Lossless image compression using n-ary context tree

Nadezda Nechaeva

06.02.2006

University of Joensuu

Department of Computer Science

Master's Thesis

Abstract

In this thesis various methods for lossless compression of source image data are analyzed and discussed. The main focus in this work is lossless compression algorithms based on context modeling using tree structure.

The central aspect in context modeling is different context templates, which are based on discrete wavelet transform coefficients, local gradients and intensity of samples in the image.

This work include research on how to use n-ary context tree structure, prediction modeling and probability assignment in lossless image compression based on context modeling technique.

The main advantage over current methods is increasing effectiveness of image compression and developing new lossless compression methods based on context modeling for different type grayscale images: medical, astronomical, noisy natural images.

Contents

1. Introduction

1.1 Motivation

In recent years, there has been a vivid interest in compression of “electronic” or digital image data. In particular, image compression is considered very useful in multimedia applications and in distributed information systems that operate in network environments. Especially, lossless compression is very useful for medical images, where loss of information is forbidden. Our main interest belongs to the lossless case, when image do not reduce any information in compression.

However, the vast majority research and development in image compression area is focused on lossy compression. It is agreed upon of fact that by allowing a small amount of distortion in the reconstructed picture one can obtain much better compression than is possible for lossless coding. From information theory it is known principle about tradeoff between the compression performance and the amount of distortion in the restored picture. Such tradeoff was formalized in rate-distortion theory of Shannon [Shan48]. In such case lossless compression can only provide moderate compression performance. Nevertheless, there are situations where only lossless (or near lossless) compression can be used, for instance, in medical images.

One of developing direction in lossless compression methods is context modeling. In general, basis of lossless compression schemes is paradigm applying “universal modelling and coding” proposed by Rissanen and Langdon in 1981 [RL81]. Context modeling is one of the varieties of modeling step. At present time compression methods are in existence, which are based on context modeling, for instance, CALIC and LOCO-I [WM97a, WSS96]. In this thesis different approaches of context modeling with applying different schemes for lossless compression are considered. One of these is context tree structure, which allows convenient storing and processing information during construction of the probabilistic model of source image data.

For lossless compression algorithms in this thesis the wavelet analysis also is used. Last decade a new concept known as wavelet was introduced. Currently, wavelets are widely applied to pattern analysis, at processing and synthesis of various signals, for example speech, medical; for studying properties of turbulent fields and in many other cases. Mallat was first, who applied wavelet to compression of images [Mallat89].

1.2 Problem formulation

The goals of this thesis can be formulated as follows:

1. Designing new context templates, which can be used for context modeling by applying the context tree with optimal and incomplete structure, for lossless compression algorithms;
2. To apply the context tree structure to algorithms, which realize lossless compression by context modeling based on various templates;
3. To analyze the context modeling approach with using n-ary general context tree;
4. Investigation of lossless compression methods: JPEG2000, LOCO-I, CALIC, which are based on paradigm of modeling and coding. Here the methods, which use context modeling, the central interest are represented.

During investigation of known lossless compression methods try to use its strong sides for developing new approaches of context modeling. For developing new templates to use the discrete wavelet transform coefficients and local gradients (this term borrows from LOCO-I method). In new methods to apply n-ary context tree with incomplete and optimal structure [AKF05] for effective processing of statistical information during the context modeling based on different templates.

1.3 Structure of the thesis

In Section 2, basic concepts of image compression are represented; measures for estimating effectiveness of compression algorithm are considered. Arithmetic coding is described, because it is a part of compression in many algorithms based on “modeling and coding” principle, and as coding usual used arithmetic coding. In Section 3, known lossless compression algorithms are described: JPEG2000, LOCO-I, CALIC; its schemes and advantages here are considered. In Section 4, common concepts of context modeling technique are described; this section contains full description of n-ary general context tree. Section 5 contains description of General Context Tree based on Intensity algorithm, which uses context modeling approach. Section 6 describes compression algorithm of General Context Tree based on discrete wavelet transform (DWT) coefficients, which uses special context template based on DWT coefficients. In Section 7, the compression algorithm of General Context Tree based on local gradient is represented; the different point here is using the local gradient as template for context modeling. Comparable analisys of main features for considered compression algorithms is in Section 8. In Section 9, the comparable analysis of compression efficiency between new and known methods of lossless compression is represented; also here are empirical experiments, which were produced during developing of new algorithms. Conclusions of the thesis are in Section 10.

2 Basic concepts in image compression

2.1 Lossless and lossy cases

The assignment of compression is to code the image data into a compact form, minimizing both the number of bits in the representation, and the distortion caused by the compression. The fundamental principle for all compression methods is following idea: if represent oft-recurring elements as short codes and rare-recurring as long codes, then the block of data needs a smaller memory size than if all elements were represented by codes of identical length.

A compression algorithm is “lossless” (or reversible) if the decompressed image is identical with the original. Respectively, a compression method is “lossy” (or irreversible) if the reconstructed image is only an approximation of the original one [Fränti00].

Some loss of information can be acceptable for the following three reasons:

1. Significant loss can often be tolerated by the human visual system without interfering with perception of the scene content.
2. In most cases, digital input to the compression algorithm itself is an imperfect representation of the real world scene. This is certainly true when the image sample values are quantized version of the real-valued quantities.
3. Lossless compression is usually incapable of achieving the high compression requirements of many storage and distribution applications.

The term lossy is used in an abstract sense, and does not mean random lost pixels, but instead means loss of a quantity such as a frequency component, or perhaps loss of noise. The fundamental question of lossy compression methods is where to lose information.

Nevertheless, the lossless compression is often applied in medical applications, because on such images all information has big significance and lossy compression here is intolerable. Lossless compression is also applied in cases where it is difficult to determine how to introduce an acceptable loss, which will increase compression. In palletized color images, for example, a small error in the numeric sample value may have an intense effect upon the color representation. Finally, lossless compression may be appropriate in applications where the image is to be extensively edited and recompressed so that the accumulation of errors from multiple lossy compression operations may become unacceptable.

In the definition of lossless and lossy compression, it is assumed that the original image is in digital form. For compression digital images are used; but source may be in analog view in the real world, and therefore, the loss in image quality already takes place in digitalization of source images, when the picture is converted from analog to digital representation. For simplicity, in compression the digitalization phase is skipped, images are stored in digital form.

2.2 Measures of compression

The compression methods are evaluated by two main criteria: *compression efficiency* and *distortion*. The most obvious measures of compression efficiency are "bit-rate" and "compression ratio". For our purposes an image is a two dimensional sequence of sample values:

$$x[n_1, n_2],\ 0 \le n_1 < N_1,\ 0 \le n_2 < N_2\,, \tag{2.1}$$

having finite size, N_1 and N_2, in vertical and horizontal directions respectively. The sample value $x[n_1, n_2]$ of source image is intensity of the location $[n_1, n_2]$ and can have the following values:

$$x[n_1, n_2] \in \{0, 1, \ldots, 2^B - 1\} \tag{2.2}$$

for unsigned imagery, where B is the number of bits on each sample. The purpose of image compression is image representation with a string of binary digits or "bits", called the compressed "bit stream", denoted as *C*. The objective is to keep the length $\|C\|$ as small as possible. In the absence of any compression, we require $N_1 N_2 B$ bits to represent the image sample values. Let us define the compression ratio as following equation:

$$\textit{compression ratio} = \frac{N_1 N_2 B}{\|C\|} \tag{2.3}$$

Equivalently, we define the compressed bit-rate, expressed in bps (bits per sample), as follows:

$$\textit{bit-rate} = \frac{\|C\|}{N_1 N_2} \tag{2.4}$$

Bit-rate is the most obvious measure of compression efficiency; it shows the average number of bits per stored pixel of the image. For lossy compression bit-rate is a more meaningful performance for image compression systems, since the least significant bits of high bit-depth imagery can often be excluded without significant visual distortion. The average number of bits spent in representing each image sample is often a more meaningful measure of compression performance, because it is independent of the precision with which original samples were represented. If the image is displayed or printed with physical size regardless the size of samples, then more meaningful measure in such case is the size of the bit-stream. Such situation is typical for lossy compression, the bit-rate is a meaningful measure only when N_1 and N_2 are proportional to the physical dimensions with which the image is to be printed or displayed.

Compression algorithms are also estimated by *distortion* measure, i.e. compression error. The more distortion we allow, the smaller the compression representation can be. The primary goal of lossy compression is to minimize the number of bits required to represent an image with an allowable level of distortion. The measure of distortion is an important feature for lossy compression.

Formally, distortion is calculated between the original image, $x \equiv x[n_1, n_2]$, and the reconstructed image, $\hat{x} \equiv \hat{x}[n_1, n_2]$. The quantitative distortion of the reconstructed image is measured by the *mean absolute error* (MAE), *mean square error* (MSE), or *peak-to-peak signal to noise ratio* ($PSNR$):

$$MAE = \frac{1}{N_1 N_2} \sum_{n_1=0}^{N_1-1} \sum_{n_2=0}^{N_2-1} \left| \hat{x}[n_1, n_2] - x[n_1, n_2] \right| \tag{2.5}$$

$$MSE = \frac{1}{N_1 N_2} \sum_{n_1=0}^{N_1-1} \sum_{n_2=0}^{N_2-1} \left(x[n_1, n_2] - \hat{x}[n_1, n_2] \right)^2 \tag{2.6}$$

$$PSNR = 10 \log_{10} \frac{(2^B - 1)}{MSE} \tag{2.7}$$

The $PSNR$ is expressed in dB (decibels), good reconstructed images typically have value of 30dB.

For estimation of compression method the following measures are applied: speed of compression, robustness against transmission errors and memory requirements of the algorithm. For estimation efficiency of algorithms in this thesis we will use the value of bit-rate.

2.3 Paradigm of compression

Compression methods of context modeling for data compression are based on a paradigm of compression with applying "universal modelling and coding" proposed by Rissanen and

Langdon in 1981 [RL81]. According to the given approach the compression process consists of two separate parts:

- modeling;
- coding.

Modeling assigns probabilities to the symbols, and coding produces a bit sequence from these probabilities. This concept is illustrated in Figure 2-1.

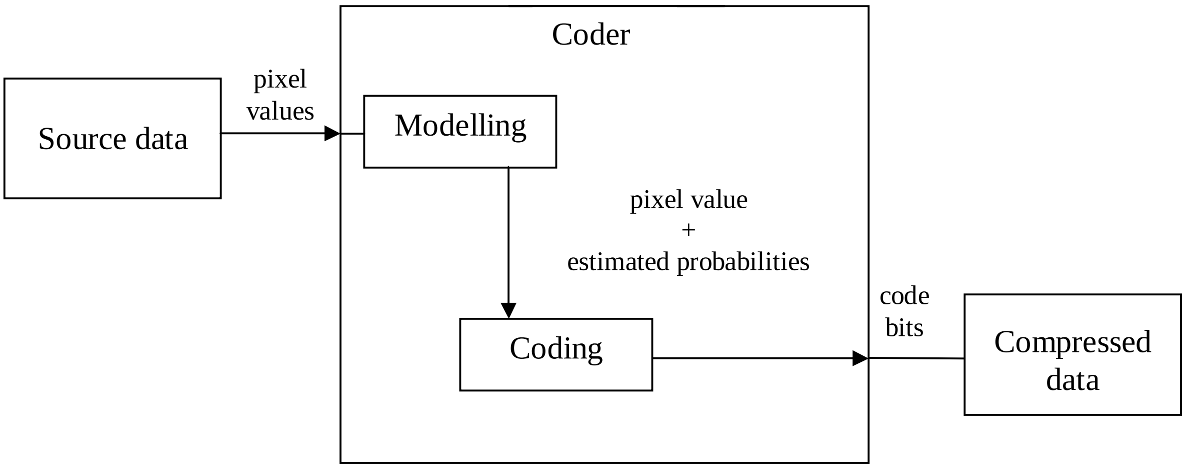


Figure 2-1. Principle compression scheme based on concept "universal modeling and coding".

Decompression scheme is symmetrical to compression scheme illustrated in Figure 2-1. The same model for both coder and decoder is used.

For all compression methods exists following common principle: if representation of oft-recurring elements is short codes and representation of rare-recurring elements is long codes, then the block of data needs a smaller memory size than if all elements were represented by codes of equal length. Exact connection between the symbol probability and its code was first established in Shannon's "noiseless source coding theorem" [Shan48]. The essence of this theorem is that element s_i with probability $p(s_i)$ is represented more advantageous by code with length $-\log_2 p(s_i)$ bits. If during the coding process length of codes equals exactly $-\log_2 p(s_i)$ bits, it means that length of coded bit-stream is minimal for all possible compression methods. Such value is denoted as *entropy*:

$$H(x) = -\log_2 p(s_i) \tag{2.8}$$

and means information content of an element s_i in the alphabet. Here source alphabet is a set of all possible non-recurrent elements from source image. The entropy rate of a random process provides a lower bound of the average number of bits that must be spent in coding each of its outcomes, and this bound may be approached arbitrary closely as the complexity of the coding scheme is allowed to grow without bound.

If the probability distribution $F = \{p(s_i)\}$ is invariable and probabilities $p(s_i)$ are independent, then the average code length is given by:

$$H(x) = -\sum_{i=1}^{k} p(s_i) \cdot \log_2 p(s_i) \qquad (2.9)$$

where k is the number of elements (or symbols) in the alphabet. This value also means *entropy of the probability distribution.*

In order to achieve good compression rate, exact probability estimation is needed. The more accurately the probabilities of symbols occurrence are estimated, the more closely codelength correspond to the optimal, and the better compression is.

Since the model is responsible for the probability estimation of each symbol, statistical modeling is one the most important tasks in data compression. It can be classified into the following three categories:

- *Static* modeling;
- *Semi-adaptive* modeling;
- *Adaptive* (or *dynamic*) modeling.

Static modeling is the simplest case. In the static modelling the same model (code table) is applyed to all input data ever to be coded. Code table with predefined alphabet is constructed based on a test set of data used this alphabet. Unfortunately the static modelling fails if the input data does not base on the same alphabet as model. The advantage of static modelling, is that no side information is needed to transmit to the decoder, and that the compression can be done by one-pass over the input data.

Semi-adaptive modeling analyzes the source data before coding. Probability distribution for symbols coincides to source data stream because the code table is calculated after analyzing the input stream. The main feature for semi-adaptive modeling is collection of statistical information from the source data, and the encoding is based on the semi-adaptive code table. Disadvantage of semi-adaptive modeling is that constructed model must be stored in the compressed file.

Adaptive model changes the symbol probabilities during the compression process in order to adapt the statistics during the process. Initially the compression process starts with an initial model, so the model does not need to be transmitted. During the process, the model adaptes by the symbols, which have been transmitted already. It is important that the model gets adapted only by the symbols, which have been transmitted already, since the decoder needs to be able to adapt the model in the same way later at the decompression process. Since the decoder knows the before transmitted symbols, it can adapt to the model in the same way than the coder did.

The properties of different modeling strategies are summarized as follows [Fränti00]:

Static modelling:	**Semi-adaptive modelling:**	**Dynamic modelling:**
+ One-pass method	- Two-pass method	+ One-pass method
+ No side information	- Side information needed	+ No side information
- Non-adaptive	+ Adaptive	+ Adaptive
+ No updating of model during compression	+ No updating of model during compression	- Updating of model during compression

2.4 Arithmetic coding

Arithmetic coding is known as optimal coding method. In respect to Shannon's theorem [Shan48], the best possible code contains a contribution of $-\log_2 p$ bits from the encoding of each symbol whose probability of occurrence is p.

The most important advantage of arithmetic coding is its flexibility, which means that it can be used in conjunction with any model that can provide a sequence of probabilities. For example, adaptive statistical models may be used, because the coding process does not depend on the modeling part, i.e. this great flexibility is result from the separation of the coder from the modeling process [RL81].

In arithmetic coding, the data is represented by an interval of real numbers between 0 and 1. During the processing, the interval needed to represent the data becomes smaller, and the number of bits needed to specify this interval grows. Successive symbols reduce the size of the interval with accordance to the probabilities of the symbols generated by the model. More likely symbols reduce the range by less than unlikely symbols, and hence, add fewer bits to the compressed data.

