the bitstream and the bitstream itself. Figure 3.3 shows the structure of a compressed image. The overall header which is comprised of all basic marker segments needed for the decoder, starts with an *start of image (SOI)* marker, followed by the *start of frame (SOF)* marker. The SOF header contains information concerning image dimensions like width and height and also the bit precision of the samples and how many components are contained. This marker is followed by a *start of scan marker (SOS)* which most importantly contains the NEAR value used by the encoder [IT98].

To determine the probability distribution of the prediction errors in JPEG-LS, the so called *context modeling* is used. Each sample x is encoded by processing four neighboring samples a,b,c and d which layout is shown in Figure 3.2 (b). This template is called *causal template* and by calculating the gradients on the template it determines whether the encoder enters the run or regular mode. Run mode is selected when all gradients are within the tolerance set by $NEAR$, otherwise the regular mode is performed. The gradient computation is shown in the next section 3.2.

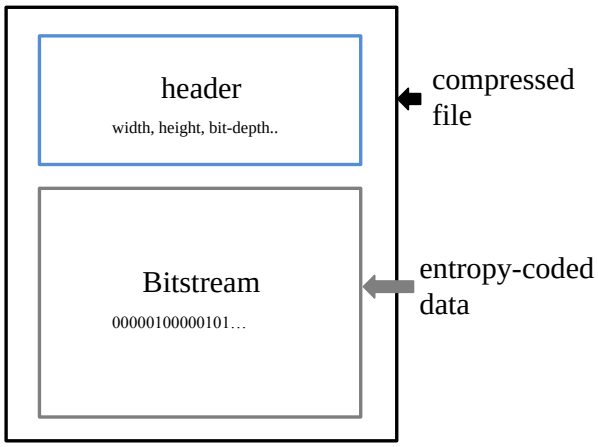


Figure 3.3: General structure of a compressed image (source: own figure after [Wan11a])

For simplicity reasons only the regular mode is implemented in this work as it is sufficient to show a proof of concept of combining JND with JPEG-LS.

The regular mode will perform a prediction and error encoding procedure. Firstly a median edge prediction is calculated from three neighbors of sample Ix. The causal template 3.2 (b) is used here while only the neighbors a, b and c are relevant. The prediction error is then determined by computing the difference between sample x and the predicted value. To negate the difference as good as possible the prediction error is therefor corrected by a context dependent term. Finally a golomb coding algorithm writes the data into the output file.

3.2 Compression steps in baseline JPEG-LS regular mode

The encoder writes all relevant marker segments into the output file and initializes needed variables. After that the actual encoding process begins by processing the source image via the raster scan. Up to the sample which is to be processed all preceding pixels are known. More exactly, all pixels on the left side and above the current sample. Special care must be given to the behaviour in corner cases where not all neighbors are available as they would exceed the image bounds. This problem is considered in [IT98] where it is assumed that there is statistically a strong correlation between the current sample and its neighbors.

For each pixel the causal template is used to calculate three local gradients D1,D2 and D3 indicated by the following equations:

$$D1 = d - b$$
$$D2 = b - c$$
$$D3 = c - a$$

If D1, D2 and D3 are all equal to zero (lossless) or D1, D2 and D3 < NEAR (near-lossless) the run mode is selected, otherwise the encoder will enter the regular coding procedure.

The more important function of the local gradients is their role in context modeling. D1-D3 show how much the value of the neighbors a,b,c and d around the current sample Ix change and determine the context. As $P = 8$ and only one component is present the possible values for the gradients are between -255 and 255. As there are three gradients, $511^3 = 133.432.831$ different contexts may occure. As that are far too many contexts, their values are quantized to a range of [-4,4]. This reduces the number of possible contexts to $9^3 = 729$.

When comparing any two contexts whose local gradient's absolute values are in pairs the same but reversed, it is observable that their absolute error values are also the same but reversed. It is possible to take advantage from that fact by removing the sign, save it separately in a variable *SIGN* and only process the absolute values of D1-D3. Performing that action reduces the number of possible contexts to only 365. To identify the specific contexts they are mapped to an index value Q and are thus being quantized. This is called *local gradient quantization*. There are three thresholds T1, T2 and T3 defined which map the values of D1-D3 to a single index value Q. The default values are $T1 = 3, T2 = 7, T3 = 21$. These thresholds have to be overwritten when near-lossless coding is performed. The following algorithm 3.1 shows the procedure on how to quantize the gradients.

Algorithm 3.1: Quantization of D_i

if $(D_i \leq -T3)$ **then**
    $=>$ *set* $Q_i = -4$;
else if $(D_i \leq -T2)$ **then**
    $=>$ *set* $Q_i = -3$;
else if $(D_i \leq -T1)$ **then**
    $=>$ *set* $Q_i = -2$;
else if $(D_i < NEAR)$ **then**
    $=>$ *set* $Q_i = -1$;
else if $(D_i \leq -NEAR)$ **then**
    $=>$ *set* $Q_i = 0$;
else if $(D_i \leq -T1)$ **then**
    $=>$ *set* $Q_i = 1$;
else if $(D_i \leq -T2)$ **then**
    $=>$ *set* $Q_i = 2$;
else if $(D_i \leq -T3)$ **then**
    $=>$ *set* $Q_i = 3$;
else
    $=>$ *set* $Q_i = 4$;
end if

The index i is between 1 and 3. After quantization the value Q is calculated by reversing all signs of the vector (Q1, Q2, Q3) and saving the sign in the variable $SIGN$ when the first

non-zero element of the vector is negative. The calculation of Q and $SIGN$ is called *context determination*.

The next step is to get an as precise as possible prediction for the sample x (Ix) out of the neighboring pixels a, b and c where the causal template from figure 3.2 (b) is used again. The predictor being used is a *Median Edge Detector (MED)* with the following formula:

$$Px = \begin{cases} min(a,b) & for\ c \geq max(a,b) \\ max(a,b) & for\ c \leq min(a,b) \\ a+b-c & else \end{cases}$$

where Px is the resulting predicted value. Px is usually not predicted correctly so that four arrays A, B, C and N are defined to correct Px. A stores the *accumulated prediction error magnitudes*, B stores the *bias* values, C stores *prediction correction* values and N counts the *frequency of occurrence* of each context. Their initial values are defined in [IT98] and will not be discussed in more detail here. As there is no information in how much Px has to be corrected when the context occurs the first time, initially nothing will be added or subtracted. But as the algorithm continues and the context occurs more often this behaviour will change. To correct the predicted value Px, $C(Q)$ with index Q is added or subtracted from Px depending on SIGN and clamped with the function $clamp(a,b)$ to the range of 0 and $MAXVAL = 2^P - 1 = 255$. The whole procedure is called *adaptive correction*.

$$Px = clamp((Px + SIGN * C(Q)), MAXVAL)$$

Now the computation of the prediction error can be done and is shown in the following equation.

$$Errval = SIGN \cdot (Ix - Px)$$

Errval should be a value as small as possible. Ideally, it is zero which would mean that the prediction was very good. The predictor only considers a very limited number of neighboring pixels and thus has a limited local view. As images also may have big homogeneous areas or textures the prediction is often not perfectly accurate. Thus through the *context modeling* (computation of D_i, Q, A, B, C and N) which is shown above, the prediction of Px is improved.

For near-lossless coding the *Errval* is further quantized. After the quantization the reconstructed value Rx is computed and is used to encode further samples. To have consistency with the decoder, the encoder computes Rx it in the same way. The algorithm 3.2 shows the procedure. For lossless coding this is not needed.

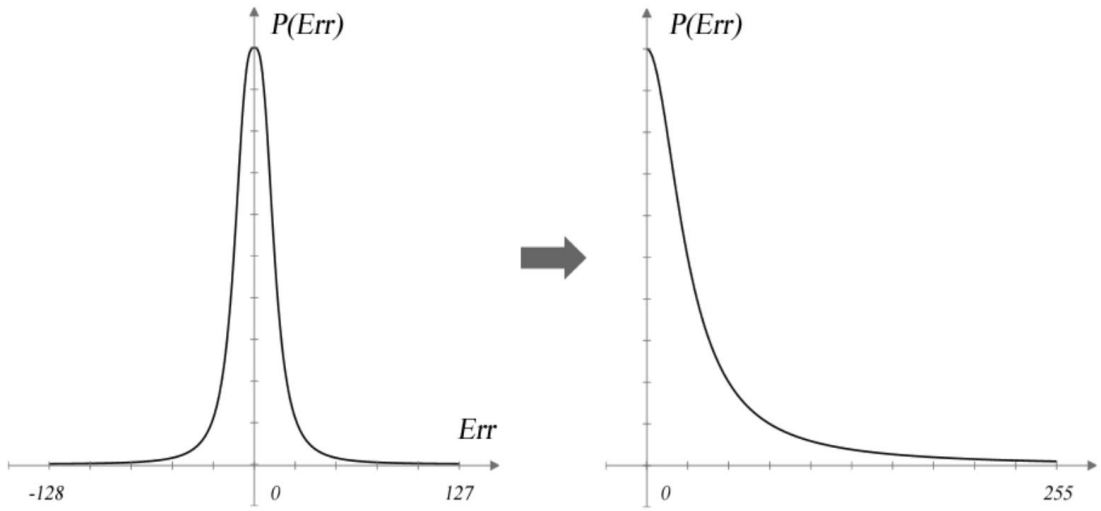
Algorithm 3.2: Error quantization for near-lossless coding and Rx computation

1: **if** $Errval > 0$ **then**
2: $\quad Errval = \frac{Errval+NEAR}{(2 \cdot NEAR+1)}$
3: **else**
4: $\quad Errval = \frac{-(NEAR-Errval)}{(2 \cdot NEAR+1)}$
5: **end if**
6: $Rx = Px + SIGN * Errval * (2 * NEAR + 1)$
7: $Rx = clamp(Rx, MAXVAL);$

The resulting range of values for *Errval* is $[-255, 255]$. That is actually not desirable as this results in a code extension from 8 Bit to 9 Bit because of the extra Bit needed for the negative sign. Thus *Errval* is set to another range by modulo reduction to a range of $[-128, 127]$.

$$Errval = \begin{cases} Errval + 256 & for\ Errval < -128 \\ Errval - 256 & for\ Errval > 127 \end{cases}$$

The error probability for real photographic pictures after figure 3.4 (a) follows a two-sided geometric distribution. This is a good prerequisite for entropy coding to reduce the amount of data. JPEG-LS uses a golomb-coder which is optimal for this case as proven in [MSW00]. As the golomb-coder only processes positive values, an *error mapping* is performed to map *Errval* to a non-negative value with a range of [0, 255]. Figure 3.4 (b) shows the distribution after the mapping procedure.