Arithmetic coding contains the following steps:

1. The coding process begins with a "current interval" $[L, H)$ initialized to $[0,1)$
2. For each symbol of the source data stream two steps are performed (see Figure 2-2):
   2.1 The current interval $[L, H)$ is divided into subintervals, one for each possible alphabet symbol. The symbol's subinterval has size that is proportional to the estimated probability of the symbol will be the next symbol in the source data stream, according to the model.

2.2 The subinterval $[L', H')$ corresponding to the symbol that actually occurs next will become as the new current interval.

3 Final interval is coded by enough bits to distinguish it from all other possible intervals.

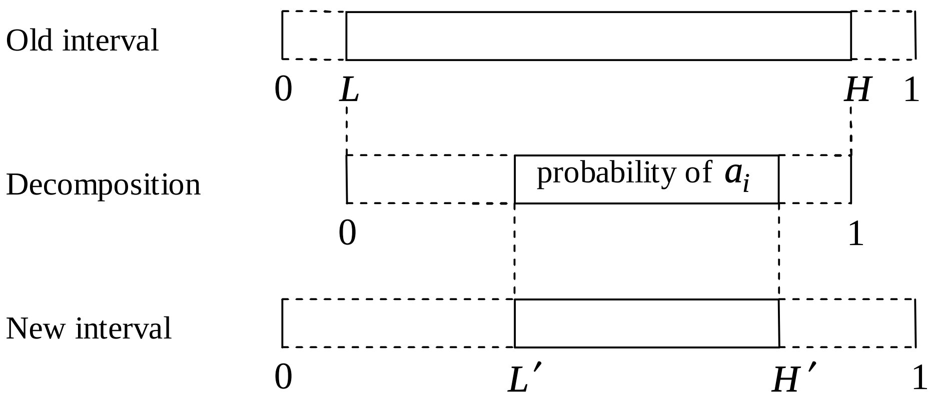


Figure 2-2. Subdivision of the current interval based on the probability of the input symbol a_i.

In step 2, subinterval corresponding to the occurred symbol a_i is computed. For this calculation following cumulative probabilities are used:

$$P_C = \sum_{k=1}^{i-1} p_{k,}\ P_N = \sum_{k=1}^{i} p_k \tag{2.10}$$

The new subinterval is given by:

$$[L + P_C(H - L), L + P_N(H - L)) \tag{2.11}$$

The product of the probabilities of the individual alphabet symbols is the length of the final subinterval, which equals to the probability p of the particular sequence of symbols in the source data stream.

Let us consider the fundamental properties of binary arithmetic. With n bits, at most 2^n different combinations can be represented, and vise versa, with n bits the code interval $[0,1)$ can be divided into 2^n parts each having the length 2^{-n}. This dividing process is illustrated in Figure 2-3.

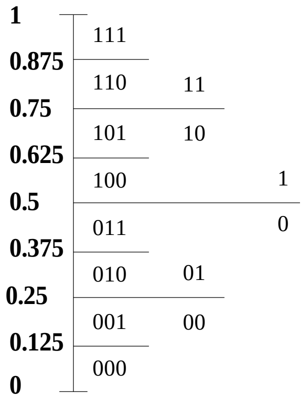


Figure 2-3. Interval $[0,1)$ is divided into 8 parts, each part has the length of $2^{-3} = 0.125$. Each interval can now be coded by using $-\log_2 0.125 = 3$ bits.

Let us denote the length of interval by A, if $A = 2^{-n}$, the interval with the length A can be coded using $-\log_2 A$ bits (assuming A is a power of 2). As described above the basic idea of arithmetic coding is to represent the source data stream as a small interval between 0 and 1. The coding process is the binary code representation of interval, which takes $-\log_2 A$ bits. Thus, the final interval is a product of the probabilities of the coded symbols:

$$A_{final} = \prod_{i=1}^{n} p_i \tag{2.12}$$

where p_i is the probability of the i^{th} alphabet symbol. The interval can be coded by:

$$C(A) = -\log_2 \prod_{i=1}^{n} p_i = -\sum_{i=1}^{n} \log_2 p_i \tag{2.13}$$

number of bits (assuming that A is a power of 2). The code length after coding by applying the same model to the source alphabet is given by:

$$C(A) = -\sum_{i=1}^{m} p_i \cdot \log_2 p_i \tag{2.14}$$

where m is the number of symbols in the alphabet, and p_i is probability of a particular symbol in the alphabet. The main conclusion here is that the code length equals to entropy, which means that the source data can be coded optimally if A is a power of 2.

If the legth of the final interval is not exactly a power of 2, then the final interval can be approximated by any of its subinterval thet meets the requirement $A = 2^{-n}$. Thus the approximation can be bounded by:

$$\frac{A}{2} \leq A' \leq A \tag{2.15}$$

yelding to the upper bound of the code length:

$$C(A) \le -\log_2 \frac{A}{2} = -\log_2 A + 1 = H + 1 \qquad (2.16)$$

The upper bound of the coding deficiency thus is 1 bit for the entire file. The number of bits used by arithmetic coding to encode a symbol with probability *p* is no bigger then entropy plus one.

One of the implementations of arithmetic coder is QM-coder. The main differences between them are as follows [Fränti00]:

- The input alphabet of QM-coder must be in binary form;
- For gaining speed, all multiplications in QM-coder has been eliminated;
- QM-coder includes its own modeling procedures.

Inspite of binary form of input alphabet QM-coder has possibility of having multi-alphabet source. The symbols are coded by one bit at a time, using a binary decision tree. The product of the node decisions probabilities equals to the probability of each symbol.

The multiplication operations are replaced by fast approximations or by shift-left-operations. Let us denote the more probable symbol as MPS, and the less probable symbol as LPS. The interval is divided so that LPS subinterval is above the MPS subinterval. If probability of LPS equals *Qe* and the interval is *A*, then the lengths of LPS and MPS subintervals are *A·Qe* and *A·(1-Qe)* respectively. Such subdivision process is illustrated in Figure 2-4.

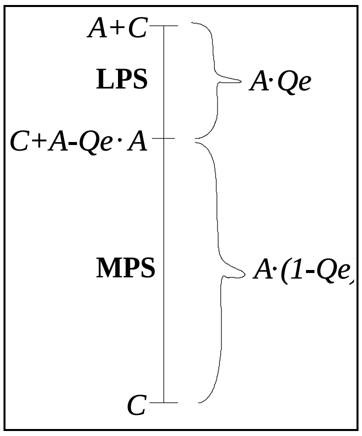


Figure 2.4. Illustration of symbol ordering and ideal interval subdivision.

Instead of operation in the scale $[0,1)$, the QM-coder operates in $[0,1.5)$. Renormalization (or zooming) is processed every time the length of the interval gets below half the scale 0.75. Thus the length of interval always is in the range $0.75 \le A < 1.5$. The row approximation is giben by:

$$A \approx 1 \;\Rightarrow\; A \cdot Qe \approx Qe \qquad (2.17)$$

During coding a symbol the interval is changes as follows:

After MPS:

$$\begin{aligned} &C \text{ is unchanged} \\ &A = A \cdot (1 - Qe) = A - A \cdot Qe \approx A - Qe \end{aligned} \tag{2.18}$$

After LPS:

$$\begin{aligned} &C = C + A \cdot (1 - Qe) = C + A - A \cdot Qe \approx C + A - Qe \\ &A = A \cdot Qe \approx Qe \end{aligned} \tag{2.19}$$

All multiplication operations are eliminated, except those needed for the renormalization. Multiplications in renirmalization can be performed by bit-shifting operations.

QM-coder has its own modeling procedure. The modeling phase determines the context to be used and the binary decision to be coded. QM-coder then picks up the corresponding probability, performs the actual coding and updates the probability distribution if necessary.

3. Lossless compression methods

3.1 JPEG2000 standard

The JPEG2000 standard supports both for lossy and lossless compression. This standard is supported in most applications that process image and video data.

Modified wavelet trellis coded quantization (WTCQ) algorithm is the basis of JPEG2000. The WCTQ algorithm has the following components: the discrete wavelet transform (DWT), trellis coded quantization (TCQ) [MF90] (using step sizes chosen via Langrangian rate allocation), and binary arithmetic bit-plane coding of subbands. The bit-plane coding operates on TCQ indices (trellis quantized wavelet coefficients) in a way that enables successive refinement. The bit-plane coding is processed in order from most to least significant. To exploit dependencies within subbands, spatial context moles are used. In WTCQ algorithm, vector quantization realizes based on finite set of scalar quantizers, where optimal quantizer are found by search of all variants. Such vector quantization has big complexity, therefore uniform scalar quantizer is used in practice with deadzone and trellis coded quantization used optionally [MT02]. Contexts can be chosen within a subband and across subbands. The WTCQ bit-plane coder uses inter-subband context to maximize flexibility in scalable decoding and to facilitate parallel implementation [MF90].

In general encoding process in JPEG2000 standard can be shown as illustrated in Figure 3-1.

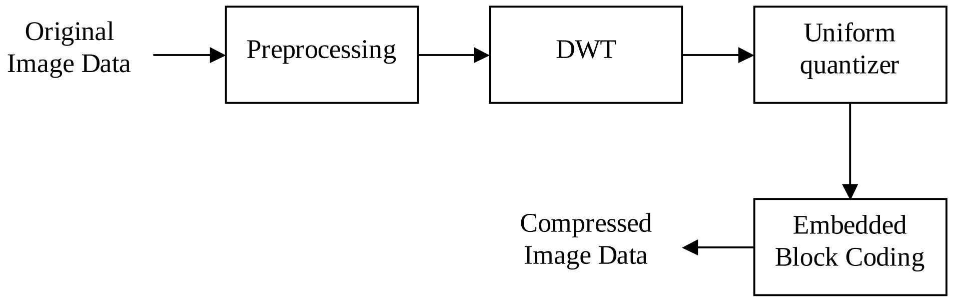


Figure 3-1. The JPEG2000 encoding process

In the preprocessing step, the tiling is processed; level offset and irreversible color transform (ICT). The image to be encoded might be larger than the amount of memory available to the encoder. To solve this problem source image is partitioned into rectangular and non-overlapping tiles of equal size. JPEG2000 expects its input sample data to have a nominal dynamic range centered about zero. This expectation is necessary since current standard uses high-pass filtering. The main motivation for the offset stage is that almost all of the subband samples produced by DWT involve high-pass filtering, and hence, have a symmetric distribution about zero. The level offset stage ensures that this expectation is met. If the original B-bit image sample values are unsigned (non-negative) quantities, an offset of -2^{B-1} is added so that the samples have a signed representation in the following range:

$$-2^{B-1} \leq x[n] < 2^{B-1} \tag{3.1}$$

If the data is already signed (centered about zero), no adjustment is performed.

The color transform is optional. It may be used only when three or more colour components are available and only when the first three components all have identical size and identical bit-depth. The assumption is that the first three components contain the rad, green and blue sample values of a color image. ICT converts RGB data into YC_bC_r format with luminance (or intensity) channel and two colour difference channels. Irreversible color transform is performed by applying the following formula into source image sample values:

$$\begin{bmatrix} Y \\ C_b \\ C_r \end{bmatrix} = \begin{bmatrix} 0.299 & 0.586 & 0.114 \\ -0.169 & -0.331 & 0.500 \\ 0.500 & -0.419 & -0.081 \end{bmatrix} \begin{bmatrix} R \\ G \\ B \end{bmatrix} \tag{3.2}$$

Irreversible colour transform is applied for lossy compression, for lossless mode reversible color transform exists, which is approximation of ICT.

Discrete wavelet transform is one of the main steps in JPEG2000 standard. By applying DWT to each tile image are decomposed into high- and low subbands as illustrated in Figure 3-2.

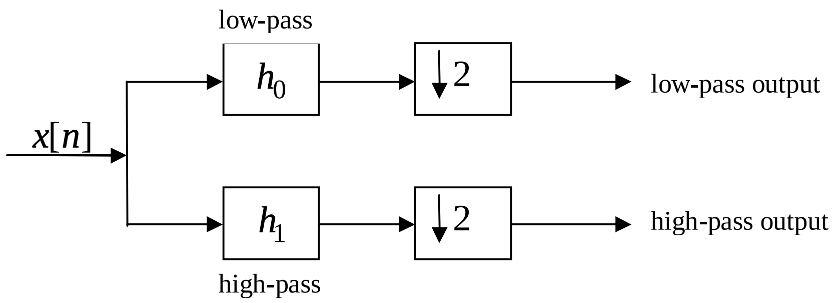


Figure 3-2. The DWT principle scheme.

The DWT is performed by filtering each column and row of the preprocessed image tile with a high- and low-pass filters. Because this process is double the number of samples, the output from each filter is downsampled by 2, so that the sample rate remains a constant. DWT is performed by applying Mallat's pyramidal algorithm [Mallat89]. At first are processed all columns and then all rows from the source image. The DWT process is illustrated in Figure 3-3.

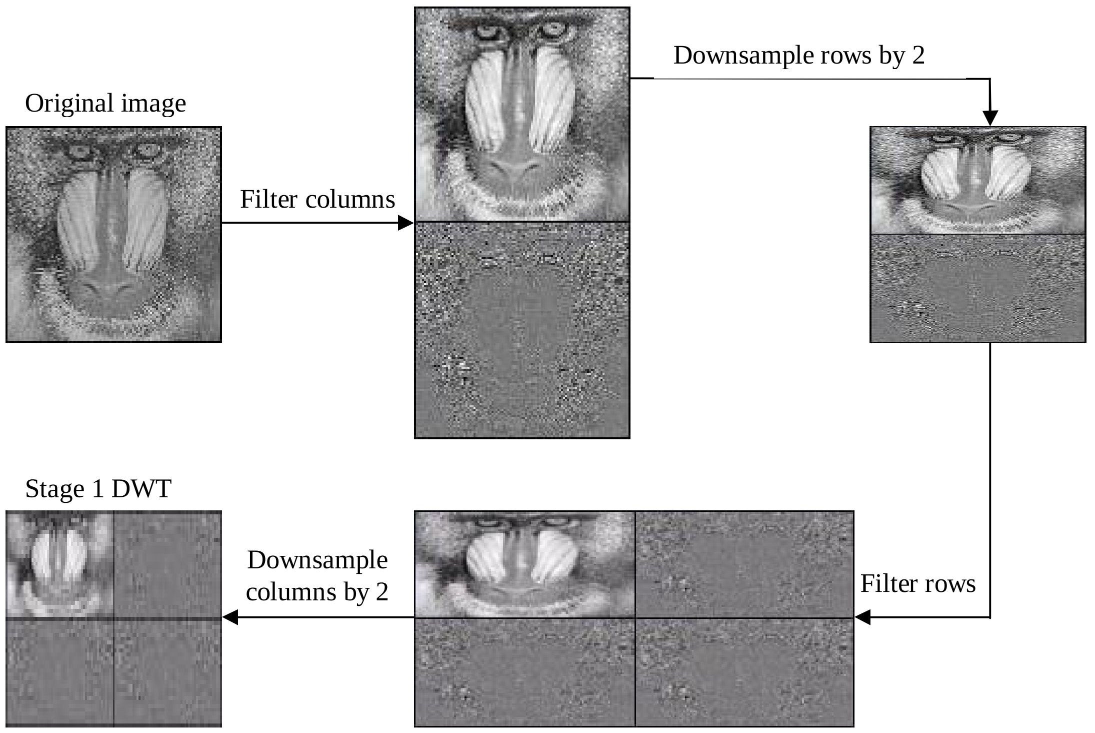


Figure 3-3. DWT process for the Y component.

In practice, JPEG2000 applies DWT at first to columns and then to rows, but it does not matter if rows or columns of the component matrix are filtered first, the result is the same. In Figure 3-3 the Stage 1 DWT for the Y component of the original preprocessed image tile is illustrated. In Stage 2 the same scheme as on Figure 3-3 is applied into upper left subband, which was obtained in Stage 1. This process is iterative and the number of stages is parameter. The JPEG2000 standard supports value of DWT stages D in the range $0 \le D \le 32$. Typical values are in the range $D = 4$ through $D = 8$ with $D = 5$ sufficient to obtain near optimal compression performance for the full resolution image. For reversible DWT is applied “spline 5/3 transform”, since the low- and high-pass analysis filters have 5 and 3 taps respectively [MT02].