(a) Two-sided geometric error distribution (b) one-sided geometric error distribution

Figure 3.4: Geometric error distribution (source: [Ger11])

For the mapping algorithm a golomb coding variable k is needed. The computation of k is context-dependend and the variables A and N are used and computed by the encoder and decoder in the same way. The following formula finds k.

$$k < \lceil log_2(\frac{A(Q)}{N(Q)}) \rceil$$

A special mapping is performed when lossless coding ($NEAR = 0$), $k = 0$ and $(2 \cdot B(Q) \leq -N(Q))$ are true.

$$MErrval = \begin{cases} 2 \cdot Errval + 1 & \text{, for } Errval \geq 0 \\ -2 \cdot (Errval + 1) & \text{, } else \end{cases}$$

If the first criteria is not met a regular mapping is performed. For near-lossless coding, the mapping is independent of the value of k.

$$MErrval = \begin{cases} 2 \cdot Errval & \text{, for } Errval \geq 0 \\ -2 \cdot Errval - 1 & \text{, } else \end{cases}$$

After the mapping *MErrval* can finally be golomb coded and thus inserted as bytes into a bitstream. Two variables are set: $q = \frac{MErrval}{2k}$ and $qmax = 23$ for lossless or $qmax = 25$ for near-lossless grayscale 8 bit bit-depth images. The following algorithm then appends the code to the bitstream whereas the function $AppendToBitstream(n, m)$ appends m n times.

Algorithm 3.3: Golomb coding

```
1: if (q < qmax) then
2:     AppendToBitStream(0, q);
3:     AppendToBitStream(1, 1);
4:     AppendToBitStream((MErrval − (q · 2^k), k);
5: else
6:     AppendToBitStream(0, qmax);
7:     AppendToBitStream(1, 1);
8:     AppendToBitStream((MErrval − 1), qbpp);
9: end if
```

Each byte is filled with the golomb-coded bit-stream starting from its most significant bit. If the last bit does not fill up a complete byte, the remaining bits of that byte are set to 0. To avoid the accidental detection of a marker in the entropy-coded segment, *bit stuffing* is performed. If eight binary ones are written consecutively, they are followed by a binary zero. This additional zero is ignored by the decoder.

Finally as the last step in the regular mode is the *update of the variables* A, B, C and N. A and B accumulate the prediction error magnitudes and values for context Q where N counts the

Algorithm 3.4: Context variable update

$B(Q) = B(Q) + Errval \cdot (2 \cdot NEAR + 1)$;
$A(Q) = A(Q) + abs(Errval)$;
if (N(Q) = RESET) **then**
    $A(Q) = \frac{A(Q)}{2}$;
    if $(B(Q) \geq 0)$ **then**
        $B(Q) = \frac{B(Q)}{2}$;
    else
        $B(Q) = -(\frac{1-B(Q)}{2})$;
    end if
    $N(Q) = \frac{N(Q)}{2}$;
end if
$N(Q) = N(Q) + 1$;

Algorithm 3.5: Bias computation and update of the prediction correction

if $(B(Q) \leq -N(Q))$ **then**
    $B(Q) = B(Q) + N(Q)$;
    if $(C(Q) > -127)$ **then**
        $C(Q) = C(Q) - 1$;
    end if
    if $(B(Q) \leq -N(Q))$ **then**
        $B(Q) = -N(Q) + 1$;
    end if$(B(Q) > 0)$
    $B(Q) = B(Q) - N(Q)$;
    if $(C(Q) < 128)$ **then**
        $C(Q) = C(Q) + 1$;
    end if
    if $(B(Q) > 0)$ **then**
        $B(Q) = 0$;
    end if
end if

number of occurrences of the context Q. The variable $RESET$ halves A, B and N if its default value 64 is reached. The correction value C is also updated in every iteration and then clamped to their possible values. Algorithm 3.4 and respectively 3.5 show these procedures as proposed in [IT98].

The decoder follows the same procedures of the encoder but with the difference that the golomb encoding and error mapping steps are inverted resulting in *golomb decoding* and *reverse error mapping* [IT98].

4 Just-Noticeable-Distortion Calculation

Just-Noticeable-Distortion is a principle used in perceptional coding to reduce the bit-rate and thus reduce the file size of the output image. This chapter will show the advantages of this approach and explains the idea behind the implementation described in Chapter 5.

The error visibility threshold of a certain area around a pixel depends on many factors. One is the average background luminance behind the pixel and another is the spatial nonuniformity of the background luminance.

Weber's law indicates that the HVS is more sensitive to lumance contrast than absolute luminance values [CL95] [Pla96] [SS83]. If the luminance of a eye stimulus is just noticeable from the surrounding luminance, the ratio of just noticeable luminance difference is approximately constant. This is called *Weber fraction*. The noise in very dark areas tends to be less visible than noise in an environment with higher luminance due to the presence of ambient illumination. Through a modification the Weber fraction increases while the background luminance decreases when the background luminance is low. This can be used and incorporated in a coder to take advantage of by assuming high visibility thresholds in very dark or very bright areas and low thresholds in regions around the middle of the gray scale (value of 127).

The reduction of the noise visibility caused by the increase in the spatial heterogeneity of the background luminance is also known as *spacial masking*. This means that when adding noise to an image, the visibility of the noise is much higher in an uniform background region than in a region with high contrast. That feature has already been taken advantage of in many other approaches to improve the coding efficiency [NP77] [Pir81]. Nonetheless the masking effect is a highly complicated process and there exist various forms of masking. Justifying those in theoretical formulation is very difficult. Many subjective experiments were done by different research groups.

The masking function of spacial activity around a pixel is calculated by weighting each pixel as the sum of the horizontal and vertical luminance slopes at the neighboring pixels after [NP77]. The weight decreases as the distance between the neighbors and the current sample increases. A defined noise visibility function then measures the subjective magnitude as the masking function exceeds a predefined threshold. High values of the masking function lead to low values of the visibility function while high values of the visibility function in turn lead to low values of the masking function.

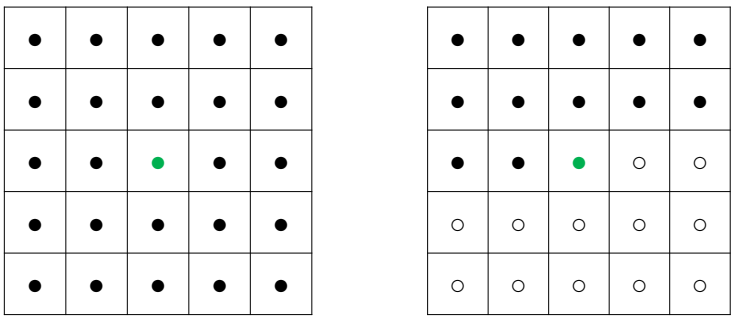
The goal is to calculate a JND value which substitutes the $NEAR$ value dynamically at runtime. Theoretically it is possible to save the different calculated JND values from the encoder in a separate table and transmit those together with the compressed output to the

decoder, resulting in a two-pass processing of an image. That would have to be repeated for every picture as the calculated JND values are only usable for a specific image. Doing that would most likely not reduce the overall file size enough to achieve a better compression ratio because of the additional JND data transmitted. Thus a better approach has to be used.

In this work a runtime JND calculation will be performed at both the encoder and decoder side which increases the runtime but in turn makes the transmission of a JND table obsolete and achieves better compression ratios. As mentioned, the $NEAR$ value is substituted which results in $QSS = 2 \cdot JND + 1$. An attempt to do that was already made in [LL10] while using a neural network to correct an estimated JND_E value with a refined JND_R one. Though neural networks are very powerful for pattern recognition they need special treatment. Essentially the neural network has to be trained, the more extensive the better. After sufficient data has been processed, the neural network codec then would be able to output a JND value which resembles the actual true JND value JND_T as much as possible by adding JND_R to JND_E. The drawback of that approach is the time needed to train the neural network and the amount of data that has to be processed in advance for the training stage. Also the processed training information has to be saved and is likely to be large.

In [CL95] the JND value is calculated from a 5x5 window which center is the current sample.

In the encoder all pixels are available so that the encoder can use all necessary information for the calculations. In the decoder though the situation is different. As JPEG-LS uses a raster scan method only the pixels up to the sample Ix that is supposed to be decoded are present. Figure 4.1 illustrates this problem, where (a) represents the status at the encoder side and (b) represents the status at the decoder side for the same sample Ix. Black dots represent available pixels, white dots are pixels which are not currently present and the green dot is the current sample Ix.



(a) Encoder view (b) Decoder view

Figure 4.1: Available pixels at encoder and decoder side (source: own picture after [LL10])

As the calculation conditions are not as good at the decoder side, the method must be adapted to the respected situation. As mentioned above in [LL10], a neural network may be

used to get a refined JND value JND_R to balance out the estimation error. In this work a more conventional approach without a neural network is taken by modifying the calculation directly in the JPEG-LS coder and finally comparing the results without calculating a JND_R value.

The JND model proposed by [CL95] uses the average gray level of the surrounding background to calculate the critical perceptual error thresholds. For simplicity and runtime reasons the correlation between the average background luminance behind the pixel and the spatial nonuniformity of the background luminance is being ignored and it is assumed that the higher value of both functions is dominating and then used as the true JND value JND_T.