After DWT transform quantization is performed. The wavelet coefficients are quantized by using uniform scalar quantizer with deadzone. For each subband b, a basic quantizer step size denote as Δ_b is used to quantize all coefficients in that subband according to:

$$q = sign(y)\left\lfloor \frac{|y|}{\Delta_b} \right\rfloor \tag{3.3}$$

where y is the input to the quantizer (in our case it is value of DWT coefficient), $sign(y)$ denotes the sign of y, Δ_b is size of quantization step and q is the resulting quantizer index. Deadzone means that quantization range about 0 is $2\Delta_b$. Obviously, the quantization step is eliminated for lossless compression.

Embedded block coding is final stage in JPEG2000 compression scheme. The basis of this encoding is context-based adaptive binary arithmetic coder, which is used to compress each bit-plane. Subbands with DWT coefficients are partitioned into small separate code blocks (e.g. 64×64 or 32×32 samples) such that code blocks from a subband have the same size. Each code block is coded separately. During the coding process coefficients into the block are represented as bit-planes. One of these bit-planes is contained from sign of coefficients, others correspond to different digits of coefficient values (position of bit in plane corresponds to position of coefficient in the block). Coding of coefficients means coding of bits, which form these coefficients. So, the arithmetic coding is bit-oriented. Arithmetic coder (adopted QM-coder) here is context based, context is formed as function of bits values, which are surround of coding bit.

Since context-based arithmetic coding is employed, it means that the context selection is necessary. Context-based binary arithmetic coding is a key component in the JPEG2000 image compression standard. The high-compression efficiency of the JPEG2000 is in part due to the careful selection of contexts. In JPEG2000, 18 contexts are specified to code information.

In the context-modeling module, all quantized transform coefficients of the code-blocks are expressed in signmagnitude representation and divided into one sign bitplane and several magnitude bit-planes (from most significant to least significant). During coding scan, the bit-plane can be divided into several stripes. Each stripe is composed of four row samples. The bit-plane is scanned stripe by stripe. In order to improve the embedding of the compressed bit-stream, each bit-plane is coded in three coding passes. Each sample in a bit-plane is coded in only one of the three coding passes. The three coding passes and the condition for each pass are described as follows:

1) Significant pass (SP): The coded sample is insignificant and at least one of the neighbour samples is significant.
2) Magnitude refinement pass (MRP): The relative sample of the previous bit-plane is set significant.
3) Cleanup pass (CP): Those samples that have not been coded by pass SP or pass MRP in current bit-plane.

The Bit-Plane Coding (BPC) works on strips of four elements along the rows. The code block scan is carried from left to right. The BPC requires four-state information bits and 1 magnitude bit for each bit position. The state information bits determine in which pass each bit is coded and are used in the generation of context and data bits. The four-state information bits are as follows:

1. Significance bit (σ) – This is set whenever the magnitude bit of the corresponding subband coefficient is 1 for the first time.
2. Visited once bit (η) – This bit is set when the bit is coded in a pass.
3. Magnitude refinement coded bit (σ') – This bit is set the first time the magnitude refinement primitive is used.
4. Sign bit (χ) – This bit is 0 for positive numbers and 1 for negative numbers and is obtained from sign-magnitude representation of the subband values.

All the state bits except for η bits are maintained across all the bit planes. The η bits are reset at the end of each bit plane. It should be noted that σ and χ for the neighbors that are outside the strip are assumed to be zero. All the three passes make use of one or more of the following four primitives – zero coding (ZC), sign coding (SC), magnitude refinement coding (MRC), and run length coding (RLC). All the primitives use context which is a binary representation of the neighboring pixels. Context for the data in bit position X is formed from the eight neighboring values in the σ matrix as illustrated in Figure 3-4.

D_0	V_0	D_1
H_0	X	H_1
D_3	V_1	D_2

Figure 3-4. Template for context selection.

The 8 neighbors are classified into three groups: horizontal neighbors (H), vertical neighbors (V), and diagonal neighbors (D).

Let us consider primitives more detail:

- ZC – uses nine (contexts 0–8) out of possible 19 contexts. The data is the magnitude of the bit position.
- SC – uses five contexts (contexts 9–13) and is a two-step process. In the first step, the σ and χ of the horizontal and vertical neighbors are used to form the horizontal and vertical *contributions* and a *XOR* bit. In the second step, context is formed from

the two contributions and data is formed by *exclusive OR* operation of the *sign bit* and the *XOR bit.*

- MRC – uses three contexts (contexts 14–16). The contexts are formed based on whether it is the first time the magnitude refinement is being used on a certain position and its eight immediate neighbors. The data is the magnitude bit.
- RLC – uses the remaining two contexts (contexts 17–18). It is invoked only at the beginning of a strip if the σ of all the eight neighbors is 0 for all the bits in a strip. If none of the bits in the strip become significant, context 17 with data = 0 is used. On the other hand, if any bit does become significant, context 17 with data = 1 is used. This is followed by most significant bits and least significant bits of zero index (ZI) of the bit position which contains the 1 bit. Context 18 is used for ZI bits.

As mentioned earlier, each bit plane is coded in three passes. The first bit plane is coded just with the CP. In the SP, all the bits whose $\sigma = 0$ and have at least one of the immediate eight neighbors with $\sigma = 1$ are coded using ZC primitive. If the bit becomes significant, the SC primitive is used σ and of the bit being coded is set to 1. When ZC is applied, the corresponding η is set. In MRC, all the bits with corresponding η =0 and $\sigma = 1$ are coded using MR primitive. The corresponding σ' bit is set to 1. In CP, if η =0 and $\sigma = 0$ for the first element in the strip, the RLC condition is checked. If the RLC condition (mentioned in the RLC primitive) is satisfied, the RLC primitive is used. If one of the bits in the strip becomes significant, then SC is used and σ is set for that bit. This is followed by application of ZC + SC for the rest of the bits in the strip. If the RLC condition is not satisfied, then ZC + SC is used for all the elements with η =0 and $\sigma = 0$.

After context modeling step obtained context data is coded. For encoding the arithmetic coder is used.

Decoding process basically performs the opposite of the encoder, decoding scheme is illustrated in Figure 3-5.

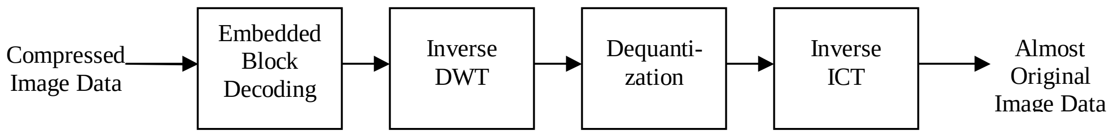


Figure 3-5. The JPEG2000 decoding process.

Representation of information, which was obtained after discrete wavelet transform, very comfortable, because it allows perform zooming copies of the image without full reverse transform. For obtaining of zooming copy of original image it is enough to decode part of information and perform partly reverse transform. So, JPEG2000 standard supports quick and easy zooming. Also important advantage is possibility of access to separate element from the image without full decoding.

3.2 LOCO-I

LOw COmplexity LOssless COmpression for Images (LOCO-I) [WSS96] is the algorithm for lossless and near lossless compression of continuous-tone images. In other words this algorithm is a “low complexity projection” of the universal context modeling paradigm, matching its modeling unit to a simple coding unit based on Golomb codes.

This lossless compression algorithm on a paradigm of “universal modeling and coding” [RL81] is based. It means that compression scheme consists of two distinct and independent components: modeling and coding (see Section 2.3). The main objective in LOCO-I method is to systematically mapping the image modeling principles into a low complexity plane, both from modeling and coding perspective. The key purpose in this process is that separation between modeling and coding becomes less clean under the low complexity coding constraint.

The modeling part can be formulated as an inductive inference problem. During the modeling the image is observed in raster-scan order, it allows accumulate for each instant i past data $x^i = x_1 x_2 ... x_i$, for making inference on the next sample value x_{i+1} based on past data x^i. The modeling stage in LOCO-I is generally broken into following steps [WSS96]:

1. A prediction step, in which a deterministic value $\hat{x}_{i+1}$ is guessed for the next sample value x_{i+1} based on subset of the available past data sequence x^i (a *causal template*).
2. The determination of a *context* in which x_{i+1} occurs (again, a function of a past subsequence).
3. Construction of a probabilistic model for the prediction residual (or error signal) $e_{i+1} = x_{i+1} - \hat{x}_{i+1}$, conditioned on the context of x_{i+1}.

In this scheme, the prediction step is accomplished with an adaptively optimized, context-depend linear predictor, and the statistical modeling is performed with an optimized number of parameters (variable-size quantized context). The modeled prediction residuals in order to attain the ideal code length are encoded [RL81].

c	a	d
b	x	

Figure 3-6. Causal template.

The prediction and modeling units are based on causal template, which is illustrated in Figure 3-6. On this template x is current pixel, a, b, c, d are neighboring pixels in the relative positions shown in the figure. Based on values of a, b, c, d is made guessing the value of pixel x_{i+1}. Ideally, such guessing should be done by adaptively learning a model conditioned on the local edge direction, but in practice for lower complexity another approach for prediction is applied. The solution in LOCO-I consists on performing a primitive test to detect vertical or horizontal edges. If an edge is not detected, then the guessed value for x_{i+1} is $a+b-c$, if the current pixel belongs to the "plane" defined by the three adjacent samples with "heights" a, b and c. This expressed the expected smoothness of the image in the absence of edges. Specifically, predictor guesses [WSS96]:

$$\hat{x}_{i+1} = \begin{cases} \min(a,b) & \text{if } c \geq \max(a,b) \\ \max(a,b) & \text{if } c \leq \min(a,b) \\ a+b-c & \text{otherwise} \end{cases} \tag{3.4}$$

Assuming, without loss of generality, that $a \leq b$, then the predictor of (3.4) can be interpreted as picking a in many cases where a vertical edge exists left of the current location, b in many cases of an horizontal edge above the current location, or a plane predictor $a+b-c$ if no edge has been detected. This predictor do not use value of d, which will be used in context modeling step. Such predictor (3.4) was termed as "median edge detector".

The context, which will use for encoding of the current prediction residual, in LOCO-I is built out on following differences [WSS00]:

$$g_1 = d-b,\ g_2 = b-c,\ g_3 = c-a \tag{3.5}$$

These differences represent *local gradient*. After calculation these differences are quantized into a small number of approximately equiprobable connected regions by applying the quantizer $k(\cdot)$. The quantization aims at maximizing the mutual information between the current sample and its context. To keep symmetry, the regions are indexed as $-4, \ldots, -1, 0, 1, \ldots, 4$ with $k(g) = -k(-g)$ for a total of 729 different quantized context triplets. For prediction residual e_{i+1}, if the first non-zero element of a triplet is $C_i = [q_1, q_2, q_3]$, where $q_j = k(g_j),\ j = 1, 2, 3$, is negative, the encoded value is $-e_{i+1}$ by using context $-C_i$. Merging contexts of "opposite signs" give in a total of 365 different contexts. For 8-bit per pixel alphabet (which correspond to 8-bits grayscale images), for differences $g_j,\ j=1,2,3$ the default quantization regions are $\{0\}$, $\pm\{1,2\}$, $\pm\{3,4,5,6\}$, $\pm\{7,8,\ldots,20\}$, $\pm\{e \mid e\geq 21\}$, and their corresponding negative counterparts. However, the value of boundaries is adjustable parameter, except the central region must be $\{0\}$.

Coding of prediction residual is based on Golomb codes [Golomb66], whose structure enables simple calculation of the code words without recourse to the storage of code table. Specifically, prediction residuals are encoded by using following codes:

$$C=\{G_{2^k}(M(\cdot)) \mid k\geq 0\}\mathbf{U}\{G_1(M'(\cdot))\}, \tag{3.6}$$

where G_m denotes the Golomb code of order m, $M(x)$ denotes the mapping from an integer x to its index in the interleaved sequence $0,-1,+1,-2,+2,\ldots$ (starting form index zero), and $M'(x)=M(-x-1)$. The map M is often called as Rice mapping. The code parameter k is computed by the C programming language “one-liner”:

```
for (k = 0; (N<<k)<A; k++);                    (3.7)
```

where N counts quantity of prediction residuals that have been coded at that context, and A collects the magnitudes of the prediction residuals for that context. So, the adaptive symbol-by-symbol coding has low complexity, than more complex arithmetic coders.

So, LOCO-I characterizes low complexity of compression algorithm, it is main advantage of this method. Also to be interested of unique predictor (3.4) in this standard, it is best possible predictor in condition of low complexity.

3.3 CALIC

The CALIC standard is context-based, adaptive, lossless image codec, which is based on paradigm of “universal modeling and coding” [RL81]. This codec is characterized higher lossless compression of continuous-tone images with relatively low time and space complexity of compression algorithm. During development of CALIC standard new efficient algorithmic techniques have been processed for context formation, quantization and modeling.

CALIC is sequential coding scheme, which encode and decode source image in raster scan order with a single pass through the image. For coding process only two previous scanned lines in prediction and context modeling steps are used.

First step in CALIC scheme is prediction; this compression algorithm has GAP-Gradient-Adjusted Prediction that utilizes priori knowledge of image smoothness. The GAP is simple, adaptive, nonlinear predictor, which can adapt itself to the intensity gradients near the predicted pixel; it weights the neighboring pixels of current sample according to the estimated gradients of the image [WM97a]. Let us denote value of current pixel as $I[i, j]$. For prediction and modeling causal template illustrated in Figure 3-7 is used.

		nn	nne
	nw	n	ne
ww	w	**?**	

Figure 3-7. Causal template for adjacent pixels in prediction and modeling.

Let us denote adjacent samples as follows:

$$\begin{aligned} &I_n = I[i,\ j-1], I_w = I[i-1,\ j], I_{ne} = I[i+1,\ j-1] \\ &I_{nw} = I[i-1,\ j-1], I_{nn} = I[i,\ j-2], I_{ww} = I[i-2,\ j] \\ &I_{nne} = I[i+1,\ j-2] \end{aligned} \tag{3.8}$$

Formulas (3.8) mean north, west, northeast, northwest, north-north, west-west and north-northeast respectively. The locations of these pixels are illustrated in Figure 3-7.

The gradient of the intensity function is estimated by following quantities:

$$\begin{aligned} d_h &= |I_w - I_{ww}| + |I_n - I_{nw}| + |I_n - I_{ne}| \\ d_v &= |I_w - I_{nw}| + |I_n - I_{nn}| + |I_{ne} - I_{nne}| \end{aligned} \tag{3.9}$$

Clearly, d_h and d_v are estimates within a scaling factor of the gradients of the intensity function near current pixel $I[i, j]$ in the horizontal and vertical directions. Values of d_h and d_v for detecting magnitude and orientation of edges in the input image are used. In formulas (3.9) the absolute values are used, the reason for using absolute differences is to prevent cancellation of values of opposite signs. Value of d_h means value of horizontal gradient, d_v means value of vertical gradient. GAP predictor uses values of gradients by following principle. If value of vertical gradient d_v bigger than value of horizontal gradient d_h on some threshold value (typical threshold value is 80), then in current part of image exists clearly marked horizontal edge, therefore predictor value $\hat{I}[i, j]$ for current pixel equals value of left pixel $I_w = I[i-1,\ j]$. Similarly, if value of horizontal gradient bigger than value of vertical gradient on 80, then prediction value $\hat{I}[i, j]$ equals value of upper pixel $I_n = I[i, j-1]$. Otherwise, the prediction value is obtained by following linear predictor:

$$\hat{I}[i,\ j] = (I_n + I_w)/2 + (I_{ne} - I_{nw})/4 \tag{3.10}$$

In CALIC contexts for error modeling are formed by embedding 144 texture contexts into four energy contexts to form a total of 576 compound contexts. Texture contexts are formed by quantization of a local neighborhood of pixel values to a binary vector:

$$\begin{aligned} C = \{x_0, x_1, x_2, x_3, x_4, x_5, x_6, x_7\} = \\ \{I_n, I_w, I_{nw}, I_{ne}, I_{nn}, I_{ww}, 2I_n - I_{nn}, 2I_w - I_{ww}\} \end{aligned} \tag{3.11}$$

Vector C is then quantized to an 8-bit binary number $B = b_7 b_6 ... b_0$ by using the prediction value $\hat{I}[i, j]$ as the threshold:

$$b_k = \begin{cases} 0, if\ x_k \geq \hat{I}[i, j] \\ 1, if\ x_k < \hat{I}[i, j] \end{cases} \quad 0 \leq k < K = 8 \tag{3.12}$$

Clearly, B captures the texture pattern in the modeling context are indicated of the prediction error behavior:

$$e = I[i, j] - \hat{I}[i, j] \tag{3.13}$$

Also note that an event x_i need not be a neighboring pixel to $I[i, j]$ in current context. It can be a function of some adjacent pixels. x_6 and x_7 represent the events whether the prediction value $\hat{I}[i, j]$ forms concave waveform with respect to neighboring pixels in vertical and horizontal directions.