The algorithm starts ba calculating the average gray level bg(x,y) through a 5x5 window and operator B around the current sample with the coordinates (x,y). Then the maximum weighted gradient mg(x,y) is found in the same 5x5 window by using different operators $G1 - G4$. After that the functions f_1 and f_2 are processed whereas f_1 calculates the spatial masking effect and f_2 returns a value to represent the visibility threshold due to background luminance. The mathematical formulas and operators are shown below.

(1) $$bg(x,y) = \frac{1}{32} \sum_{i=1}^{5} \sum_{j=1}^{5} p(x-3+i, y-3+j) \cdot B(i,j)$$

(2) $$grad_k(x,y) = \frac{1}{16} \sum_{i=1}^{5} \sum_{j=1}^{5} p(x-3+i, y-3+j) \cdot G_k(i,j) \qquad ,k = 1,2,3,4$$

(3) $$mg(x,y) = \max\{|grad_k(x,y)|\} \qquad ,k = 1,2,3,4$$

(4) $$f_1(bg(x,y), (mg(x,y)) = mg(x,y)\alpha(bg(x,y)) + \beta(bg(x,y))$$

(5) $$f_2(bg(x,y)) = \begin{cases} T_0 \cdot (1 - (bg(x,y)/127)^{1/2})) + 3 & for\ bg(x,y) \leq 127 \\ \gamma \cdot (bg(x,y) - 127) + 3 & for\ bg(x,y) > 127 \end{cases}$$

$$\alpha(bg(x,y)) = bg(x,y) \cdot 0.0001 + 0.115$$

$$\beta(bg(x,y)) = \lambda - bg(x,y) \cdot 0.01$$

T_0 in f_2 marks the visibility threshold when the background gray level is 0 and γ the slope of the line that models the function at higher background luminance. α and β are the background-luminance dependent functions, where λ in $\beta(x,y)$ affects the average amplitude of visibility threshold due to the spatial masking effects.

The operators for (2) are shown in figure 4.2 where the weighting coefficient decreases as the distance to the central pixel increases. The operator for (1) is illustrated in figure 4.3 where a weighted low-pass operator is used.

0	0	0	0	0
1	3	8	3	1
0	0	0	0	0
-1	-3	-8	-3	-1
0	0	0	0	0

G1

0	0	1	0	0
0	8	3	0	0
1	3	0	-3	-1
0	0	-3	-8	0
0	0	-1	0	0

G2

0	0	1	0	0
0	0	3	8	0
-1	-3	0	3	1
0	-8	-3	0	0
0	0	-1	0	0

G3

0	1	0	-1	0
0	3	0	-3	0
0	8	0	-8	0
0	3	0	-3	0
0	1	0	-1	0

G4

Figure 4.2: Operators for calculating the weighted average of luminance changes in four directions (source: [CL95])

1	1	1	1	1
1	2	2	2	1
1	2	0	2	1
1	2	2	2	1
1	1	1	1	1

B

Figure 4.3: Operator for calculating the average background luminance (source: [CL95])

Through experiments by Chou and Lie in [CL95] it was shown that the values for T_0, γ and λ increase accordingly to the viewing distance to the image. The used default values of the

three variables were $T_0 = 17, \gamma = \frac{3}{128}$ and $\lambda = \frac{1}{2}$ for a viewing distance of 6 times the image height [CL95].

The formulas presented above are easily calculated by the encoder and JND_T is obtained. If no changes are done at the decoder side and assuming that the non-present pixels are zero, the operators will be unbalanced. The sum of each G_i (i = [1,4]) operator equals to zero. So if half the pixels in the 5x5 input window are set to zero a false sudden drop in luminance might be introduced or assumed and returns a different JND_E value from the goal value of JND_T. The same effect will occur when instead of the 5x5 window, the operators are set to zero for the second half. This returns the same results because of the used element-by-element multiplication of the formulas between the 5x5 window and the five operators. Figure 4.4 shows an example of a modified operator G1 where the red area marks the half which is set to zero. It is clearly visible that the operator is now unbalanced.

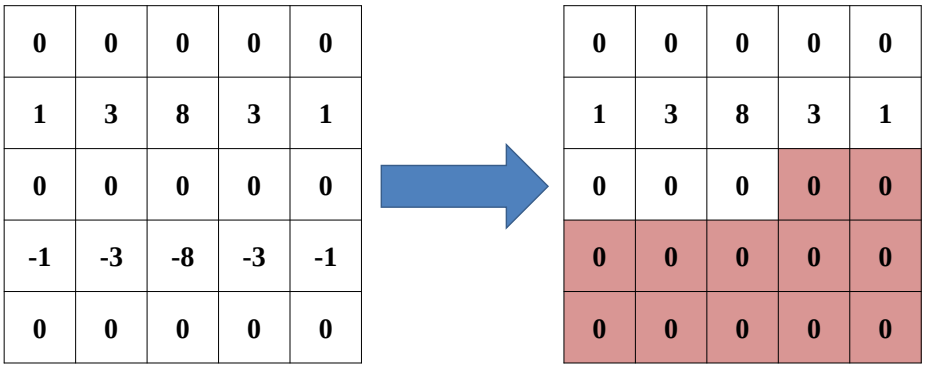
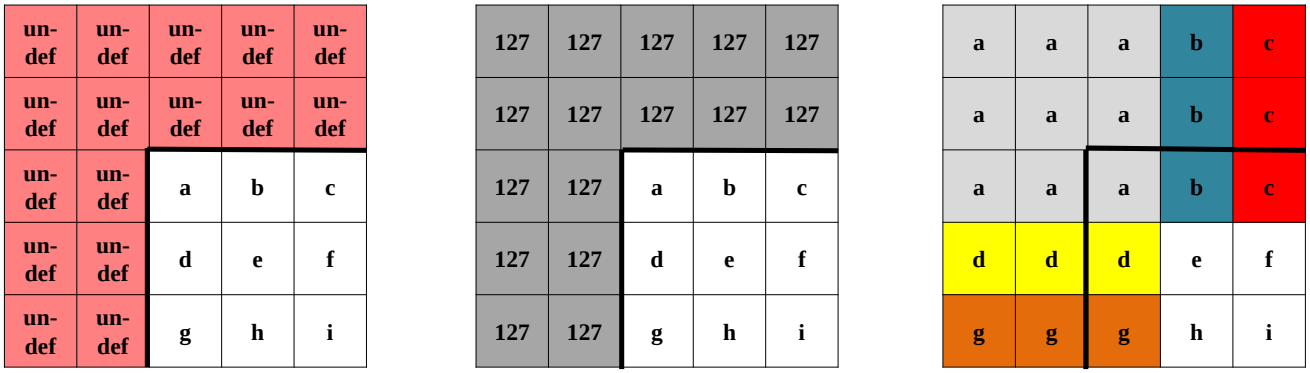


Figure 4.4: Modified G1 operator with the bottom part set to zero

Another problem are the corner cases and sidelines where the operators and 5x5 window are out of bounds of the source image. Not dealing with this problem results in access errors as the codec would try to access non-existent samples.

Figure 4.5 (a) shows the upper left initial corner case. All undefined pixels are set to zero where the letters "a" to "i" represent the image values from the source image. It would be possible to ignore the image properties and use a homogeneous template which just assumes that all unknown pixels have a middle gray value as shown in figure 4.5 (b). That reduces the detection of false luminance changes but is neither adaptive nor does it allow to get a better JND_E value in a reliable way. It would also still introduce sharp luminance changes if the borders of the original picture are very dark or very bright.



(a) initial state (b) homogeneous outer area (c) used extended matrix

Figure 4.5: States of the corner and side cases under various conditions

Instead a copy approach is used. The original picture is extended by two rows on the top and bottom of the image and also by two columns on the left and right side of the picture. That effectively extends the image matrix by 4 rows and 4 columns in total. In figure 4.5 (c) only the upper left corner case is shown as all other side and corner cases are handled analog. The corner pixel "a" is copied two rows up and two columns to the left to fill up the empty space (gray color). Also a pixel copy is performed for the four pixels in the top left edge. Then for each pixel at the beginning of the row "d" and "g" the respective value is copied to the two empty spaces to their left (yellow and orange). The same principle is used for the columns with the values of "b" and "c" (teal and red) where the values are copied two rows upwards instead. This returns an extended image which considers the image properties a little more.

Though doing that is easy on the encoder side, it is not that simple to create the full extended matrix on the decoder side. While decoding, there is no information available about the neighborhood in the beginning and at the later stages below and to the right of a current sample and the processing of the first row and first column have to be modified.

A possible way is to encode the first and last rows as well as the first and last columns with a predetermined fixed $NEAR$ value and add all reconstructed values to the beginning of the output bitstream. From that information the decoder would read that information and is able to create a frame-like matrix where the surrounding values are present and all other values are set to zero. Then the decoder can normally calculate the JND value and decode the rest of the bitstream during runtime as indicated in the original idea. But doing so would increase the file size which contradicts the goal to minimize the number of bytes added to the bitstream.

In this work a combination of both ideas is used to create the extended matrix. So that the encoder and decoder return consistent results. The encoder processes and calculates the JND values just as the decoder would. That means that only the causal pixels are regarded and JND_E instead of JND_T is used to compress the image. To create the extended matrix the

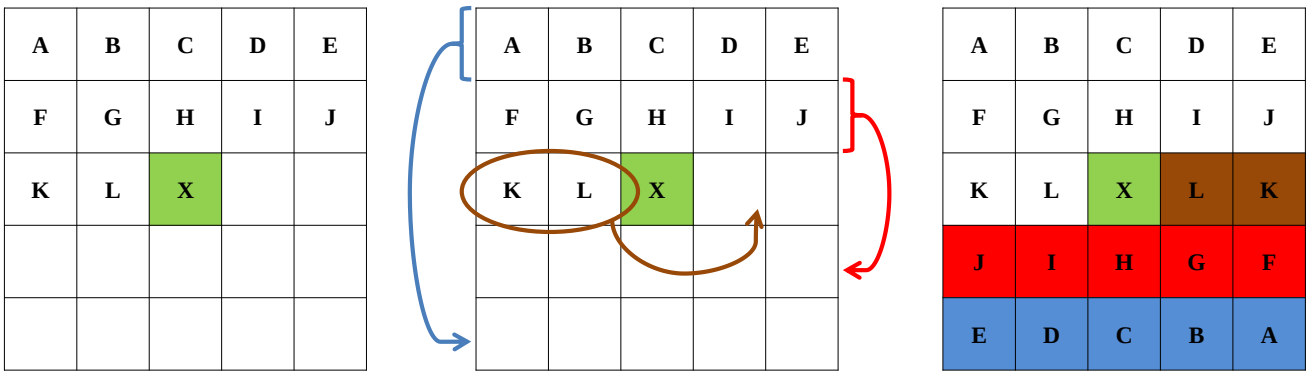
values of the sample neighborhood are inserted on the fly during runtime while the first row and column is coded with a fixed NEAR value.

It is assumed that the minimum JND value is 5 which ensures a certain minimum quality of the picture in the corner and side cases. In non-adaptive JPEG-LS coding usually a $NEAR$ value between 1 to 5 is used [LWZ+11] to guarantee image quality. A possible degradation in the image quality in the first row and first column is only visible when zooming in and examining the affected pixels closely. With that, the encoder starts with a $NEAR = 5$ and also uses this value to encode all corner and side cases for the first row and all first columns of each row. After compressing the current sample it copies the reconstructed value Rx to the respective sides and corners as described in the original idea in figure 4.5 (c).

Unfortunately as the neighborhood of a JND_E calculation is different from a JND_T calculation, the values will most of the time not be the same. In [LL10] the calculated JND_E value added to a refined JND_R value. This refinement value returns a correcting value JND_R where ideally $JND_T = JND_E + JND_R$. If $JND_T < JND_R + JND_E$, the compressed image will have a worse quality since the quantization step size is bigger than it should be. Analogous, when $JND_T > JND_R + JND_E$ a smaller QSS is used and the resulting image quality is better at the expense of a possibly higher compression ratio. Here no neural network is used, instead the calculation of JND_E is influenced directly.

In an unadapted window the initial state of figure 4.6 (a) is used where the empty cells are filled up with zeroes. That results in an effective usage of half of the picture's information and falsifies the calculated JND_E value a lot compared to later introduced methods. The operators G1 to G4 and B are not balanced anymore and will assume a sudden drop in luminance.

The first idea to get a JND_E without refinement is to mirror the current neighborhood so that no undefined values are left. This means the first two rows are copied and mirrored to the respective bottom rows and the two values left of the current sample are also copied and mirrored to the right of it. Figure 4.6 (a) shows the initial state while 4.6 (b) illustrates the window after the copy operations. Empty spaces represent undefined image values, the green cell represents the current sample and the brown, red and blue areas are the corresponding mirrored values which were copied.



(a) initial state (b) copy directions (c) final state

Figure 4.6: Mirroring the 5x5 window

Using the method of figure 4.6 returns better values for JND_E but as the pixels are perfectly mirrored, the operators G1 to G4 will always return a zero after being applied. Figure 4.7 shows a fictional 5x5 mirrored window where G3 is applied to. Empty cells represent a value of zero to focus on the relevant numbers. After performing the calculation operations the values are summed up as described in the formula by Chou and Lie [CL95]. This results in an overall value of zero as they eliminate each other.

122	134	155	144	152
152	255	102	152	119
198	197	X	197	198
119	152	102	255	152
152	144	155	134	122

(a) mirrored input values

		1		
		3	8	
-1	-3		3	1
	-8	-3		
		-1		

(b) G3 operator

		155		
		306	1216	
-198	-591		591	198
	-1216	-306		
		-155		

(c) results

Figure 4.7: Calculation example in a mirrored neighborhood with operator G3

122	134	155	144	152
152	255	102	152	119
198	197	X	197	198
119	152	102	255	152
152	144	155	134	122

(a) mirrored input values

1	1	1	1	1
1	2	2	2	1
1	2		2	1
1	2	2	2	1
1	1	1	1	1

(b) B operator

122	134	155	144	152
152	510	104	304	119
198	394		394	198
119	304	204	510	304
152	144	155	134	122

(c) results

Figure 4.8: Calculation example in a mirrored neighborhood with operator B

Figure 4.8 shows the result when the B operator is applied to the fictional mirrored neighborhood. After summing up the values of figure 4.8 (b) the final result is divided by 32 (the sum of all numbers in operator B). Actually, applying the B operator to an unchanged 5x5 window without a mirrored neighborhood and dividing the result by 16 instead of 32 would return the same results, making the neighborhood mirroring completely obsolete.

As the JND calculation algorithm only uses the maximum values of G1, G2, G3, G4 and B, solely the value of B is effectively used in the end. Even though a 5x5 window with filled up values is used and a better result compared to an unadapted neighborhood is obtained, it neglects more than half of the original JND computation idea.

The implementation used in this work is halving and modifying Chou's operators and window to only use half of the neighborhood, effectively strictly considering the causal pixels only. For that, the original operators G1 and G2 are omitted as they would produce falsified values by using an unbalanced mask. The operators G3, G4 and B are changed so that they only perform a calculation on the causal part while staying balanced. Figure 4.10 shows all three adapted operators while figure 4.9 shows the used neighborhood where the green areas are the actually used input values. The cell with the X marks the current sample. All white empty cells in figures 4.9 and 4.10 represent values of zero.

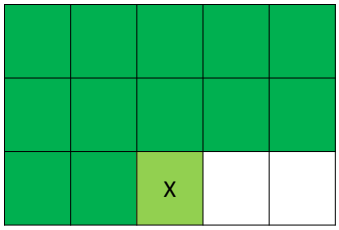
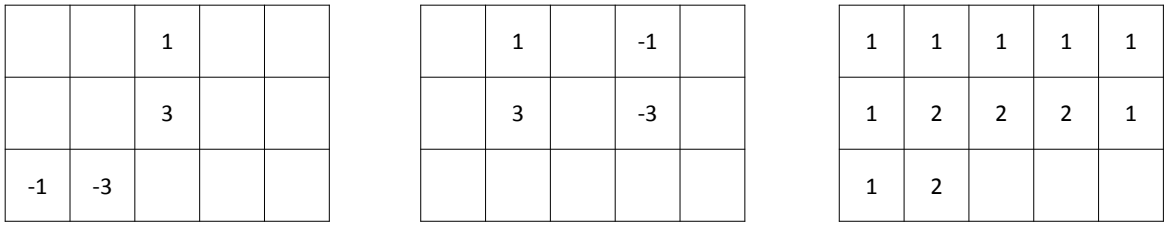


Figure 4.9: Used input neighborhood



(a) adapted G3 (b) adapted G4 (c) adapted B

Figure 4.10: Adapted operators

The first three equations of the original calculation have to be adapted to the changed environment. Below are the respective updated versions shown.

(1) $$bg(x,y) = \frac{1}{16} \sum_{i=1}^{3} \sum_{j=1}^{5} p(x-3+i, y-3+j) \cdot B(i,j)$$

(2) $$mg(x,y) = \max\{|grad_k(x,y)|\} \qquad ,k = 3,4$$

(3) $$grad_k(x,y) = \frac{1}{4} \sum_{i=1}^{3} \sum_{j=1}^{5} p(x-3+i, y-3+j) \cdot G_k(i,j) \qquad ,k = 3,4$$

The remaining original formulas are not changed. That way the algorithm does not need to be changed much to realize all approaches during testing.

The next chapter 5 shows the implementation of the described steps within this chapter. Also the adaption on the encoder and decoder side is explained.

5 Integration of the JND-Calculation with near-lossless JPEG-LS

This chapter describes the implementation of the JND calculation introduced in chapter 4 and also explains the used approach to integrate JND into a baseline regular mode JPEG-LS codec. Finally the necessary encoder and decoder adaptions are shown.

5.1 JND calculation implementation

Four different modes for the calculation of the different JND values are defined. Mode 1 refers to the JND_T calculation while the modes 2 to 4 represent the JND_E processes according to the different ideas discussed in chapter 4. Mode 2 outputs a value based on the bottom half of the input window set to zero and uses the standard operators. Mode 3 represents the calculation with a mirrored input window and also uses the standard operators. Finally mode 4 is processing a halved input window with modified operators.

Firstly the general framework of the JND calculation has to be realized. The different modes are selectable by a parameter. The function "JND_apx.m" performs all necessary calculations and is used by the encoder and decoder in the same way.

As the input image's dimensions are too small for the JND calculation by four rows and four columns, it has to be extended. The original idea to expand all rows and column in all four directions and inserting the respective values before performing any JND calculations can only be done on the encoder side. The pseudo code 5.1 shows the idea for the JND_T processing. Images will be called matrices in the following as MATLAB processes them as such.

Algorithm 5.1: Create an extended matrix for JND_T processing

```
1: function CREATE_EXTENDED_MATRIX_JND_T(input_matrix)
2:     output_matrix = enlarged input_matrix by 4 rows and 4 columns
3: //copy corners and sides
4:     output_matrix(corners) = input_matrix(corners);
5:     output_matrix(sides) = input_matrix(sides);
6: return output_matrix;
7: end function
```

As known, the decoder has no information about the pixels which follow the current sample. So the codec can still create an extended matrix, the above algorithm 5.1 has to be adapted to that situation. In all JND_E calculations and all modes the extended matrix is created during runtime on the fly. Both encoder and decoder use the same method so that the output is consistent and the results are reproduceable. The pseudo code algorithm 5.2 shows the taken approach and is called whenever Rx is calculated. The input is a matrix with the dimensions of the original to be coded image while already extended by four rows and four columns. All not yet processed cells are set to zero.