The texture contexts are combined with quantized error energy to form compound modeling context. Error energy contexts are obtained by using following error energy estimator:

$$\Delta = d_h + d_v + 2|e_w| \tag{3.14}$$

where $e_w = I[i-1, j] - \hat{I}[i-1, j]$, d_h and d_v are horizontal and vertical gradients respectively. Value of e_w is chosen in absolute because large errors tend to occur consecutively. Estimator Δ is quantized to four levels yielding a quantized error energy context $Q(\Delta)$ which is combined with the quantized texture pattern $0 \leq B < 2^K$ to form compound modeling context, denoted as $C(d, b)$. At a glance we have $4 \times 2^8 = 1024$ different compound contexts. However, not all 2^8 binary codeword of the B quantizer defined by (3.12) are possible, available only 144, so the total number of valid compound contexts is only $4 \times 144 = 576$.

Coding step in CALIC is begun by coding of error energy estimator Δ as defined in (3.14). Conditioning the error distribution on Δ leads to separation of prediction errors into classes of different variances [WM97a]. Thus entropy coding of errors using estimated conditional probability $p(e \mid \Delta)$ improves coding efficiency over using $p(e)$. Also Δ has to be quantized to a small number of L levels for time and space efficiency. In practice, $L = 8$ is sufficient. Larger L improves coding efficiency marginally. Estimating $L = 8$ conditional error probabilities $p(e \mid Q(\Delta))$ requires only a modest amount of memory while estimating probabilities for entropy coding.

CALIC uses an adaptive m-ary arithmetic coder. However, CALIC does not apply an m-ary arithmetic coder to prediction errors directly. Instead it first remaps prediction errors into alphabet of size 2^z instead of 2^{z+1} for a z-bit image [WM97a]. Prediction errors can potentially take on 2^{z+1} possible values from range $[-2^z + 1, 2^z - 1]$, they can be mapped into the range $[0, 2^z - 1]$. Also, the tails of error distributions are truncated and an escape mechanism to further reduce the number of code symbols is used.

So, CALIC is very efficient lossless compression algorithm of continuous-tone images with relatively low time and space complexity. Here entropy coding step is independent of source modeling, and it is required to all lossless image compression algorithms.

4. Context modeling

4.1 Fundamentals

For compression algorithms based on paradigm of “universal modeling and coding” the first step is modeling (see Section 2.3). Context modeling is one of the varieties of modeling step. At present time the compression methods based on context modeling are known, for instance, CALIC, LOCO-I (see Section 3).

The problem is probability estimation of symbols’ occurrence in each position of source data stream. For lossless compression case we can use only the information that is known both of encoder and decoder. It means that probability estimation of occurrence symbol depends on properties of data block previously processed.

Let us denote the context modeling as an estimation of probability of occurrence symbol (element, pixel, sample or set of different objects) depending on previous symbols, or a context. In practice, term “context” is used as collection of neighboring symbols, which are surrounding current symbol. It is a context in the broad sense. Left-side and right-side contexts exist in practice, i.e. as left-side context is considered the set of adjoining symbols from left side, for right-side context respectively.

If the length of the context is finite, then context modeling is denoted as *finite-context modeling*. Finite context modeling is based on the fact that a symbol that has appeared recently will appear in a near future, so the more times that it appears the higher is the probability that it appears again, so every time it has been seen, we increment its probability. Let us denote the maximum length of usable context as *order* of context *N*. For example, during context modeling with order 3 for last symbol in sequence “…milk…” context with maximum length is “mil”; as contexts here are following strings: “mil”, “il”, “i” and empty string. All of these contexts with length from *N* to 0 denote as active contexts, i.e. for estimation of symbol can be used cumulative statistics about these contexts. The entropy of an *N*-order context model is given by:

$$H_N = -\sum_{j=1}^{N} p(c_j)\cdot\left[\sum_{i=1}^{k} p(x_i|c_j)\cdot\log_2\left(p(x_i|c_j)\right)\right] \qquad (4.1)$$

where $p(x\,|\,c)$ is the probability of symbol *x* in a context c, and *N* is the number of different contexts.

In general case, the context model for each context with finite length happened in source data stream is generated. Any context model contains counter mechanism of all symbols, which are happened in corresponding context to this model. The value of counter of symbol x in current context increases after each happened of symbol x in this context.

The main question of context modeling is selection of optimal context for more precise estimation. And aim of context modeling is to find such context.

Context model has an order, as was described above. Once we have different orders the question of how to use them arises. *Blending* is one of the solutions, such method means combining the probability of all the orders in a single probability for every symbol from alphabet [BCW89]. Such combining is done by adding together the probability of the same symbol under different contexts. However, in practice probabilities of higher orders tend to be more reliable and thus achieve higher compression, the way we reflect this in blending is by weighting higher orders that is multiplying them to give them more importance when they are added together. Every context has a different weight which we can choose beforehand, or change while compressing.

Let us consider context model with order N. Let us assume $p(x_i \mid o)$ is probability of symbol x_i from alphabet A in the finite context model with order o. This probability is adaptively and will change during compression process. The blended probabilities are given by following formula:

$$p(x_i) = \sum_{o=-1}^{N} w(o)\, p(x_i \mid o) \tag{4.2}$$

where $w(o)$ is weight of model with order o and

$$\sum_{o=-1}^{N} w(o) = 1 \tag{4.3}$$

As order of context model here is considered length of corresponding context. In formulas (4.2) and (4.3) the order of context model -1 is used, context model with order -1 assumes equal probability for all alphabet symbols. So, for context models with order 0 and 1 we have the same probability distribution of alphabet symbols.

Probability estimation $p(x_i \mid o)$ is defined by following formula:

$$p(x_i \mid o) = \frac{f(x_i \mid o)}{f(o)} \tag{4.4}$$

where $f(x_i \mid o)$ is number of appearance of symbol x_i in current context with order o, $f(o)$ is overall number of context appearance in the processed data stream. For $f(o)$ exists following equation:

$$f(o) = \sum_{x_i \in A} f(x_i \mid o) \tag{4.5}$$

So, the simple blending can be constructed from the mechanism of choosing the probability estimation of the context by applying formula (4.4).

If weight $w(-1) > 0$ it is guarantee successful compression for any symbol of source data stream, because existence of context model with order $o = -1$ allows to obtain nonzero probability estimation and code with final length.

Recognize *fully blended* context modeling, when prediction is defined by statistics of all usable orders in context model, and *partially blended* otherwise. However using blending is slow, though it gives good compression.

In practice for image compression the context is usually the value of adjacent samples, such neighboring pixels form *context template*. The main requirement is accessibility of pixels from context template both the encoder and decoder. In Figure 4-1 typical context template for image compression is illustrated, when as context is took pixels from the west, north, northwest and northeast locations of the current pixel.

NW	N	NE
W	●	

Figure 4-1. Example of context template with size four.

In such example (with context template in Figure 4-1) the number of contexts equals number of all possible combinations with adjacent four pixels and depend on number of colors on each pixel. If we have grayscale image with 8 bits per pixel and 256 colors, for one pixel in template the number of contexts is $2^8 = 256$, for two pixels in context this value is $2^{2\times 8} = 65536$. So, number of contexts increases exponentially with the number of pixels in context template. A solution of this problem is quantization of samples values and reducing by such method possible combinations.

4.2 Context Tree

4.2.1 Structure of context tree

For storing statistical information during context modeling it is convenient to use special tree structure. We refer to this structure as *context tree*. Using of such structure allows using larger number of neighboring pixels in context template without the context dilution problem, when the statistics are distributed over too many contexts, thus affecting the accuracy of the probability estimations. Context tree provides flexible approach for modeling the contexts with larger number of adjacent pixels on template.

The contexts are represented as a tree, in which the context is constructed pixel by pixel. In Figure 4-2 the example of context tree is illustrated. Each node represents single context with its statistical information.

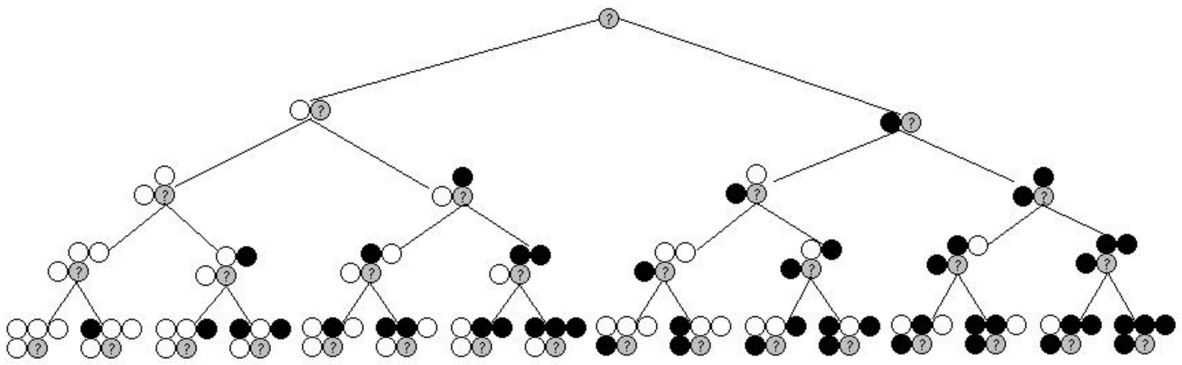


Figure 4-2. An example of context tree with depth of 4.

The two children of a context correspond to the parent context increased by one more pixel. The position of this pixel is fixed in predefined order on context template. Context template here can be different, for example as illustrated in Figure 4-3, where pixel with label “?” means coding pixel and others positions are context pixels.

4	2	3
1	**?**	

Figure 4-3. Context template with size 4.

If context tree is constructed based on template from Figure 4-3, it has structure as shown on Figure 4-2. Template defines order for adjacent pixels during context modeling; it means that first context is pixel from west, after that context is increased by one pixel from north location and so on based on template’s order.

Context tree is constructed based on following notations [Kopyl04]:

- The information about context is stored in the leaves.
- Every tree node has many branches as there are colors in the image in that particular context.

- The children of a node correspond to their parent by adding one more pixel at the position defined by context template (see Figure 4-3).
- The context selection is made by traversing the context tree from root to leaf, each time selecting the branch according to the value of pixel in the corresponding position within context template (see Figure 4-2).

4.2.2 Static and semi-adaptive approaches

Context tree can be generated based on two alternative approaches. In the semi-adaptive approach the context tree is optimized directly for the source image. For such constructing additional pass is required over the image.

Another approach of context tree construction was proposed in [FA99] uses static tree, which is optimized on a training image. It is possible, because trees of similar type images (binary images) are similar. The main problem of static approach is control the growth of the tree.

The difference between these methods is that first semi-adaptive model exactly conforms to source image, but it attains more calculation time; second static model does not require any additional calculation, but on non-binary images (grayscale or color images) it has not good compression results, because similarity of non-binary images is worse than on binary images.

4.2.3 Construction of an initial context tree

Constructing of initial context tree begins by processing through the image data to collecting statistics for all potential contexts, leaves and internal nodes. Let us denote a is number of colors in source image and the m neighboring pixels $x_1,...,x_m$ as x^m .Each node of the context tree stores information of a counts for all symbols, generated at the current context. The algorithm of the context tree construction is as follows [AKF05]:

Step 1: Create a root of the tree.

Step2: For all i = 1 to n (n is the number of pixels in the image), traverse the tree along the path defined by the template pixels x^{i-1}, where the positions of pixels are defined according to predefined scan order. If some positions of the pixels in x^{i-1} are outside of the image, then set these pixels to zero. If some node, visited according to the correspondent symbol of x^{i-1}, does not have a consequent branch (for transition to the symbol x^{i-1}), then create the necessary child node and process it. Each new node has a counters, which are initially set to zero. In all visited nodes, increase the count of x_i by 1.

This completes the construction of the context tree for all possible contexts. The time complexity of this algorithm is $O(n)$.

4.2.4 Pruning of context tree

Pruning is very important step in context tree construction. The context tree is optimized according to encoded data. One of possible approaches for context tree pruning is represented here. The main goal in pruning process is excluding of contexts with small contribution and gets context tree with optimal structure.

The pruning is produced by comparing of the parent node and its children nodes for finding the optimal combination of siblings. Let us denote overall tree structure as T and by w nodes of current context tree T, i.e. $w \in T$. As $c(w)$ the number of bits is denoted, which are needed for storing the node w in the compressed file. $c(w)$ is given by:

$$c(w) = \begin{cases} 1, \text{if } w \text{ is a leaf} \\ a+1, \text{otherwise} \end{cases} \tag{4.6}$$

where a equals number of colors in the source file. Denote as $S(T)$ the set of all terminal nodes of the context tree T, $n_i(s)$ is count of the symbol i, encoded by the statistical model, pointed by the terminal node $s \in S(T)$. By the estimated codelength, generated by a terminal node s here we understand the value of the following expression [WR95, MF98]:

$$\tilde{c}(n_1(s), n_2(s), \ldots n_a(s)) = \begin{cases} 0, & \text{if } n_1(s) = n_1(s) = \ldots = n_a(s) = 0 \\ -\log_2 \dfrac{\prod_{i=1}^{a} \prod_{j=0}^{n_i(s)-1} (j+e)}{\prod_{j=0}^{n_0(s)+n_1(s)+\ldots+n_a(s)-1} (j + a \cdot e)}, & \text{otherwise.} \end{cases} \tag{4.7}$$

The parameter e here depends on different modeling schemes of coding [HV91]. At the start of encoding we consider all elements as equiprobable and set e equal to $1/a$. This definition (4.7) corresponds algorithmically to one pass arithmetic coder. Cost of the context tree T has following form [AKF05]:

$$L(T) = \sum_{w \in T} c(w) + \sum_{s \in S(T)} \tilde{c}(n_1(s), n_2(s), \ldots, n_a(s)) \tag{4.8}$$

Cost of context tree is sum between description of context tree and estimated code length.

The main aim of pruning process is making such modification in the of context tree, that the cost function (4.8) will be minimized. This problem can be solved by applying bottom-up algorithm [Norhe94], where the main principle is that the optimal tree consists of optimal subtrees.

Let us denote vector of counts for any node $w \in T$ as $\tilde{n}(w) = (n_1(w), n_2(w), ..., n_a(w))$, the child nodes as w_i, and the node configuration vector as $v = (v_1, ..., v_a)$, $v_i \in \{0,1\}$. The node configuration vector has such structure, that define of which node branches are considered to be remained: if $v_i = 0$, it means that i^{th} branch will be deleted from the node, otherwise branch does not change. The subtree $\hat{T}$, which starts from the given node w as from root, has following principle: the optimal cost $L_{opt}(\hat{T})$ for any given subtree $\hat{T} \subseteq T$ can be expressed by the following recursive equations [AKF05]:

$$L_{opt}(\hat{T}) = \begin{cases} 0, & \text{if } \hat{T} \text{ is null} \\ \tilde{c}(\tilde{n}(w)) + 1, & \text{if } \hat{T} \text{ has no subtrees} \\ \min\limits_{v} \{L_v(\hat{T}, v)\}, & \text{otherwise,} \end{cases} \tag{4.9}$$

where

$$L_v(\hat{T}, v) = \tilde{c}\left(\tilde{n}(w) - v \circ \left(\sum_i \tilde{n}(w_i) \right) \right) + \sum_i \left(v_i \cdot L_{opt}(\hat{T}_i) \right) + a + 1 \tag{4.10}$$

The operation '$\circ$' here is the *Hadamard product* (it other words it is the element by element product of two vectors/matrices). The tree $\hat{T}_i \subset \hat{T}$ is a subtree of $\hat{T}$, root of $\hat{T}_i$ is its child node w_i. So, the pruning process has following steps [AKF05]:

Step 1: If T has no child nodes, then return the accumulated codelength of its root by using formula (4.9).