Algorithm 5.2: Create an extended matrix for JND_E processing

```
1: function CREATE_EXTENDED_MATRIX_JND_E(x, y, input_matrix)
2: //check which case is true and perform the appropiate action
3:     if input_matrix(x,y) == cornercase then
4:         output_matrix(corners) = input_matrix(corners);
5:     else if input_matrix(x,y) == sidecase then
6:         output_matrix(sides) = input_matrix(sides);
7:     end if
8: return output_matrix;
9: end function
```

All equations of the original idea of [CL95] are used and implemented where the variables are set to their default values. The different operators G1-G4 and B in their respective settings for the various modes are created as 5x5 matrices while the adapted operators are extended to fit into a 5x5 window as seen in figure 4.10. This is done so that the code does not need to be changed for different window dimensions. Applying the respective operators is sufficient to simulate the desired situations.

A function *JND_apx* contains all calculations for all modes. The parameters are coordinates (x,y) in the matrix of the current sample, the extended image and the JND mode selector. The coordinates mark the center of the 5x5 windows. At the beginning of the function this window is copied into a temporary 5x5 matrix for further calculations. By doing this it is possible to take advantage of MATLAB's matrix processing and performing an element-by-element multiplication and finally summing them up. The sub-function $perform\_calculation(input\_type)$ returns a JND value according to the parameter *input_type* which selects what kind of JND is to be calculated. The calculations of the sub-functions follow the formula by [CL95]. To execute the element-by-element multiplication with the values of the 5x5 window with the various operators the ".*" MATLAB command is used, followed by two "sum" commands to add up all values. For example to calculate the background luminance the complete command would be:

```
bg = sum(sum(5x5_neighborhood .* B_operator))
```

The following pseudo code algorithm 5.3 shows the approach.

Algorithm 5.3: Perform the JND calculation

```
1:  function JND_APX(x, y, input_matrix, JND_mode)
2:  //check which mode is selected and assign the result to JND_value
3:      if JND_mode == 1 then
4:          JND_value = perform_calculation(JND_T);
5:      else if JND_mode == 2 then
6:          JND_value = perform_calculation(JND_E with unmodified operators);
7:      else if JND_mode == 3 then
8:          JND_value = perform_calculation(JND_E with mirrored neighborhood);
9:      else if JND_mode == 4 then
10:         JND_value = perform_calculation(JND_E with adapted operators);
11:     else
12:         throw_error('invalid mode selected');
13:     end if
14: return JND_value;
15: end function
```

A special adjustment has to be done for the JND_E calculation with the adapted operators. For the operator B the division after summing up all values has to be changed from 32 to 16 as only half of the neighborhood is used (only the causal pixels). Also the divisor of 16 for the luminance change gradients has to be reduced to 4. The following MATLAB algorithm 5.4 only shows that part of the sub-function $perform\_calculation(input\_type)$.

Algorithm 5.4: Perform the JND calculation

```
1: if JND_mode == 4 then
2:     grad = abs(grad) / 4;
3:     mg_res = max(grad);
4:     bg_res = bg_res/16;
5: else
6:     grad = abs(grad) / 16;
7:     mg_res = max(grad);
8:     bg_res = bg_res/32;
9: end if
```

5.2 Encoder Adaption

Integrating the JND calculation into baseline JPEG-LS regular mode is relatively easy done. Special care has to be taken for all variables that are dependent on the NEAR value. This means that the thresholds $T1, T2$ and $T3$, the range variable $RANGE$ and the $qbpp$ all have to be adapted dynamically during runtime.

As the neighborhood for the first row and first column is unknown in the beginning of the compression, a fixed JND value is used. In non adaptive JPEG-LS a NEAR value between 1 and 5 is usually used since this ensures a certain quality of the compressed image [LWZ+11]. For every encoded sample the encoder writes back the reconstructed pixel to the input matrix as intended and also writes that value into the extended matrix and performs the "on the fly" matrix extension described in section 5.1.

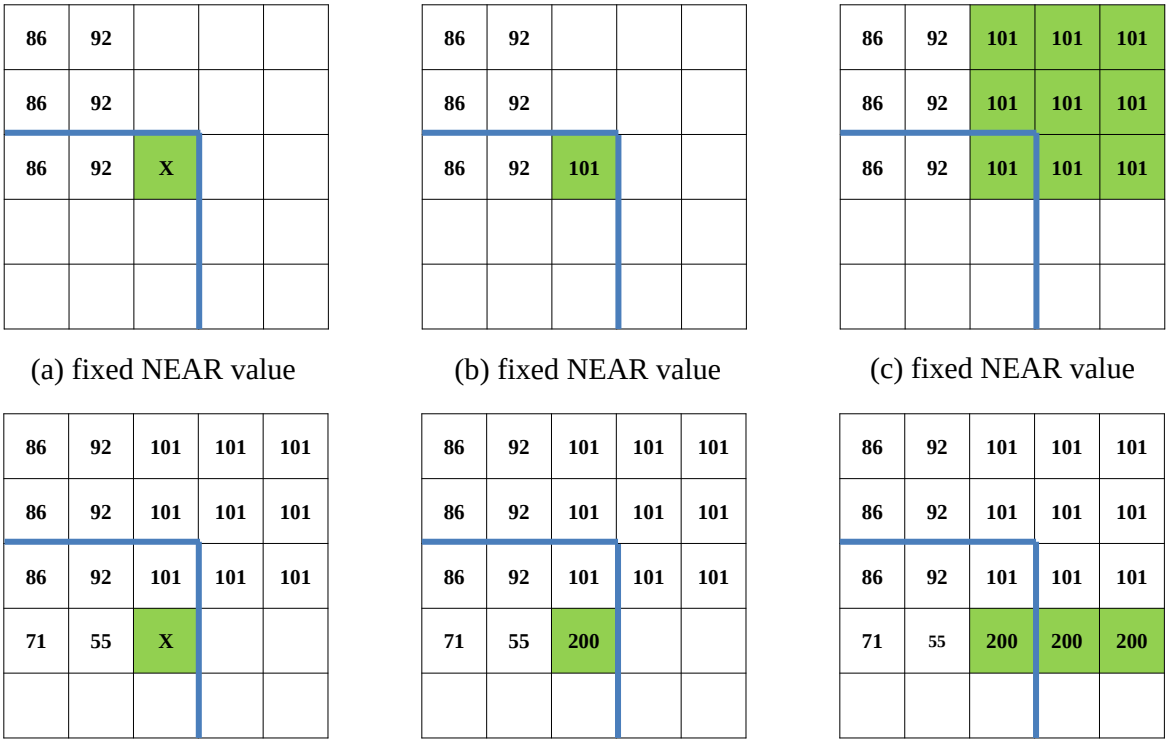


Figure 5.1: Updating the extended matrix

As only the causal pixels are relevant for the JND calculation and are all present from the first row and first column on, no further special adjustment is needed for the other side cases. Sub-figures 5.1 (a) and (d) show the status before "X" is encoded, while sub-figures 5.1 (b) and (e) show the status of the matrix after the reconstructed value is written. Finally sub-figures

5.1 (c) and (f) show the matrix after the "on the fly" matrix extension was performed. The blue lines mark the original image dimension bounds.

If the JND calculation algorithm would be applied without copying the Rx values into the extended matrix, the output would be wrong JND values. The operators would be multiplied to with zero because the extended matrix contains only zero values in every cell after initialization. This always results in a returned JND value of 20 for the first row:

(1) $$bg(x,y) = \frac{1}{32} \sum_{i=1}^{5} \sum_{j=1}^{5} 0 \cdot B(i,j) = 0$$

(2) $$grad_k(x,y) = \frac{1}{16} \sum_{i=1}^{5} \sum_{j=1}^{5} 0 \cdot G_k(i,j) = 0 \qquad ,k = 1,2,3,4$$

(3) $$mg(x,y) = \max\{|grad_k(x,y)|\} = 0 \qquad ,k = 1,2,3,4$$

(4) $$f_1(0,0) = 0 \cdot \alpha(bg(x,y)) + \beta(bg(x,y)) = \lambda = 0.5$$

(5) $$f_2(0) = T_0 \cdot (1 - (0/127)^{1/2})) + 3 = 17 + 3 = 20 \quad for\ bg(x,y) \leq 127$$

$$\text{with } \alpha(0) = 0 \cdot 0.0001 + 0.115$$
$$\beta(0) = \lambda - 0 \cdot 0.01$$

f_1 always returns a 0.5 because λ is set to 0.5 and f_2 always returns an 20 as T_0 is 17. Finally only the bigger value of f_1 and f_2 is used which is why the algorithm will always return a 20.

The special handling of the first row or column does not need to be repeated for the last row or column as all necessary pixels are already present. Figure 5.1 (a) to (c) shows the updating mechanism for the first row with the fixed NEAR value. The sub-figures 5.1 (d) to (f) show the status of the second row and last column.

Actually the last two rows of the extended matrix are not really necessary as those values will never be used anyways. They are still included so that the simple 5x5 window copy mechanism of the JND_apx function will not throw an out-of-bounds access error.

Finally, the JND calculation is called inside the raster scan loop. It can be inserted anywhere before the regular mode and the run mode selection via the local gradients D_i is called. As there exist a special case for the first row and first column, a short check has to be performed in matter of the processed sample. The pseudo algorithm 5.5 shows the JND case selection.

Algorithm 5.5: Automatic JND case selection

```
1: if row == 1 or column == 1 then
2:     set NEAR to a predefined fixed value;
3: else
4:     set NEAR to calculated JND type;
5: end if
```

In either way the variables dependant of $NEAR$ have to be adapted every time $NEAR$ is changed. Those are the thresholds T1 to T3 to calculate the index Q, $RANGE$ and $qbpp$. The following MATLAB code 5.6 needs to be executed after the NEAR value is set and before the coder calls the regular or run encoding processes.