Step 2: Calculate optimal costs $L_{opt}(T_i)$ for all subtrees $T_i \subset T$, which are started from the child nodes of *T* root.

Step 3: Find the optimal configuration vector $\tilde{v} = \arg\min\limits_{v} L_v(T, v)$ according to $L_{opt}(T_i)$, the vectors of counts $\tilde{n}(t)$, and $\tilde{n}(t_1), ..., \tilde{n}(t_N)$.

Step 4: Prune out the children subtrees according to the vector $\tilde{v}$.

Step 5: Return the value of cost function $L_v(T, \tilde{v})$.

In Step 3 of pruning process the optimal configuration vector $\tilde{v} = \arg\min\limits_{v} L_v(T, v)$ is constructed. It is found by using steepest descent optimization algorithm. According to formulas (4.9) and (4.10) the optimization problem for an intermediate node $\hat{T}$ has following form [AKF05]:

$$\tilde{v} = \arg\min_{v\in C^a}\left\{\tilde{c}\left(\tilde{n}(w) - v\,\mathbf{o}\left(\sum_i \tilde{n}(w_i)\right)\right) + \sum_i \left(v_i \cdot L_{opt}\left(\hat{T}_i\right)\right) + a + 1\right\} \qquad (4.11)$$

Steepest descent algorithm for finding of optimal configuration vector has following steps [AKF05]:

Step 1: In this step the direction and starting point of the search are found. Values of the optimized function are calculate according to (4.11): $L_0 = L_v(\hat{T}, v)\big|_{v=(0,0,..0)}$ and $L_1 = L_v(\hat{T}, v)\big|_{v=(1,1,..1)}$. If the $L_0 < L_1$ than the search direction Δ is $+1$ and starting point is $\bar{v} = (0, 0, \ldots, 0)$, otherwise the direction is -1 and the starting point is $\bar{v} = (1, 1, \ldots, 1)$. The left bound ($LB$) of search is set of 1.

Step 2: For the current starting point $\bar{v}$, starting value $L_v(\hat{T}, \bar{v})$ and current left bound of search are obtained all neighboring points $\bar{v}^k$, $k \in [LB, \ldots, a]$, where $\bar{v}^k = \{\bar{v}_1, \ldots, \bar{v}_k + \Delta, \ldots, \bar{v}_N\}$, $\bar{v}_k + \Delta \in \{0,1\}$, and all values $L_v(\hat{T}, \bar{v}^k)$. If there is no such neighboring points than return as the minimum search result the starting point $\bar{v}$ and starting value $L_v(\hat{T}, \bar{v})$.

Step 3: Minimal cost function is calculated as $\tilde{L}_{\min} = \min_k \left\{L_k\left(\hat{T}, \bar{v}^k\right)\right\}$. If the found value $\tilde{L}_{\min}$ is bigger or equal than the starting value, then the starting point $\bar{v}$ and starting value $L_v(\hat{T}, \bar{v})$ are returned as the result. Otherwise all $\tilde{k} : L_v\left(\hat{T}, \bar{v}^{\tilde{k}}\right) - \bar{L}_{\min} \le l$ are calculated (here value l = 0.01, this number was found experimentally). The optimization algorithm is called for each $\tilde{k}$ recursively, i.e. the new starting point is $\bar{v}^{\tilde{k}}$, starting value is $\bar{L}_{\min}$, the left bound of search is $\tilde{k}$ and go to Step 2. From all resulting points of optimization we find those one, which gives us the minimum value of the cost function $L_v(\hat{T}, \bar{v})$. This point is the result of optimization and the estimated codelength, achieved at this point is the resulting value of optimum search process.

The number of calculations of this quasi-optimal algorithm in the worst case is 2^a, which is the same as in the case of the full-search approach, when all possible variants of subtrees configurations are calculated and the best one is chosen. But in practice the quasi-optimal approach gives us strong reducing of the tree construction time.

5. General Context Tree based on Intensity

The lossless compression algorithm will consider in this section. Main feature of this compression algorithm is using of General n-ary Context Tree (GCT) with incomplete and optimal structure [AKF05]. Here GCT means the same as context tree, which was described in Section 4.2 above, but we denote it as "general", because it is non-binary tree with number of colors more than two. Current algorithm contains effective construction of an optimal incomplete n-ary context tree with taking into account memory and time requirements.

This algorithm has orientation to raster map images. Such class of images is characterized by a small number of colors, a lot of structured details and big size of images. The linear-predictive coding technique, as used at LOCO-I and CALIC (see Section 3.2 and Section 3.3), works well on images with smooth changes of colors but if on image exists the sharp changing of colors these methods work worse than considered algorithm.

5.1 Context template

The context modeling is based on pixels from specified region; such region is denoted as context template, which also defines order in the context. In our case, during context modeling the source image is observed in raster scan order. Let us consider two dimensional template [AKF05], which is illustrated in Figure 5-1.

			28	24	27			
	30	22	18	14	17	21	29	
	20	12	10	6	9	11	19	
26	16	8	4	2	3	7	15	25
23	13	5	1	***U***				

Figure 5-1. Default location and order of the neighboring pixels with maximum depth 30.

Current context template is based on adjacent pixels, maximum value of neighboring samples is 30. Obviously, big size of template increases the size of context tree and it takes more computer memory, therefore in practice more convenient value is less then 30. In Figure 5-1 special template is illustrated, where current pixel is denoted as ***U***, it has the *m* neighboring pixels $x_1,...,x_m$ as x^m, where *m* denotes the values of adjacent pixels in context.

5.2 Compression scheme

The compression scheme consists of two main steps: context modeling and coding. First step corresponds to statistical modeling. During this step, GCT is constructed and pruned. In the second step, the arithmetic coding or in other words entropy-based coding is produced. The scheme of the proposed compression algorithm is illustrated in Figure 5-2.

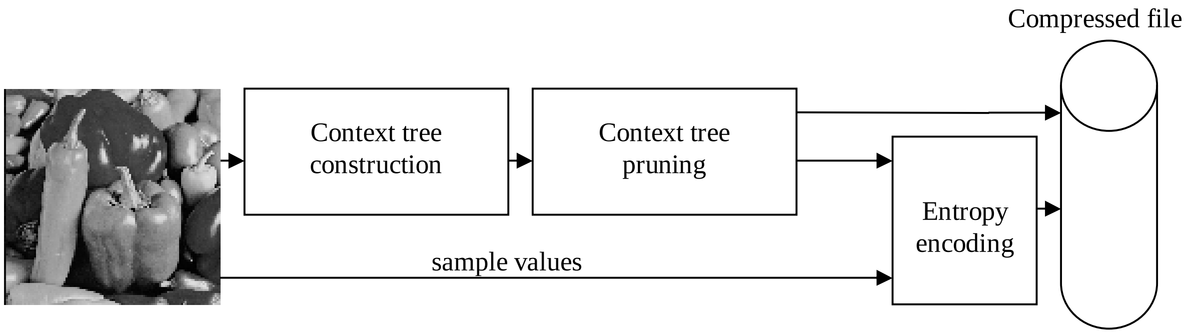


Figure 5-2: The principle scheme of the GCT-I algorithm.

Let us denote the current compression algorithm as General Context Tree based on Intensity (GCT-I), because it produce context modeling with using this GCT structure based on special template used intensity (or value) of adjacent pixels.

Let us consider more detail this compression scheme shown in Figure 5-2. The first step here is the construction of GCT with incomplete and optimal structure. This step consists of the following operations:

1. Construction of context model using GCT;
2. Pruning of GCT for obtaining the optimal structure of GCT.

In Section 4.2.3, this is described as construction of initial context tree. Here we use semi-adaptive approach for the context tree construction. This algorithm is applicable also for context tree, when number of colors a is greater than two. As context template the region defined as in Figure 5-1 is used (see Section 5.1). The probability of the current pixel $\boldsymbol{U}$ is conditioned on its context x^m. The probabilities of different contexts, as well as the probabilities of the pixels generated in a given context, are usually treated as being independent.

For constructing the context model we have to assign probabilities to each new symbol generated at the current context $n_i(x^m)$. So, probability of a new symbol in current context is given by:

$$n_k(x^m) = n(U = k | x^m) \tag{5.1}$$

The conditional probability of the event $U = k,\ k \in [1,..,a]$ will be as following equation:

$$p(U = k | x^m) = \frac{n_k(x^m)}{\sum_{j=1}^{a} n_j(x^m)} \qquad (5.2)$$

After construction, the GCT is pruned by the algorithm described in Section 4.2.4. In the second step, the encoding of the given statistical model is performed using GCT with optimal structure by entropy-based encoder (QM-coder). The probability estimator of entropy-based coder operates by the following formula:

$$p(x_i = k | x^m) = \frac{n_k(x^m) + e}{\sum_{j=1}^{N} n_j(x^m) + a \cdot e} \qquad (5.3)$$

The parameter *e* here depends on different modeling schemes of coding [HV91]. At the start of encoding, we consider all elements as equiprobable and set *e* equal to $1/a$.

6. General Context Tree based on DWT coefficients

The lossless compression algorithm used context modeling based on Discrete Wavelet Transform (DWT) coefficients will reperesent in this section. The main distinction from GCT-I method (see Section 5) is a preprocessing step before context modeling. During preprocessing step the DWT is performed on source image, after that context modeling is processed on DWT coefficients. This operation is motivated of using advantage of the correlation between neighboring coefficients during the context modeling. In such notation, DWT coefficients are used instead pixels of source image.

The idea of applying DWT as preprocessing was produced from JPEG2000 standard, where DWT is one of the steps (see Section 3.1). For current algorithm the GCT structure (see Section 4.2) is used for effective storing and processing the information. Also the important feature is context templates: two different templates based on DWT coefficients for context modeling are used in this algorithm.

6.1 Discrete Wavelet Transform

The DWT [SN96] is generally understood as a dyadic tree-structured subband transform with the multi-resolution structure identified as on Figure 6-1. Compression schemes based on such structure are usually known as wavelet-based schemes.

The importance of wavelet transform derives from an interest in the regularity of the waveforms which are represented by discrete sequences of samples. The DWT is used effectively in known compression standard JPEG2000. It is important step in this compression algorithm (see Section 3.1).

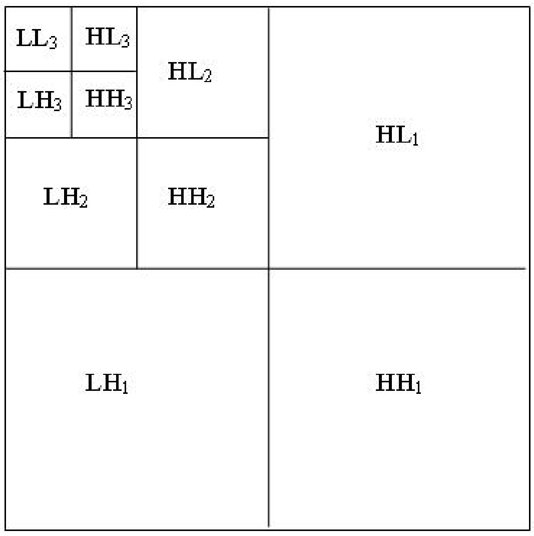


Figure 6-1. Passband structure for a two dimensional tree-structure subband transform with *D*=3 levels.

Let us consider discrete signal as $x[k]$. Also confine attention to the Hilbert space of square-integrable functions on the real-line, L^2(R). Inner products are defined by $\langle x, y\rangle = \int_{-\infty}^{\infty} x(t)y^*(t)dt$, in such notation wavelet transform is concerned with functions $x(t)$ for regularity.

Basis for L^2(R) is wavelet basis, i.e. family of functions, $y_n^{(m)}(t)$, all derived by translation $y(t)=y(t+1)$ and dilation $y(t)=y(mt)$ (expansion) of a single “mother wavelet” $y(t)$, according to

$$y_n^{(m)}(t)=\sqrt{2^{-m}}y(2^{-m}t-n) \tag{6.1}$$

$y_n^{(m)}(t)$ are linearly independent and span L^2(R). That is any signal can be written as following form:

$$x=\sum_{m=-\infty}^{\infty}\sum_{n=-\infty}^{\infty} y_1^{(m)}[n]y_n^{(m)} \tag{6.2}$$

where $y_1^{(m)}[n]$ is a sequence of real numbers, which contain information about signal.

Increasing value of m corresponds to increasing the scale (dilation) of the wavelet functions $y_n^{(m)}(t)$. The factor $\sqrt{2^{-m}}$ from equation (6.1) ensures that the wavelet basis signals all have identical norm or energy, i.e. $\|y_n^{(m)}\|=\|y\|, \forall n, m$. This is important if $\{y_n^{(m)}\}_{m,n\in Z}$ is to form an orthonormal basis for L^2(R), although wavelet basis need not necessary be orthonormal.

Let us consider following definitions in terms of “multi-resolution” analysis. Multi-resolution analysis is defined on L^2(R) as a set of sub-spaces: $\ldots \subset V^{(2)} \subset V^{(1)} \subset V^{(0)} \subset V^{(-1)} \subset V^{(-2)} \subset \ldots$, which are satisfying following properties:

1. $\mathbf{U}_{m\in Z} V^{(m)} = L^2(R)$.

2. $\mathbf{I}_{m\in Z} V^{(m)} = \{0\}$. It means that every $x \in L^2(R)$ has non-zero resolution so that its projections, $x^{(m)}$, converge to 0, as $m \to \infty$, where convergence belongs to $L^2(R)$.

3. $x(t) \in V^{(0)} \Leftrightarrow x(2^{-m}t) \in V^{(m)}$

4. $x(t) \in V^{(0)} \Leftrightarrow x(t-n) \in V^{(0)}$

5. Exists orthonormal basis $\{j_n\}_{n\in Z}$, for $V^{(0)}$ such that $j_n(t) = j(t-n)$. The function $j_n(t)$ is called the “scaling function”.

Parameter *m* means here “scale parameter”; since it decreases as the scale of signal features decreases (resolution increases). The important property of basis functions is that the basis functions for $V^{(0)}$ are integer translates of a single scaling function. Third and fifth properties, together indicate that each resolution space $V^{(m)}$ has an orthonormal basis $\{j_n^{(m)}\}_{n\in Z}$, where $j_n^{(m)}(t) = j^{(m)}(t - 2^m n) = \sqrt{2^{-m}}\, j(2^{-m}t - n)$.

Let us denote $W^{(m)}$ the orthogonal complement of $V^{(m)}$ in $V^{(m-1)}$, so that $W^{(m)} \perp V^{(m)}$ and $W^{(m)} \oplus V^{(m)} = V^{(m-1)}$. The aim is to find orthonormal basis, $\{y_n^{(m)}\}_{n\in Z}$, for each $W^{(m)}$ where basis functions, $y_n^{(m)}$ are translated and dilated versions of a single mother wavelet, y. First and second properties ensure that $\{y_n^{(m)}\}_{n,m\in Z}$ is orthonormal basis for L^2(R).

Since $V^{(0)} \subset V^{(-1)}$ the scaling function $j(t)$ may be expressed as a linear combination of the functions: $j_n^{(-1)}(t) = \sqrt{2}\, j(2t-n)$, which span $V^{(-1)}$. In general, $j(t) = \sqrt{2}\sum_{n=-\infty}^{\infty} h_0[n]\, j(2t-n)$, where $h_0[n]$ is some sequence of weights.

Forward case in DWT is known as analysis, inverse case – synthesis. So, denote $g_0[n]$ and $g_1[n]$ as filters of synthesis, $h_0[n]$ and $h_1[n]$ as filters of analysis. Between $h_0[n]$ and $h_1[n]$ exists following relation: $h_1[n] = (-1)^{n+1} h_0[-(n-1)]$. Functions $y(t)$ and $j(t)$ are orthonormal, so for $y(t)$ we have following equation:

$$y(t) = \sqrt{2}\sum_{n=-\infty}^{\infty} h_1[n]\, j(2t-n) \qquad (6.3)$$

So, we have scaling function: $\varphi(t)=\sqrt{2}\sum_{n=-\infty}^{\infty}h_0[n]\varphi(2t-n)$ (6.4)

detail function: $\psi(t)=\sqrt{2}\sum_{n=-\infty}^{\infty}h_1[n]\varphi(2t-n)$ (6.5)

$$x=\underbrace{\sum_{n\in Z}y_0^{(0)}[n]\phi(t-n)}_{\in V^{(0)}}=$$

$$\downarrow h_0 \downarrow h_1$$

$$\underbrace{\sum_{n\in Z}y_0^{(1)}[n]\phi_n^{(1)}(t)}_{\in V^{(1)}}+\underbrace{\sum_{n\in Z}y_1^{(1)}[n]\psi_n^{(1)}(t)}_{\in W^{(1)}}$$

$$\downarrow h_0 \downarrow h_1$$

$$\underbrace{\sum_{n\in Z}y_0^{(2)}[n]\phi_n^{(2)}(t)}_{\in V^{(2)}}+\underbrace{\sum_{n\in Z}y_1^{(2)}[n]\psi_n^{(2)}(t)}_{\in W^{(2)}}$$

$$\downarrow h_0 \downarrow h_1$$

Figure 6-2. Expand of signal using analysis filters.

Analysis of signal is illustrated in Figure 6-2. The input signal, $x(t)$, is characterized at some resolution, say $V^{(0)}$, by the discrete sequence $y_0^{(0)}[n]$, such that $x=\sum_{n\in Z}y_0^{(0)}[n]\varphi(t-n)$. This sequence can be decomposed into low- and high-pass subband sequences, $y_0^{(1)}[n]$ and $y_1^{(1)}[n]$, using the analysis system of the two channel subband transform.

6.2 Practical aspects for DWT

In our work we consider lossless compression schemes, so for DWT we need also lossless mode: close approximations to spline 5/3 transform. Filters for analysis have following form:

$$h_0[n] = \begin{cases} 0.75, n = 0 \\ 0.25, n = \pm 1 \\ -0.125, n = \pm 2 \end{cases} \quad \text{low-pass filter}$$

$$h_1[n] = \begin{cases} 1, n = 0 \\ -0.5, n = \pm 1 \end{cases} \quad \text{high-pass filter}$$

Applying on the signal such filters have the following form:

Low-pass filter $\sum_{k=-2}^{2} h_0(k) x_{n-k} = -\frac{1}{8} x_{n+2} + \frac{1}{4} x_{n+1} + \frac{3}{4} x_n + \frac{1}{4} x_{n-1} - \frac{1}{8} x_{n-2}$ (6.6)

High-pass filter $\sum_{k=-1}^{1} h_1(k) x_{n-k} = -\frac{1}{2} x_{n+1} + x_n - \frac{1}{2} x_{n-1}$ (6.7)

To obtain filters of synthesis we used formulas:

$$g_0[n] = h^{-1}(-1)^n h_1[n],\ g_1[n] = h^{-1}(-1)^n h_0[n] \tag{6.8}$$

where h is subband gain factor, for reversible DWT we have to define η=1.

In practice we apply DWT on images, which are represented as two-dimensional array, which contains value of color of each pixel. In such case, we have to process two-dimensional DWT. Two-dimensional transform implies of consecutive applying of the subband transform separably at first to the columns and then the rows of the two dimensional sequence (image array). After applying DWT to initial array, x[**n**], we have four subbands: LL subband, which contains scale information after applying low-pass analysis in vertical and horizontal directions; HL (horizontally high-pass) subband and it involves the application of the low-pass analysis in the vertical direction and the high-pass analysis in the horizontal direction; LH (vertically high-pass) subband and it involves the application of the high-pass analysis in the vertical direction and the low-pass analysis in the horizontal direction, and HH subband similarly. Finite sequence with coefficients is obtained by applying the one dimensional subband transform first to each column of x[**n**] and then to each row of the result. Similarly, the reverse transformation is obtained by applying the reverse one dimensional transform first to each row and then to each column of the result.

In practice for realizing such process Mallat’s *pyramidal algorithm* [Mallat89] is used, which is described below.

6.3 Pyramidal algorithm for practical realization of DWT

Let us put DWT coefficients into transformation matrices H and G applied to the data vector. H works as a smoothening filter (low-pass filter), and G works to bring out data's “detail” (high-pass filter). Wavelet coefficient matrix is applied to the data in hierarchical order. The wavelet coefficients are arranged so that odd rows contain an ordering of wavelet coefficients that act as smoothening filter, and the even rows contain an ordering of wavelet coefficients that act to bring data's detail. The matrix is first applied to the original, full-length data vector. Then vector is smoothed and decimated by half and the matrix applied again. Process continues until a trivial number of data remain. That is, each matrix application brings out higher resolution of the data while at the same time scaling the remaining data. The output of discrete wavelet transform (DWT) consists of the remaining “scale” components, and all of the accumulated “detail” components [Mallat89].

Mallat’s pyramidal algorithm is processed finite set A^n of $n = 2^N$ input data. Filters H and G applied to this data and create output streams that are half of the length of the original input. In such notations forward transform can be described by the following equations: $A^{j-1} = HA^j$ (low-pass filter), $B^{j-1} = GA^j$ (high-pass filter), $j = N,..,1$.

Equation for inverse transform: $A^j = H^* A^{j-1} + GB^{j-1}$

Matrices are defined as follows: $H_{ij} = \frac{1}{2} c_{2i-j}$, $G_{ij} = (-1)^{j+1} c_{j+1-2i}$

Note filter matrices G and H have twice as much columns as rows. Forward wavelet transform starts with G and H of size $n \times 2n$. At each step of transform calculated vector A^{j-1} (and B^{j-1}) is twice as short as A^j **(**B^j**)**. The number of columns and rows in G and H decreases by 2 with each step, until the limit of 1×2 reached and the last $A^{j-1} = A^0$ and $B^{j-1} = B^0$ produced; both contain only one element or we reached predefined levels of transform. Inverse wavelet transform reverses this process.

$$H = \frac{1}{2}\begin{bmatrix} c_1 & c_0 & & & & & & \\ c_3 & c_2 & c_1 & c_0 & & & & \\ & & c_3 & c_3 & c_1 & c_0 & & \\ & & & & c_3 & c_2 & c_1 & c_0 \end{bmatrix}, \; G = \frac{1}{2}\begin{bmatrix} c_0 & -c_1 & -c_2 & -c_3 & & & & \\ & & c_0 & -c_1 & -c_2 & -c_3 & & \\ & & & & c_0 & -c_1 & -c_2 & -c_3 \\ & & & & & & c_0 & -c_1 \end{bmatrix}$$

By their construction H and G are orthonormal: $HG^* = 0$.

In practice no matrix multiplication performed. Rather data values f_i convolved with filter coefficients.

The output of low-pass filter $(Hf)_i$ is: $a_i = \frac{1}{2}\sum_{j=1}^{n} c_{2i-j+1} f_i, i = 1 \ldots \frac{n}{2}$. (6.9)

The output of high-pass filter $(Gf)_i$ is: $b_i = \frac{1}{2}\sum_{j=1}^{n} (-1)^{j+1} c_{j+2-2i} f_i, i = 1 \ldots \frac{n}{2}$ (6.10)

In many cases the odd, or low-pass filter has the most of the “information content” of the original input signal. The even, or high pass output contains the difference between the true input signal and the value of the reconstructed input if it were to be reconstructed only from the information given by the odd output. In general higher-order wavelets tend to put more information into the odd output and less into the even output. For reconstruction applying reverse low-pass filter has the form:

$$f_i^L = \sum_{j=1}^{n/2} c_{2i-j} a_i, i = 1,..,n \tag{6.11}$$

Applying reverse high-pass filter has the form:

$$f_i^H = \sum_{j=1}^{n/2} (-1)^{j+1} c_{j+1-2i} b_i, \ i = 1,..,n \tag{6.12}$$

The perfect reconstruction is a sum of the inverse low-pass and inverse high-pass filters and the perfectly reconstructed signal is $f = f^L + f^H$.

Complexity of pyramidal algorithm is $O(n)$. We have to perform number of steps:

$$n + \frac{n}{2} + \frac{n}{2^2} + \ldots + \frac{n}{2^k} \le 2n \tag{6.13}$$

The complexity for forward and inverse DWT is $O(n)$.

6.4 Context template based on DWT coefficients

As a context template we use template, which is illustrated in Figure 6-3. Basis of this template is the same as illustrated in Figure 5-1. The main concept for current context is using DWT coefficients for context modeling; in contrast to template from Figure 5-1 here we use DWT coefficient instead of neighboring pixels. It means that we apply template from Figure 5-1 to DWT coefficients. The main idea for using DWT coefficient is existence strong correlation between DWT coefficients.

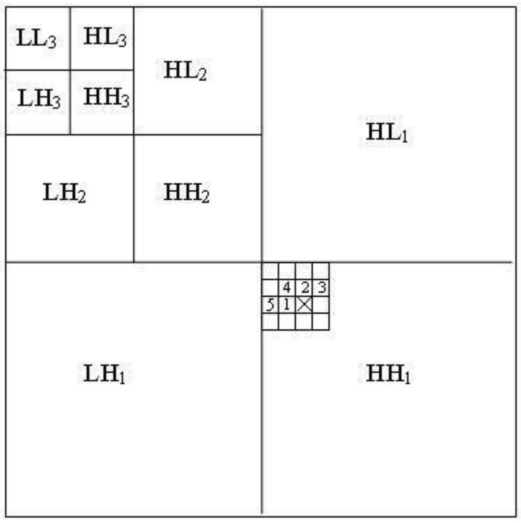


Figure 6-3. Context template based on DWT coefficients with context size 5.

We apply such template in raster scan order on DWT coefficients. Choosing parameter in such case is the number of adjacent coefficients. We can take this value in range $[1, 30]$.

6.5 Context template based on conjugate DWT coefficients

Current context template based on DWT coefficients with using one conjugate coefficient from up-level. Such context template is illustrated in Figure 6-4.

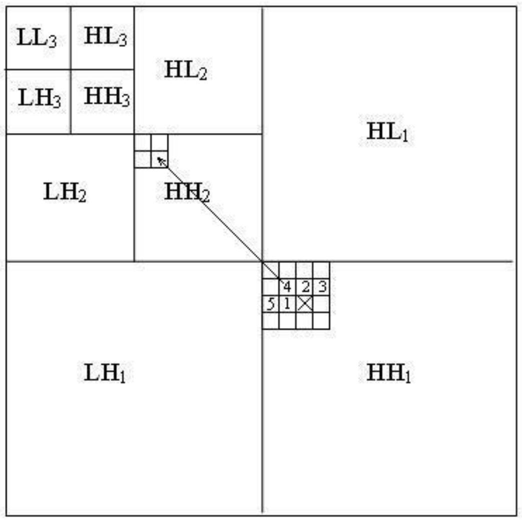


Figure 6-4. Context template based on conjugate DWT coefficients with context size 5.

The main concept for using such template is correlation between coefficients on different DWT levels. The idea of such approach is replacing of neighboring pixel from context template (see Figure 6-3) on corresponding (conjugate) up-level DWT coefficient for pixel on position X (see Figure 6-4). Here we use principle that 1 coefficient from up-level has 4 conjugate coefficients from low-level. As illustrated in Figure 6-4 for coefficient from HH_1 subband we take conjugate coefficient from HH_2. It makes sense for reverse procedure, when we will make decoding.

Obviously, we can make such replacement for any neighboring coefficient from current context template. We made set of experiments, where such operation was produced sequentially for coefficients in template with size 5.

6.6 Compression scheme

Let us denote current lossless compression algorithm as General Context tree based on DWT coefficients (GCT-W). This compression method is lossless algorithm by context modeling based on DWT coefficients. Our algorithm has following steps: discrete wavelet transform, context modeling and entropy encoding. Such process is illustrated in Figure 6-5.

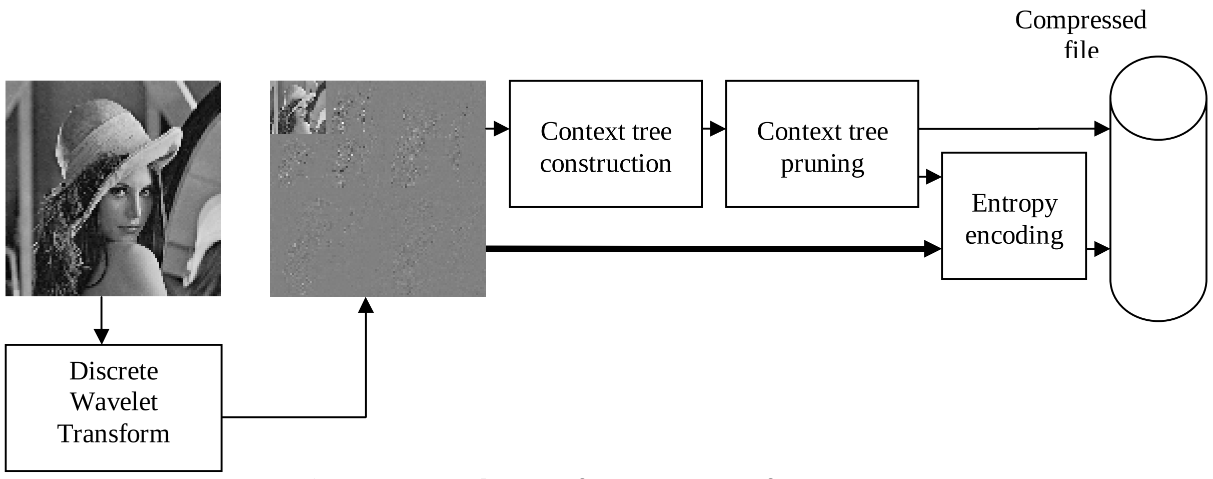


Figure 6-5. Scheme of compression for GCT-W.

First step here is DWT. Low-pass and high-pass filters are defined by formulas (6.6) and (6.7). Detail description about DWT is in Section 6.1, Section 6.2 and Section 6.3.

Second step is context modeling, which applies on DWT coefficients. The main idea for using DWT coefficients is the existence of correlation between such coefficients. For context modeling we have two different context templates, which are described above in Section 6.4 and Section 6.5. In other word we make GCT construction, which was described in Section 4.2.3.

Third step is pruning of GCT. Context tree is pruned for finding the optimal combination of siblings. After this operation we have incomplete n-ary tree structure with optimal combination of siblings, which was found by comparing the parent node and its children nodes (see Section 4.2.4). Forth step is entropy based encoding (QM-coder implementation) of DWT coefficients by using information stored in constructed pruned context tree structure (see Section 2.4).

After these steps we have compressed file. Reverse process includes entropy based decoding and reverse DWT.

7. General Context Tree based on local gradients

The lossless compression algorithm used context modeling based on local gradients is reperesented in this section. The method combines GCT approach (see Section 4.2) with a predictor, which was proposed in LOCO-I method (see Section 3.2). Modeling step realizes context modeling based on local gradients. Definition of local gradient is the same as for LOCO-I, but quantization of gradient is different; here we apply uniform scalar quantizer defined by the quantization range.

For coding process the arithmetic coder (QM-coder) is used, the feature of its process is coding of prediction error, so in current algorithm uses statistical and predictive modeling. This lossless compression method uses advantages from GCT approach and LOCO-I compression algorithm.

7.1 Prediction modeling

The prediction modeling consists of the following steps:

1. Prediction of the current pixel value;
2. Calculation of prediction error;
3. Collecting the error probability distribution.