Algorithm 5.6: Updating NEAR-dependant variables

```
1: T_i = Ti_update;
2: RANGE = ((floor((MAXVAL + 2 * NEAR) / (2 * NEAR + 1)))+1);
3: qbpp = ceil(log2(RANGE));
```

T_i is an array which stores all three thresholds values T1, T2 and T3. The updating mechanism *Ti_update* is derived from the JPEG-LS standard [IT98] which is actually a part of a procedure, initiated by a special marker called "LSE". This marker usually redefines some preset parameters when needed and it is possible to tell the coder to use new values for T1, T2, T3 and others. As all other variables are never needed to be changed and the threshold values have to be updated for every new $NEAR$ value, only the update part for the thresholds are implemented. In fact the LSE marker has various possible information stored while an identifier tells the coder what kind of information the integrated numbers represent. This data is also not used by this work which in turn makes the addition of an LSE marker obsolete. This reduces the final file size of the compressed image.

Ti_update uses the $NEAR$ value and maximum pixel value $MAXVAL$ to calculate the new states. The exact algorithm to update T_i in general can be looked up in section C.2.4.1.1.1 of the baseline JPEG-LS standard [IT98]. The specific method to update the three variables for grayscale images with a maximum value $MAXVAL$ of 255 is as follows:

(1) $T_1 = 3 + NEAR \cdot 3$
(2) $T_2 = 7 + NEAR \cdot 5$
(3) $T_3 = 21 + NEAR \cdot 7$

5.3 Decoder Adaption

JPEG-LS is a highly symmetric codec which means in a nutshell that the decoder follows the encoder steps in a reversed order. All JND algorithms discussed in the previous section 5.2 for the encoder are used at the decoder side in the same way. Thus the JND calculation is also done at the beginning of every raster scan iteration where a fixed value for the first row and first column is used. The update of T_i and all other variables are done in the same way as well as the runtime creation of the extended matrix.

Verifying that the decoded image equals the saved image that is encoded into the bitstream a simple method can be used. In lossy encoding the reconstructed pixel of the current sample is being written into the original input matrix and that value is used in all further iterations.

If the encoder reaches the end of a picture, the values in the input matrix will all have been changed to the value of their reconstructed counterparts. The decoder, if reconstructed correctly from the bitstream, will return an output matrix with the very same values. Thus the encoded image matrix just needs to be written into a bitmap file with a function. In MATLAB that is done by the following statement:

imwrite(*in_mat*, *output_name.bmp*), *'bmp'*);

in_mat is the input matrix which is to be saved while *output_name.bmp* is the name of the output file. *'bmp'* determines what kind of file type is chosen. Both encoder and decoder can use this command after processing all samples to save their results.

To compare both encoder and decoder bitmap outputs a simple comparison can be performed. The following MATLAB code 5.7 shows the used method.

Algorithm 5.7: Comparing two matrices

```
1: if matrix_1 == matrix_2 then
2:     disp('1')
3: else
4:     disp('0')
5: end if
```

All calculated JND values can be saved in a separate matrix and allow to compare the different values created by the different JND calculation modes. To check for sudden JND changes, it is possible in MATLAB to visualize the JND matrix with the following command:

imshow(uint8(JND_values*10))

where *JND_values* contains all calculated JND results and the constant 10 is needed so that the shown picture contains any visible information at all. As the JND values are usually between 3 and 25, showing those values unchanged would return a nearly completely black image where details are hard to spot.

6 Evaluation and Results

The results returned by the JND JPEG-LS codec tests have to be evaluated. First the visual quality measurement methods that were used are described. In the following section an evaluation of the test results is given.

6.1 Visual Quality Measurements

Two visual quality measurement methods were used. The first one is the *Mean Opinion Score (MOS)* and the second one is the *Multi-scale Structural SIMilarity (MS-SSIM)* test.

6.1.1 Mean Opinion Score Test

The Mean Opinion Score method is widely used in any compression related codec when the human perception system is the final receiver. The ITU recommendation P.800 and P.800.1 standardize the MOS method for acoustic cases [P.896] [P.806]. The described techniques are also usable for evaluating the quality of images.

The tests are done in a dark room with a viewing distance of six times of the image height on a 24 inch monitor. The distance and light environment are crucial factors as the coding parameters are set to that situation as proposed in [CL95]. None of the test subject are affiliated in any way with coding schemes, trained or have knowledge in how compression codecs work in detail. They are normal computer users from various fields where the range of age is between 24 and 36. Near-sighted, far-sighted and people without any viewing inhibitions took part in the tests, whereas the number of near-sighted testers dominated. Two picture test sets were used with a total of 15 participants which meets the minimum requirement of six to twelve test subjects proposed in [P.896]. The duration of each test session was a maximum of 20 minutes to ensure the necessary concentration, avoids fatigue and reduces the probability that a test subject is overwrought. Furthermore all responses are obtained independently and a short interview was held after the picture reviewing session.

The listening-only procedure from [P.896] is used while adapting it to image viewing conditions.

Two test sets are processed by the JND JPEG-LS codec whereas the test series presented to the test subjects is a selection of the standard test picture set provided by the ITU and a selection from the JPEG-LS reference images. It has to be noted that not all images can be used for the MOS test. As the dimensions of the pictures in the JPEG-LS set are sometimes

far greater than the native resolution of 1920 x 1080 of the 24 inch monitor, those images were only processed for compression ratio, runtime and MS-SSIM comparisons but are not used for the MOS test. To view such big images on a smaller screen, they would have to be scaled down. During that interpolation process the compression errors are clad. That results in an overall better perception of the image quality compared to viewing them in their native resolutions. This would falsify the overall test and thus is not included in the reviewing sessions.

Actually all images from the ITU test set are usable as their dimensions are sufficient small to be natively displayed on the test environment described above. From the JPEG-LS test set only the images BAND1, FAXBALLS, HOTEL, GOLD, CMPND1, CT, TARGET, FINGER and US can be used for the MOS test. It is not possible to perform all necessary tests with all image combinations in one session while not exceeding the time limit of 20 minutes. For that reason a further smaller selection of a total of 115 picture pairs is chosen.

The MOS test itself consists of a series of pictures which are shown in pairs. Let A be the reference image and B the test image. An equal number of A-B and B-A pairs are presented in a random order. The goal is to desensitize the tester to a fixed pattern which would unintentionally falsify the results too.

A *Degradation Category Rating (DCR)* procedure can be done where the test subject is supposed to judge weather image B has a worse quality than image A. Some *null pairs* (A-A) are included in the test to check the quality of anchoring. A five-point degradation category scale is used as follows:

5 Degradation is not visible
4 Degradation is visible but not annoying
3 Degradation is slightly annoying
2 Degradation is annoying
1 Degradation is very annoying

A second possible test method is the *Comparison Category Rating (CCR)* method. As the DCR only tests in how much the second sample's quality is *degraded* compared to the first sample a possible improvement cannot be tested. With the CCR such a comparison is easily possible. Basically the same testing method including the addition of null pairs is used but with another rating system. The category scale used is:

3 Much Better
2 Better
1 Slightly Better
0 About the Same
-1 Slightly Worse
-2 Worse
-3 Much Worse

In the test sequence not only the unprocessed reference pictures are compared to the compressed files but also compressed files with each other in different JND modes.

The used test method was the Comparison Category Rating system. Special care has to be taken where pairs are presented in the opposite order B-A. Those test values must have their signs switched during evaluation, otherwise a simple averaging of the numerical scores approximately returns zero.

6.1.2 Multi-scale Structural Similarity Test

The MOS method is a widely used technique to judge in how well a compression method preserves the original image quality. Though it is quite reliable it also has significant drawbacks. To obtain valid test data a lot of testers are needed who preferably have no affiliation with compression methods. This is often expensive, time consuming and in rapid developing sometimes not doable.

Different objective grading systems exist where the mean squared error (MSE) and the signal-to-noise ratio (PSNR) are widely used. While they are easy to calculate, have a clear physical meaning and are convenient for optimization purposes, they also have drawbacks. For example, MSE assumes that the loss of perceptual quality is directly related to the visibility of the error signal. Unfortunately as this might be plausible, two images with two different error types can have the same MSE where the error in one image is not perceivable when looked at [WBSS04].

The HVS is a complex and highly nonlinear system. Most objective grading models are based on linear or quasilinear operators which use restricted and simplistic stimuli. By examining natural images, it was observed that they are highly structured [WBSS04]. Some pixels are dependent on each other, especially when very close to each other and thus carry important information about the structure of the objects in the visual scene.

The Multi-scale Structural Similarity method compares the structures of the reference and the distorted signals, measures and uses them to approximate the perceived image quality. It is an improvement of the single-scale structural similarity method (SSIM) while the MS-SSIM can adapt to different viewing conditions [WSB03].

The basic SSIM index essentially consists of three functions where each approximates a certain aspect of the pixels. Let x and y be two discrete non-negative signals that have been aligned with each other where they represent the same spatial location from two images being compared and μ_x, σ_x^2 and σ_{xy} be the mean of x, the variance of x and the covariance of x and y respectively. With that the luminance l, contrast c and structure s comparison components for measuring are defined in [WBSS04] as follows:

$$l(x,y) = \frac{2\mu_x\mu_y+C_1}{\mu_x^2+\mu_y^2+C_1} \tag{1}$$

$$c(x,y) = \frac{2\sigma_x\sigma_y+C_2}{\sigma_x^2+\sigma_y^2+C_2} \tag{2}$$

$$s(x,y) = \frac{\sigma_{xy}+C_3}{\sigma_x\sigma_y+C_3} \tag{3}$$

where C_1, C_2 and C_3 are small constants given by

$$C_1 = (K_1L)^2, C_2 = (K_2L)^2 \text{ and } C_3 = C_2/2 \tag{4}$$

L is the dynamic range of the pixel values (L = 255 for 8 bits / pixel in grayscale images) and $K_1 \ll 1$ and $K_2 \ll 1$ are two scalar constants. With all the SSIM index between signal x and y is then:

$$SSIM(x,y) = [l(x,y)]^{\alpha} \cdot [c(x,y)]^{\beta} \cdot [s(x,y)]^{\gamma} \tag{5}$$

α, β and γ are parameters to adjust the relative importance of the respective three components.