During first step the prediction is processed, it means calculation value for current pixel based on a subset of the available past sequence (causal template). The idea about such prediction is used in LOCO-I (see Section 3.2). The prediction is based on the causal template illustrated in Figure 7-1, where ***U*** is current pixel and a, b, c and d are adjacent pixels in the corresponding positions as illustrated in figure.

c	a	d
b	**U**	

Figure 7-1. Causal template.

Initial image is observed pixel by pixel in raster-scan order. At each time instant i, and after having scanned past data $U^{i-1} = U_1 U_2 ... U_{i-1}$, we try to make inferences on current pixel U_i. The prediction of U_i is based on primitive test to detect vertical or horizontal edges. If an edge is not detected, then the guessed value is $a+b-c$.Specifically, predictor guesses:

$$\hat{U}_i = \begin{cases} \min(a,b) & if\ c \geq \max(a,b) \\ \max(a,b) & if\ c \leq \min(a,b) \\ a+b-c & otherwise \end{cases} \tag{7.1}$$

Assuming, without loss of generality, that $a \leq b$, then the predictor of (7.1) can be interpreted as picking a in many cases where a vertical edge exists left of the current location, b in many cases of an horizontal edge above the current location, or a plane predictor $a+b-c$ if no edge has been detected. So, here the same principle for predictor as in LOCO-I method is used.

Following step is calculation of prediction error, which is given by:

$$e_i = U_i - \hat{U}_i \tag{7.2}$$

After calculation of prediction error its probability distribution is obtained into source image, which is observed in raster-scan order.

7.2 Definition of local gradient

Each pixel ***U*** in the image has local gradient, thus capturing the level of activity (smoothness, edginess) surrounding a pixel, which governs the statistical behavior of prediction errors.

Local gradient is represented as the following differences:

$$g_1 = d - b,\ g_2 = b - c,\ g_3 = c - a \tag{7.3}$$

where a, b, c and d are values of neighboring pixels (see causal template from Figure 7-1).

After calculating of local gradient it is quantized by applying uniform scalar quantizer (see Section 7.3 below). We need the quantization step, because differences (7.3) have large region of support (we will construct context tree, by using local gradient as context. In such case we have restriction of computer memory for constructing and processing large context tree).

In practice we use grayscale images with colors from the range $[0, 255]$, it means that our differences have region of support $[-255, 255]$. In other words, for quantization the following mapping is made:

$$g : [-255, 255] \rightarrow [p, q] \tag{7.4}$$

by applying following equation:

$$\tilde{q} = \frac{q-p}{510} g + \frac{p+q}{2} \qquad (7.5)$$

where p and q are boundaries of quantization region, $\tilde{q}$ is quantized value of g. This uniform scalar quantizer divides initial range $[-255, 255]$ into parts with equal size.

Obviously, for context tree construction the quantized local gradient g_1, g_2, g_3 is applied as context template (see Section 7.4).

7.3 Scalar quantization

The scalar quantizer is very useful method for lossy compression schemes. The input of quantizer is the original data, and the output is always one among a finite number of levels.

Quantizer can be described as a function that maps each element in a subset of the real line to a particular value in that subset [Max60, Lloyd82]. Such function has discrete set as output values, and this set is usually finite. Obviously, this is a process of approximation, and a good quantizer is one which represents the original signal with minimum loss or distortion.

A quantizer can be specified by its input partitions and output levels (also called reproduction points). If the input range is divided into parts with equal spacing, then the quantizer is termed as a uniform quantizer, otherwise it is termed as a non-uniform quantizer. A uniform quantizer can be easily specified by its lower bound and the step size, or its number of output levels when range is divided into parts with equal size. Also, implementing a uniform quantizer is easier than a non-uniform quantizer.

Let us consider scalar quantizer with output levels M in general case. Partitioning the real line into M disjoint intervals is denoted in a following way:

$$I_q = [t_q, t_{q+1}), q = 0, 1, \ldots, M-1 \qquad (7.6)$$

with

$$-\infty = t_0 < t_1 < \ldots < t_M = +\infty \qquad (7.7)$$

Within each interval, a point $\hat{x}_q$ is selected as the output value (or codeword) of I_q. In this case if initial range is divided into equal parts, we have uniform scalar quantizer, otherwise, no-uniform. A scalar quantizer is following mapping:

$$Q : R \rightarrow \{0, 1, \ldots, M-1\} \qquad (7.8)$$

Specifically, for a given input value x, $Q(x)$ is the index q of the interval I_q which contains x. Just the same way a quantizer divides its input into discrete levels of output, a dequantizer is one which receives the output levels of a quantizer and converts them into normal data. The dequantizer is given by:

$$\overline{Q^{-1}}(q) = \hat{x}_q \tag{7.9}$$

The scheme of uniform scalar quantization is illustrated in Figure 7-2.

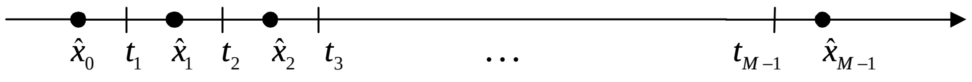


Figure 7-2. Graphical representation of uniform scalar quantization.

This figure shows that when $x \in I_q = [t_q, t_{q+1})$, that $\overline{Q^{-1}}(Q(x)) = \overline{Q^{-1}}(q) = \hat{x}_q$. Clearly, the t_q can be through of as thresholds, or decision boundaries for the $\hat{x}_q$. For instance, if the input value $t_1 \le x < t_2$, the quantized version of x is $\hat{x}_1$ (index = 1). Specifically, $Q(x) = 1$ and $\overline{Q^{-1}}(Q(x)) = \overline{Q^{-1}}(1) = \hat{x}_1$.

Measure of optimality for quantizer is quantization error, which is given by following equation:

$$e = x - \hat{x} \tag{7.10}$$

where x is input value and $\hat{x}$ is quantized version of x.

Optimal quantizer has to satisfy following conditions:

1. Given the output levels or partitions of the encoder, the best decoder is one that puts the reproduction points $\hat{x}_q$ on the centers of mass of the partitions. This is known as *centroid condition.*
2. Given the reproduction points $\hat{x}_q$ of the decoder, the best encoder is one that puts the partition boundaries exactly in the middle of the reproduction points, i.e. each input value x is translated to its nearest reproduction point. This is known as *nearest neighbour condition*.

7.4 Context template based on local gradient

Current context template is based on quantized local gradient, which is defined by formulas (7.3) and quantized by using formula (7.5), quantization range $[p, q]$ have to be smaller than

$[-255, 255]$, because context tree will have impossible big size for computer memory for such big initial range of local differences.

Here quantized local gradient is basis of context template, which is illustrated in Figure 7-3. Each pixel in current image has local gradient: g_1, g_2, g_3 (see Section 7.2). For calculation differences g_1, g_2, g_3 by formulas (7.3) we use causal template, which is illustrated in Figure 7-1. Detail description about local gradient is in Section 7.2 above.

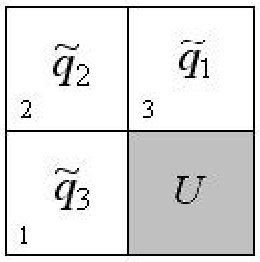


Figure 7-3. Context template based on local gradient.

Let us denote the quantized values, which are obtained by applying uniform scalar quantizer to differences g_1, g_2, g_3 by using formula (7.5), as $\tilde{q}_1, \tilde{q}_2, \tilde{q}_3$. These quantized values $\tilde{q}_1, \tilde{q}_2, \tilde{q}_3$ form the context template for current pixel ***U***. Just as template from Figure 5-1 we can say, that instead of first adjacent pixel we take $\tilde{q}_3$, second value for context template is $\tilde{q}_2$ and third is $\tilde{q}_1$ respectively.

7.5 Compression scheme

Let us denote current algorithm as General Context Tree based on local gradients (GCT-G). Current lossless compression method is proposed by context modeling based on local gradients. The main idea of this algorithm is applying the context modeling based on local gradients and encoding the value of prediction error by using entropy encoding. Scheme of such algorithm is illustrated in Figure 7-4.

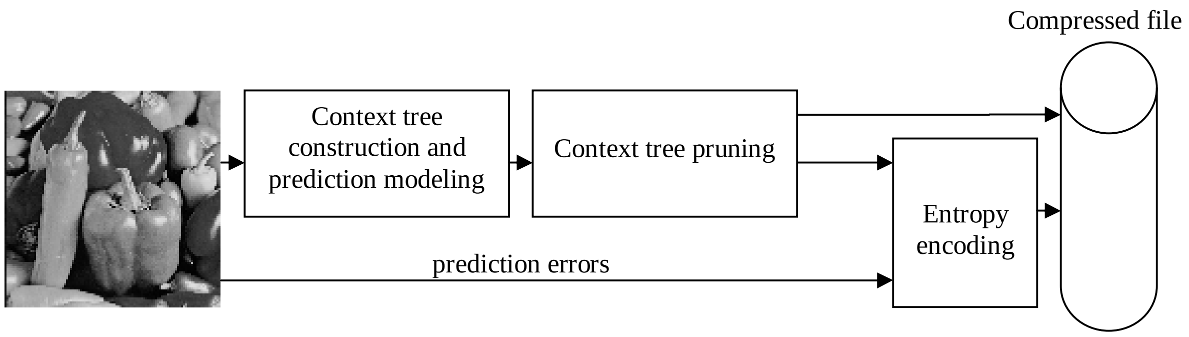


Figure 7-4. Scheme of compression for GCT-G.

The first step is context tree constructing and prediction modeling (more detail description about GCT construction is in Section 4.2). Prediction modeling (see Section 7.1) is processed together with context tree construction. Prediction error is calculated for each processed sample in the source image during the context modeling. We process image in raster-scan order and for each pixel calculate local gradient and quantize it (see Section 7.2 and Section 7.4). After experiments we found optimal quantization region for local gradient, it is $[0, 66]$. Such quantized local gradient here is used as basis for context template depicted in Figure 7-3 (see Section 7.4).

Second step is GCT pruning. After this step our context tree will have the optimal combination of siblings. For pruning tree the comparing of parent node and its children nodes is performed (see Section 4.2.4). Last step is entropy based encoding (see Section 2.4) of prediction error with using context model based on quantized local gradients. After these steps we have compressed file.

Reverse process includes entropy based decoder. After decoding we have prediction error, by using this value and predictor (7.1) we can calculate decompressed value of pixel.

8. Comparative analysis of considered methods

Lossless compression algorithms with different approaches were considered in this work. We took into account following methods: JPEG2000, LOCO-I, CALIC; and developed GCT-I, GCT-W, GCT-G algorithms. All of them have common property: the context modeling in their compression schemes. Each presented algorithm has individual features. The important properties of these algorithms are represented in Table 8-1.

Table 8-1. Properties of lossless compression algorithms.

Parameter	**Lossless compression algorithms**					
	GCT-I	GCT-W	GCT-G	JPEG2000	LOCO-I	CALIC
Context	Neighboring pixels, i.e. intensity of adjacent samples	Neighboring DWT coefficients	Quantized local gradient	Neighboring quantized DWT coefficients	Quantized local gradient	Quantized local neighborhood pixel values combined with quantized error energy
Number of various contexts	430	125	208	18	365	576
Prediction modeling step	no	no	yes	no	yes	yes
Encoding value	Value of sample	Value of DWT coefficient	Prediction error	Value of DWT coefficient	Prediction residual	Difference between actual pixel and the corrected prediction
Coder	Arithmetic coder	Arithmetic coder	Arithmetic coder	Adaptive binary arithmetic coder	Golomb codes	Adaptive *m*-ary arithmetic coder
Statistical modeling	Semi-adaptive	Semi-adaptive	Semi-adaptive	Adaptive	Adaptive	Adaptive
Information about used contexts stored in the header of compressed file	Contains information about used contexts. In average it equals 250 bytes	Contains information about used contexts. In average it equals 70 bytes	Contains information about used contexts. In average it equals 168 bytes	Contains metadata, which includes information about used contexts of each code-block. In average it equals 66 bytes	No information about used contexts in the header.	No information about used contexts in the header.

The number of various contexts is important property of context modeling. For GCT-I, GCT-W, GCT-G algorithms we can calculate only average value of this feature, because number of contexts here depends on processed image. These three algorithms have common scheme for context modeling; they store statistical information about contexts in special tree structure. For GCT-I and GCT-W algorithms the number of contexts is (number of colors)$^{\text{size of context}}$. For GCT-G number of contexts depends on quantization region $[p, q]$ of local gradient, so the value is $(q - p + 1)^3$. After construction such tree is pruned to find optimal combination of siblings with excluding contexts with small contribution, therefore we have different number of contexts for each compressed image. Number of contexts is decreased after pruning.

Along with context modeling algorithms LOCO-I, GCT-W, and CALIC use prediction modeling step. During this step prediction error and its probability distribution are calculated. It is important step, because prediction value is encoding value for these three compression algorithms.

Choosing of encoding value is important moment for context modeling. Actually, motivation of context modeling is to take advantage of correlation between encoding value and context. Such correlation has to exist for good compression result. As we can see from Table 8-1 prediction residual is used as encoding value, when quantized local gradient is used as contexts. GCT-G, LOCO-I and CALIC use such context and encoding value during context modeling, because local gradient and prediction error correlate. In GCT-W and JPEG2000 correlation between DWT coefficients are used for this purpose. Only in GCT-I algorithm correlation between neighboring pixels is used, it allows to achieve advantage on images, where linear prediction methods do not work.

Most of our algorithms use the entropy-based coder, in other words arithmetic coder. Only LOCO-I algorithm use Golomb codes to encode prediction residual. Arithmetic coder is flexibility, which means that it can be used in conjunction with any model that can provide a sequence of probabilities. This property allows its using for algorithms: GCT-I, GCT-W, GCT-G, JPEG2000, and CALIC.

All our considered lossless compression methods use statistical modeling in their compression schemes. Statistical modeling can be classified into three categories: static modeling, semi-adaptive modeling, and adaptive modeling. Algorithms GCT-I, GCT-W, GCT-G use the semi-adaptive modeling, i.e. they are two-pass methods, which have no update the model during compression and need the storing of statistical model. JPEG2000, LOCO-I, and CALIC use adaptive modeling, they are one-pass methods, which update the statistical model during compression. Advantage of adaptive approach is that such algorithms do not need side information.

The information about used contexts in the header of compressed file for GCT-I, GCT-W, GCT-G and JPEG2000 methods is stored. Structure of JPEG2000 compressed file differs from others. It consists of the blocks; each of them contains the context information and compressed stream of code block. Such structure makes conditional upon compression scheme of JPEG2000 standard, where code-blocks separately are compressed. In average the information about used context takes about 0.3% from compressed file size for GCT-I method, 0.2% for GCT-W, 0.1% for GCT-G, and 0.15% for JPEG2000 standard.

All considered algorithms have special advantages. CALIC's advantage is high coding efficiency in accomplished with relatively low time and space complexities. Application of DWT allows perform zooming copies if the image without full reverse transform in JPEG2000 standard and GCT-W algorithm, which support quick and easy zooming. The main advantage of LOCO-I is low complexity of compression algorithm. Advantage of GCT-G algorithm is using larger quantization region for local gradient as in LOCO-I method, due to this fact the GCT-G has improvement about 2% as compared with LOCO-I in compression efficiency. Advantage of GCT-I algorithm is effective context modeling on class of images, which characterize by small number and sharp changing of colors, on such images linear-predictive techniques work ineffective.

9. Experiments and Discussions

In this section the results of experiments are considered. For experiments different test sets of images were used: natural “smooth” images (see Appendix 4), medical images (see Appendix 5), natural noise images (see Appendix 6), and astronomical images (see Appendix 7). Description of these test sets is in Table 9-1.

Table 9-1. Test sets information.

Test set	Images	Average image size	Top number of colors	Bit-rate, bits per pixel	Characteristics
Medical images	5	328 x 290	256	8	Small number of colors and sharp changing of color on adjacent samples
Natural smooth images	12	430 x 425	256	8	Images with smooth color changing
Natural noise images	5	320 x 348	256	8	Smooth images with noise
Astronomical images	10	665 x 466	256	8	Images with small noise

Analysis of our results is made by comparing results of following known methods: LOCO-I, JPEG2000, CALIC (detail description about these methods is in Section 3).