The SSIM indexing algorithm for image quality assessment moves pixel-wise through the whole image space where the SSIM index is calculated at each step within a local window. With a reference picture defined, the SSIM index map can be viewed as the quality map of the other distorted image. Finally, a mean SSIM index of the quality map is used to evaluate the image quality.

This single-scale method is restricted as the perceptibility of image details are dependent on the fixed distance of the image to the observer and additionally by the individual who is reviewing the image quality. That is why the subjective evaluation varies if the conditions change. In [WSB03] a method is proposed which allows to simulate different viewing conditions by iteratively applying a low-pass filter and downsampling the filtered image by a factor of 2. The iteration goes from 1 to the highest scale M and in every j-th iteration the contrast c and structure s comparisons are calculated while the luminance l comparison is only computed at scale M. Combining the measurement at different scales results in the overall MS-SSIM evaluation. The following formula shows the resulting equation:

$$SSIM(x,y) = [l_M(x,y)]^{\alpha_M} \cdot \prod_{j=1}^{M} [c_j(x,y)]^{\beta_j} [s_j(x,y)]^{\gamma_j}$$

The MATLAB implementation of MS-SSIM at [Wan11b] was used to rate all compressed images in all coding modes while the original source file always served as the reference image.

6.2 Results

In this section the results from both the Mean-Opinion-Score test sessions and Multiscale-Structural Similarity grading methods are presented.

6.2.1 Mean Opinion Score Test Results

For the MOS test each of the 15 test subjects evaluated 115 pictures with the CCR grading system. The test conditions in the reviewing room was set up as described in section 6.1.1.

The overall best results and scores were given to mode 1 JND coded images which is the compression done with JND_T. Mode 3 and 4 (JND_E with mirrored window and JND_E with modified operators) got about the same scores and the differences were minimal. As expected the worst ratings were given to mode 2 where no modification to the 5x5 window or operators were done. The following consolidated tables 6.1 and 6.2 show the average scores given for each coding mode. 6.1 compares the average quality scores of the modes to the original image whereas 6.2 shows the average scores when comparing the JND coded pictures with each other for all modes. B-A pairs are added to the equivalent A-B pair average scores with a switched sign.

Mode	Avg. score
1	-1,1875
2	-2,4792
3	-0,4010
4	-0,9396

Table 6.1: Avg. MOS scores: Comparing JND images to the original images

Mode	Avg. score
1	-0,4556
2	-1,8268
3	-0,7033
4	-0,6137

Table 6.2: Avg. MOS scores: Comparing JND 1-4 coded images with each other

In the short interview held for every test subject after the reviewing session revealed that even for mode 2 the errors were not perceptible in image sections with more details. That is why the test subjects concentrated on the homogeneous areas to judge the quality differences. No differences could be seen in any mode for areas where a text section was encoded (e.g. pictures CMPND1 or CMPND2 of the JPEG set). Most apparent errors could be seen in

originally very dark or very bright sections. Those were nerved with gray lines of different shades and in other situations blockiness occurred. Unfortunately in all encoding modes, even with the encoding by using JND_T, a noticeable degradation of the image quality is perceptible in homogeneous parts of the pictures.

The unexpected slightly higher score given to mode 3 instead as expected to mode 4 lies in a unfortunate selection of test images for the MOS test. In the mode 4 test pictures more digitally generated pictures were used for the evaluation and those had many large homogeneous areas. Naturally the quality will seem to be inferior whereas for mode 3 more natural images were graded. Nevertheless the overall score difference between the two modes are not very large and confirm to a degree a very similar quality performance.

Figures 6.1 and 6.2 show the blockiness error in all JND coding modes by using the ITU image "camera" as an example. The blockiness is best seen on the man's coat. It is noticeable, that the fine details are hardly distinguishable from the original picture (a) and are thus visually lossless. Subfigure 6.2 (b) was encoded in mode 1, 6.2 (c) in mode 2, 6.2 (d) in mode 3 and 6.2 (e) in mode 4. Mode 1 produces the best result though the blockiness is still visible. Mode 3 and mode 4 coded images resemble mode 1 but the error on the coat is more apparent.

Figure 6.1: Original picture "camera"

(b) Mode 1

(c) Mode 2

(d) Mode 3

(e) Mode 4

Figure 6.2: Error example after JND coding

6.2.2 Multi-scale Structural Similarity Test Results

Originally a level depth of six was intended to be used to simulate a viewing distance of six times the image height. For that the counted weights of each level have to be defined, with the sum of all weights equals to 1. In MATLAB it is possible to solve an equation automatically with the following command:

$$\text{solve}('x^A + x^B + x^C + x^D + x^E + x^F = 1')$$

Here, x^i with $i = \{A, B, C, D, E, F\}$ represent the to be calculated weights. Tampering with the exponents A to F allows to choose the importance of a level relative to the other levels. In fact, any of the x^i can be assigned in any order to any level depth as long as each of the x^i are only assigned once.

Unfortunately the maximum usable level depth is the default depth of 5. Increasing it to 6 will downsample the smallest images "couple" and "camera" of the ITU test set too much so that they are smaller than the minimum required size of 11x11. Both those images have a width and height of 256 pixels in the original. Downsampling them five times results in a picture size of 16x16 as the downsampling factor is 2 for each level [WBSS04]. In the sixth level they will be only 8x8 pixels large which will then return the default error value of *-inf*.

To get consistent and comparable scores, all pictures have to be checked with the same parameters. The weights for each level were calculated by solving the equation

$$x^5 + x^4 + x^3 + x^2 + x = 1$$

with MATLAB with the previously shown command where the descending weight values were assigned to the levels 1 to 5 in an ascending order resulting in giving level 1 the highest weight and level 5 the lowest.

The MS-SSIM results are generally consistent with the results from the MOS test. No further parameters were changed and only the reference image with the JND coded counterparts were additionally passed as unsigned integer 8 bit matrices to the MATLAB MS-SSIM program, obtainable from [Wan11b].

Table 6.3 shows the overall average scores assigned to each JND mode when comparing those with the original image and also the minimum and maximum scores in each mode with the respected image name.

JND mode	avg. scores	min. value	image	max. value	image
mode 1	0,952784	0,86685	couple	0,99597	PC
mode 2	0,834730	0,66660	noisesquare	0,95684	PC
mode 3	0,954458	0,84263	couple	0,99477	PC
mode 4	0,954016	0,87331	couple	0,99450	PC

Table 6.3: MS-SSIM results

Further examination of the various MS-SSIM results shows that the implemented JND performs bad in dark images where the blockiness is clearly visible. On the other hand the best scores were given to the "PC" image of the JPEG test set. There are no large homogeneous areas present as the picture mainly consists of lines. The errors are very slightly visible and just by zooming in close enough.

6.3 Compression Ratio

A total of 40 individual pictures were processed in six different modes resulting in a database of 240 images. The last two modes were a regular mode JPEG-LS encoding without the JND adaptation with a fixed NEAR value of (5) three and (6) five. The other modes were coded with (1) JND_T, (2) JND_E without any adaptation, (3) JND_E with a mirrored neighborhood and finally (4) JND_E with adapted operators. Table 6.4 shows the respective average compression ratios over both ITU and JPEG test sets in all modes.

Mode	ITU test set	JPEG test set
1: JND_T	1:4.74673	1:4.70329
2: JND_E no adaption	1:6.14189	1:5.96143
3: JND_E mirrored window	1:4.59090	1:4.44951
4: JND_E adapted operators	1:4.73437	1:4.66310
5: NEAR 3	1:3.99560	1:3.77116
6: NEAR 5	1:4.70375	1:4.43850

Table 6.4: Compression ratios of various encoding modes with JPEG-LS

Naturally the compression ration in regular JPEG-LS coding without JND improves with a higher NEAR value. Coding a picture with JND_T values is used as a goal reference value. Mode 2 may return the best compression ratio but also has the worst image quality. Mode 4 resembles mode 1 a lot in the compression ratio while mode 3 is close to the target ratio of mode 1.

6.4 Runtimes

In this test the six compression methods were used and compared to each other again. The reference runtime is (5) with the fixed NEAR value of three. Table 6.5 lists the average in how fast or slow the other modes perform compared to mode (5). For example, a value of 1 represents a runtime the same as in mode 5. A value of 0.5 would mean that the compression is done twice as fast whereas a value of 2 would describe a performance twice as slow as the reference. The average times are obtained by dividing the sample runtime by the runtime with a NEAR = 3 compression.

Mode	**ITU test set runtimes**	**JPEG test set runtimes**
1: JND_T	2.6342	2.8080
2: JND_E no adaption	2.4229	2.6429
3: JND_E mirrored window	2.5755	2.9213
4: JND_E adapted operators	2.5894	2.8290
5: NEAR 3	1.0000	1.0000
6: NEAR 5	0.9632	0.9409

Table 6.5: Average runtimes relative to a normal compression with NEAR = 3

As expected the runtimes without the JND calculations no matter in which fixed NEAR value are much faster. As every sample needs its neighborhood analyzed during encoding and decoding, the total process duration increases with the image dimensions. Another observation is that with an fixed NEAR value greater than the reference NEAR value of 3, the runtimes decrease as the quantization step size is larger.

The in overall high runtimes results from the fact that no optimization was done or used within the MATLAB code. Other programming languages, taking advantage of parallel computing and reading and processing a whole scan line instead going through the image pixel by pixel can improve the runtimes significantly. The runtimes by themselves are not relevant for a proof of concept but can give a quick reference in what kind of slowdown a JND integration with JPEG-LS might cause.

7 Summary

The idea of reducing the file size of an JPEG-LS encoded image was successfully done by integrating a JND measurement.

Though the integration itself was easily done while not changing a lot in the original JPEG-LS codec, the resulting pictures had visible errors. Compressing more detailed images or pictures with few homogeneous areas was more successful as errors are just slightly visible. In those pictures a visually lossless compression or at least a minimally noticeable distortion (MND) can be achieved.

On the other hand, in pictures where the shades do not change in a larger spatial domain and especially where dark areas occur, the JND addition creates a visible degradation of the image quality. This is even true when using the JND_T values as originally introduced in [CL95]. This might result from omitting the run encoding mode of JPEG-LS which is called instead of the strictly used regular coding mode when the local gradients are smaller than the current JND/NEAR value. As such, the goal to maintain a visually lossless picture by using the proposed JND measurement was failed.

The performance in both quality and runtime may be improved by changing the JND calculation and implementing the runmode. Also optimizing the codec by changing how the input pixels are read or using parallel processing may improve the coding performance.

While only sticking to a regular coding, a simple limitation of the possible JND range improves the picture quality. Tests with a limitation to a range of five to ten reduces visible errors but may also increase the file size of the compressed image.

JND is a very promising approach to include perceptional coding in compression. The concept of the idea definitely works whereas an improvement and optimization of the quality and performance is certainly possible.

Bibliography

[Blu10] Blu-ray Disc Association. Audio Visual Application Format Specifications for BD-ROM Version 2.4, 2010. URL http://www.blu-raydisc.com/assets/Downloadablefile/BD-ROM_Audio_Visual_Application_Format_Specifications-18780.pdf. (Cited on page 10)

[CL95] C.-H. Chou, Y.-C. Li. A perceptually tuned subband image coder based on the measure of just-noticeable-distortion profile. *Circuits and Systems for Video Technology, IEEE Transactions on*, 5(6):467 –476, 1995. (Cited on pages 4, 8, 23, 24, 25, 26, 27, 30, 34, 41 and 51)

[CSBY06] I. Cheng, R. Shen, P. Boulanger, X.-D. Yang. Perceptual Analysis of Level-of-Detail: The JND Approach. In *Multimedia, 2006. ISM'06. Eighth IEEE International Symposium on*, pp. 533 –540. 2006. (Cited on page 10)

[Enc7] Encyclopaedia Britannica online. Academic edition, 2007-. URL http://www.bibliothek.uni-regensburg.de/dbinfo/frontdoor.php?titel_id=5389. (Cited on page 11)

[Ger11] Y. Gera. Parallelisierung eines JPEG-LS Decoders und Visualisierung von Hochgeschwindigkeits-Videoaufnahmen, 2011. (Cited on pages 4 and 20)

[Gle93] W. E. Glenn. Digital image compression based on visual perception and scene properties. *SMPTE Journal*, pp. 392–397, May 1993. (Cited on page 9)

[ISO98] ISO/IEC. ISO/IEC 13818-3 : 1998 - Generic coding of moving pictures and associated audio information, 1998. URL http://www.iso.org/iso/iso_catalogue/catalogue_ics/catalogue_detail_ics.htm?csnumber=26797. (Cited on page 10)

[IT93] ITU-T. ISO/IEC 10918-1 : 1993(E) CCIT Recommendation T.81 - Digital compression and coding of continuous-tone still images, 1993. URL http://www.w3.org/Graphics/JPEG/itu-t81.pdf. (Cited on page 16)

[IT98] ITU-T. ISO/IEC 14495-1 : 1998(E) CCIT Recommendation T.87 - Lossless and near-lossless compression of continuous-tone still images - Baseline, 1998. URL http://www.itu.int/rec/T-REC-T.87-199806-I/en. (Cited on pages 4, 13, 15, 16, 17, 19, 22 and 38)

[IT11] ITU-T. H.264 : Advanced video coding for generic audiovisual services, 2011. URL https://www.itu.int/rec/recommendation.asp?lang=en&parent=T-REC-H.264-201106-S. (Cited on page 10)

[JJS93] N. Jayant, J. Johnston, R. Safranek. Signal compression based on models of human perception. *Proceedings of the IEEE*, 81(10):1385 –1422, 1993. (Cited on pages 10, 11 and 12)

[KIK85] M. Kunt, A. Ikonomopoulos, M. Kocher. Second-generation image-coding techniques. *Proceedings of the IEEE*, 73(4):549 – 574, 1985. (Cited on pages 10 and 11)

[LL10] W.-N. Lie, W.-C. Liu. A Perceptually Lossless Image Compression Scheme Based on JND Refinement by Neural Network. In *Image and Video Technology (PSIVT), 2010 Fourth Pacific-Rim Symposium on*, pp. 220 –225. 2010. (Cited on pages 4, 24 and 29)

[LWZ+11] X. Li, Y. Wang, D. Zhao, T. Jiang, N. Zhang. Joint just noticeable difference model based on depth perception for stereoscopic images. In *Visual Communications and Image Processing (VCIP), 2011 IEEE*, pp. 1 –4. 2011. (Cited on pages 10, 29 and 36)

[MSW00] N. Merhav, G. Seroussi, M. Weinberger. Optimal prefix codes for sources with two-sided geometric distributions. *Information Theory, IEEE Transactions on*, 46(1):121 –135, 2000. (Cited on page 20)

[New06] NewScientist. Calculating the speed of sight, 2006. URL http://www.newscientist.com/article/dn9633-calculating-the-speed-of-sight.html. (Cited on page 11)

[NP77] A. Netravali, B. Prasada. Adaptive quantization of picture signals using spatial masking. *Proceedings of the IEEE*, 65(4):536 – 548, 1977. (Cited on page 23)

[P.896] SERIES P: TELEPHONE TRANSMISSION QUALITY - METHODS FOR SUBJECTIVE DETERMINATION OF TRANSMISSION QUALITY, 1996. URL https://www.itu.int/rec/T-REC-P.800/en. (Cited on page 41)

[P.806] SERIES P: TELEPHONE TRANSMISSION QUALITY, TELEPHONE INSTALLATIONS, LOCAL LINE NETWORKS - SERIES P: TELEPHONE TRANSMISSION QUALITY, TELEPHONE INSTALLATIONS, LOCAL LINE NETWORKS, 2006. URL https://www.itu.int/rec/T-REC-P.800.1/en. (Cited on page 41)

[Pir81] P. Pirsch. Design of DPCM Quantizers for Video Signals Using Subjective Tests. *Communications, IEEE Transactions on*, 29(7):990 – 1000, 1981. (Cited on page 23)

[Pla96] J. R. C. Pla. *Image Compression based on Perceptual Coding Techniques*. Ph.D. thesis, Universitat Politica de Catalunya, 1996. URL http://www.tesisenred.net/bitstream/handle/10803/6920/01JRc01de01.pdf?sequence=1. (Cited on pages 9, 10, 11, 12 and 23)

[RMB94] M. Reid, R. Millar, N. Black. A comparison of first generation image coding techniques applied to the same magnetic resonance image. In *Engineering in Medicine and Biology Society, 1994. Engineering Advances: New Opportunities for Biomedical Engineers. Proceedings of the 16th Annual International Conference of the IEEE*, pp. 586 –587 vol.1. 1994. (Cited on page 10)

[SS83] D. Swift, R. Smith. Spacial Frequency Masking and Webers Law. *Vision Research*, 23:495–505, 1983. URL http://deepblue.lib.umich.edu/bitstream/2027.42/25421/1/0000870.pdf. (Cited on page 23)

[Wan11a] Z. Wang. Data Compression Excercise, 2011. URL http://www.ipvs.uni-stuttgart.de/abteilungen/pas/lehre/lehrveranstaltungen/ss11/dk_ue_termine/start. (Cited on pages 4 and 17)

[Wan11b] Z. Wang. IW-SSIM: Information Content Weighted Structural Similarity Index for Image Quality Assessment, 2011. URL https://ece.uwaterloo.ca/~z70wang/research/iwssim/. (Cited on pages 44 and 48)

[WBSS04] Z. Wang, A. C. Bovik, H. R. Sheikh, E. P. Simoncelli. Image Quality Assessment: From Error Visibility to Structural Similarity. *IEEE TRANSACTIONS ON IMAGE PROCESSING*, 13(4):600–612, 2004. (Cited on pages 43 and 48)

[Wik] Wikipedia.de. Netzhaut. URL https://de.wikipedia.org/wiki/Netzhaut. (Cited on pages 4 and 11)

[WSB03] Z. Wang, E. P. Simoncelli, A. C. Bovik. Multi-Scale Structural Similarity for Image Quality Assessment. In *in Proc. IEEE Asilomar Conf. on Signals, Systems, and Computers, (Asilomar*, pp. 1398–1402. 2003. (Cited on pages 43 and 44)

[WSS00] M. J. Weinberger, G. Seroussi, G. Sapiro. The LOCO-I lossless image compression algorithm: principles and standardization into JPEG-LS. *IEEE Transactions on Image Processing*, 9(8):1309–1324, 2000. (Cited on page 15)

All links were last followed on May 28, 2012.

First and foremost I would like to thank Prof. Dr.-Ing. Sven Simon of University of Stuttgart for giving me the opportunity to write this study thesis at his institute.

I also want to thank my supervisor Mr. M. Sc. Zhe Wang of University of Stuttgart who supported me during my study thesis and always pointed me into the right direction. His advices and patience made it possible to make progress whenever I was stuck.

Of course, many thanks goes to all the voluntary test subjects for the MOS test. There was always a friendly atmosphere during the review sessions and it is very much appreciated that everyone had time to help. I would like to thank (in alphabetical order):

Alahe, Will

Feig, Michael

Harnisch, Andreas

Lüpke, Marvin

Maringolo, Giuseppina

Reinemann, Bianca

Reinemann, Gerd

Renner, Stefan

Reuschel, Andreas

Rios, Antonia

Sigloch, Philipp

Waizenegger, Tim

Wenz, Stefan

Wirth, Anton Sang Yeob

Zimpel, Martina

Declaration

All the work contained within this thesis, except where otherwise acknowledged, was solely the effort of the author. At no stage was any collaboration entered into with any other party.

(Kai Liu)