Let us assign short names to our algorithms:

1. **GCT-I –** lossless compression algorithm, which uses approach of General Context Tree based on Intensity. Main feature here is context modeling based on neighboring pixels from initial image. This algorithm constructs general context tree, prune it and makes entropy encoding (detail description of this method is in Section 5).
2. **GCT-W –** lossless compression algorithm, which uses approach of General Context Tree based on DWT coefficients. During modeling step it applies context modeling based on discrete wavelet transform coefficients with using GCT approach (detail description of this method is in Section 6).
3. **GCT-G –** Context Modeling based on local gradients algorithm applies context modeling based on local gradients with using GCT approach. In this algorithm context tree is constructed based on quantized local gradients and encoding makes for prediction error (detail description is in Section 7).

In Table 9-2 are results of our algorithms (GCT-I, GCT-W, GCT-G) and known algorithms (CALIC, LOCO-I, JPEG2000). Results are given in bit-rate and evaluated in bits per pixel. Initial bit-rate for all images is 8 bits per pixel.

Table 9-2. The compression efficiency (bits per pixel).

Test set	GCT-I	GCT-G	GCT-W	CALIC	LOCO-I	JPEG2000
Medical images	**3.261**	3.492	4.088	3.415	3.530	3.917
Natural smooth images	5.146	4.797	4.858	**4.422**	4.541	4.653
Natural noise images	5.558	4.995	4.954	**4.634**	4.740	4.807
Astronomical images	3.814	3.523	3.842	**3.427**	3.628	3.770

These experiments were processed for comparing effectiveness between presented algorithms. Detail results description for each image test set is in Appendix 1, Appendix 2 and Appendix 3.

Testing produced on following test sets: natural smooth images, medical images, astronomical and natural noise images. So, on smooth natural images (see Table 9-2) results for algorithms, which use linear dependences between pixels, are better.

In general case results for GCT-G are comparable to JPEG2000 and LOCO-I. In Table 9-2 we can see that on medical images we have better compression for our algorithms, it depends on structure of images. GCT-I gives better result on such class of images, because here we have small number and sharp changings of colors, because image has special structure, where predictors of known methods (LOCO-I, CALIC) do not work here. Applying context model based on DWT coefficients and local gradients also try to use linear dependences between pixels, but sometimes structure of image does not allow it in general.

For comparing medical images and natural images we considered intensity histograms for images.

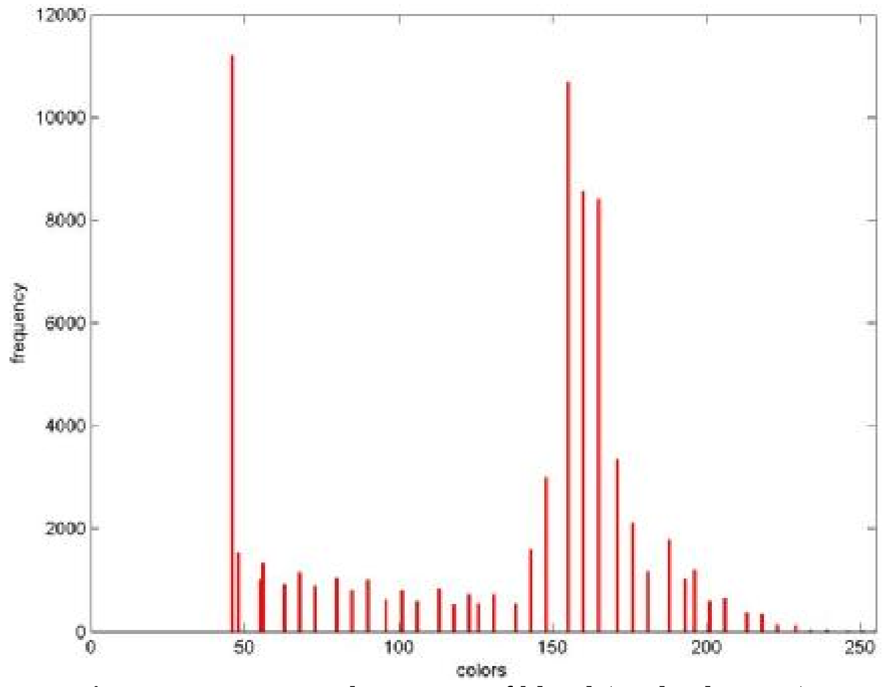


Figure 9-1. Intensity histogram of blood (medical image).

As illustrated in Figure 9-1 the number of colors in medical images is small, this property allows to constructing of context model with small number of contexts, which will full describe current image. For comparing we constructed intensity histogram for natural smooth image, it is illustrated in Figure 9-2.

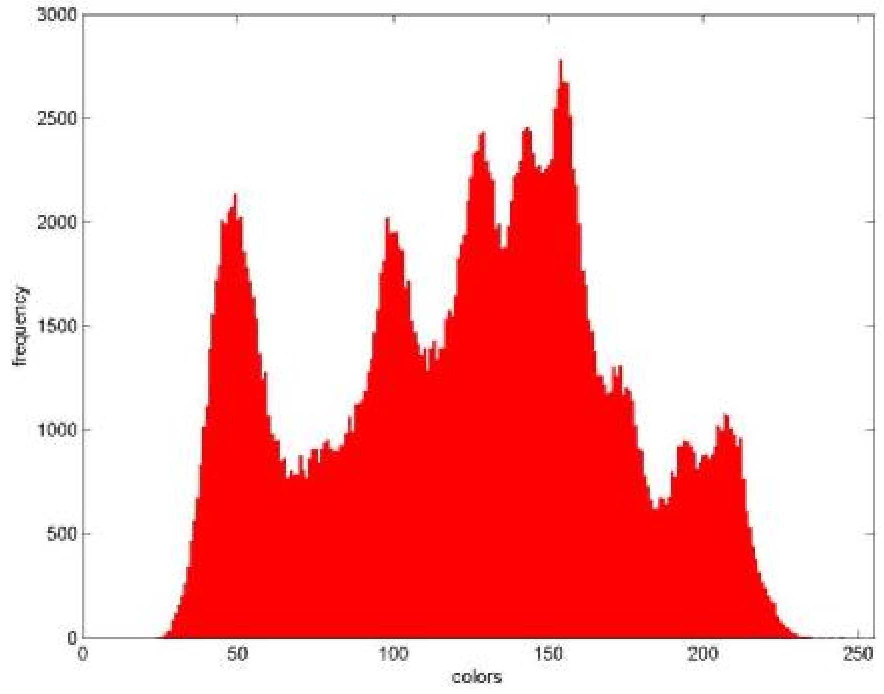


Figure 9-2. Intensity histogram of lena (natural smooth image).

Natural noise images have the similar intensity histograms as natural smooth images, it means that such images have bigger number of colors as medical images.

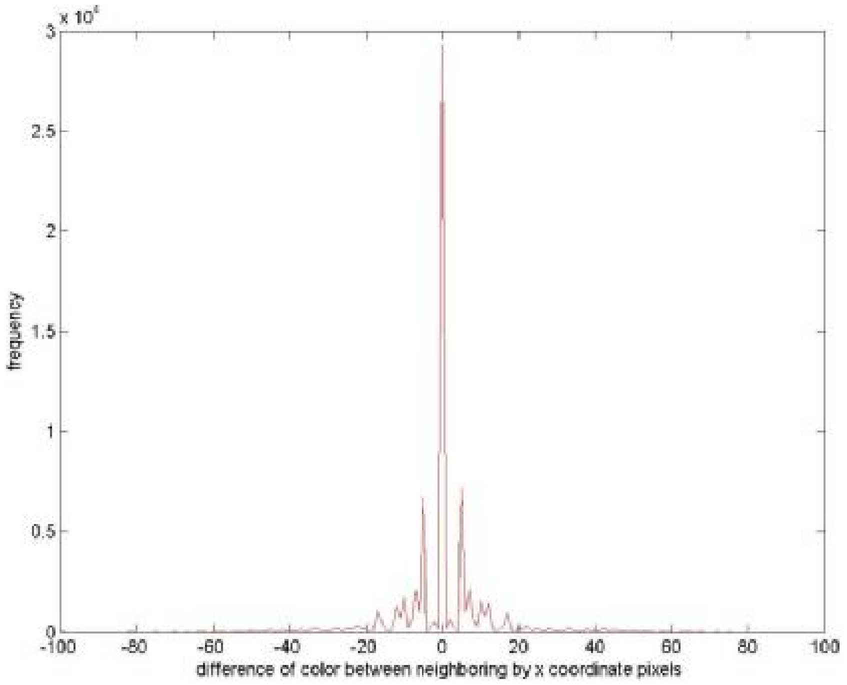


Figure 9-3. Frequency of differences between adjacent by *x* coordinate pixels for blood.

In Figure 9-3 and Figure 9-4 frequencies of differences between neighboring by *x* coordinate pixels for medical image and for smooth natural are illustrated. For medical image differences between adjacent pixels are in small region centered near zero. Figure 9-3 shows that such king of images have sharp changings of colors. This property stipulates for bad work of linear-predictive coding technique, because between pixels here we have not linear dependences.

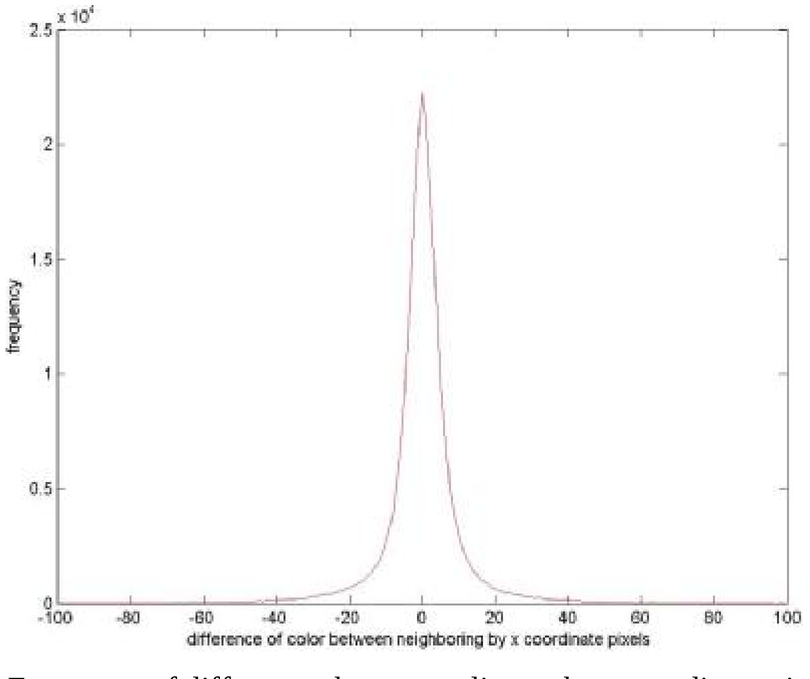


Figure 9-4. Frequency of differences between adjacent by *x* coordinate pixels for lena.

After analyzing of image properties we obtained, that natural noise images have approximately same characteristics as natural smooth images: big number of colors and linear dependence of color changing. These properties ensured that on this test set CALIC gives better compression result as GCT-I.

The main properties for medical images are small number and sharp changings of colors. Small number of colors allows to constructing of context model with smaller number of contexts, this fact gives opportunity to save computer memory for storing context model. Sharp changing of colors shows that between pixels no linear dependences, therefore on such images GCT-I gives better compression result.

Following experiment was produced for comparing the compression efficiency on astronomical images.

Table 9-3. The compression efficiency for astronomical images (bits per pixel).

File name	CALIC	GCT-G	LOCO-I	JPEG2000	GCT-I	GCT-W
Astr1	4.271	4.420	4.589	4.777	4.836	4.858
Astr2	5.488	5.662	5.696	6.296	5.775	6.363
Astr3	2.280	2.212	2.299	2.399	2.720	2.427
Astr4	1.245	1.282	1.370	1.356	1.445	1.403
Astr5	4.371	4.486	4.631	4.558	4.769	4.672
Astr6	0.518	0.602	0.527	0.655	0.650	0.668
Astr7	3.652	3.778	3.977	3.945	4.249	4.096
Astr8	4.982	5.144	5.269	5.647	5.441	5.723
Astr9	4.400	4.519	4.661	4.809	4.796	4.889
Astr10	3.060	3.126	3.265	3.256	3.461	3.321
Average	3.427	3.523	3.628	3.770	3.814	3.842

Comparing the results for different types of images in Table 9-2 we can see that on astronomical images (see Appendix 7) GCT-G method produces better compression than LOCO-I. Detail results of this experiment are in Table 9-3.

The explanation of this result is obviously, because these two algorithms use the same predictor during modeling step (see Section 7.1 and Section 3.1). Improvement is reached by GCT approach in context modeling based on quantized local gradients, quantization range in GCT-G is bigger than in LOCO-I method, so in GCT-G we have bigger number of contexts and therefore more efficient coding, it has about 2% improvement in compression efficiency. On other test sets difference between results of LOCO-I and GCT-G is small, it is about 1-2% on the both hands.

Following experiment was produced for detecting optimal value for DWT levels, denote such value as *D*. Also discrete wavelet transform is used in JPEG2000 standard (see Section 3.1), in this standard typical values are in the range *D*=4 through *D*=8 for lossless mode. We used the same low- and high-pass filters as JPEG2000. So, the question is to detect optimal value for parameter *D* in context modeling based on DWT coefficients.

For comparing were used two types of images: smooth natural images and medical images. For these two sets we have the same dependences. Experiment has two aims: detect *D* and size of context. Size of context means number of adjacent pixels on template (see Figure 6-3 in Section 6.4); also this value is depth of context tree, let us denote size of context template as s. We fixed parameter *D* and changed depth of context tree *s* in range [2; 5]. The aim was detect optimal values for these two parameters. General result of this experiment is in Table 9-4 (more detail description of result is in Appendix 2). The best results are with *D*=8 and depth = 3 or depth = 4.

Table 9-4. The compression efficiency for GCT-W method with different *D* levels (bits per pixel)

Number of DWT levels	**Natural smooth images**				**Medical images**			
	***s*= 2**	***s* = 3**	***s* = 4**	***s* = 5**	***s* = 2**	***s* = 3**	***s* = 4**	***s* = 5**
D = 3	4,861	4,862	4,862	4,862	4,106	4,104	4,102	4,102
D = 4	4,861	4,861	4,861	4,861	4,102	4,099	4,099	4,098
D = 5	4,864	4,865	4,865	4,865	4,093	4,092	4,092	4,090
D = 6	4,862	4,862	4,862	4,862	4,093	4,093	4,091	4,091
D = 7	4,860	4,860	4,861	4,861	4,091	4,090	4,089	4,088
D = 8	4,857	4,858	4,858	4,858	4,091	4,090	4,088	4,088

After depth equals 3 (or 4) we have the same results for bigger size of context, because in practice after certain depth (in our case it is 3) context tree is filled. Such occurrence is obvious, because during pruning of context tree we are excluding contexts with small contribution. Difference between compression efficiency for different context size is thousandth, optimal value here is such value after that context tree is filled (i.e. context tree contains all significant contexts). So, optimal value for contexts size approximately is 3 or 4.
Value for DWT levels has small contribution into compression. Bit-rate in bits per pixel for compressed images with different DWT levels has thousandth differences. Optimal value for DWT levels in such case is *D* = 6. After comparing results for *D*=6 and *D*=8 we can see, that increasing value of *D* makes small changes in terms of context modeling (we have the same number of colors, approximately equals number of different contexts). But applying bigger

number of *D* requires more time for DWT, therefore we take middle value, which will be best balance between compression quality in bits per pixel and compression time.

Following experiment was about constructing context template based on conjugate DWT coefficients, i.e. with using conjugate coefficient from up DWT level (see Section 6.5). The question is detecting which coefficient we can take instead of neighboring coefficient from context template; such template is illustrated in Figure 6-4. During this experiment templates illustrated in Figure 9-5 are used.

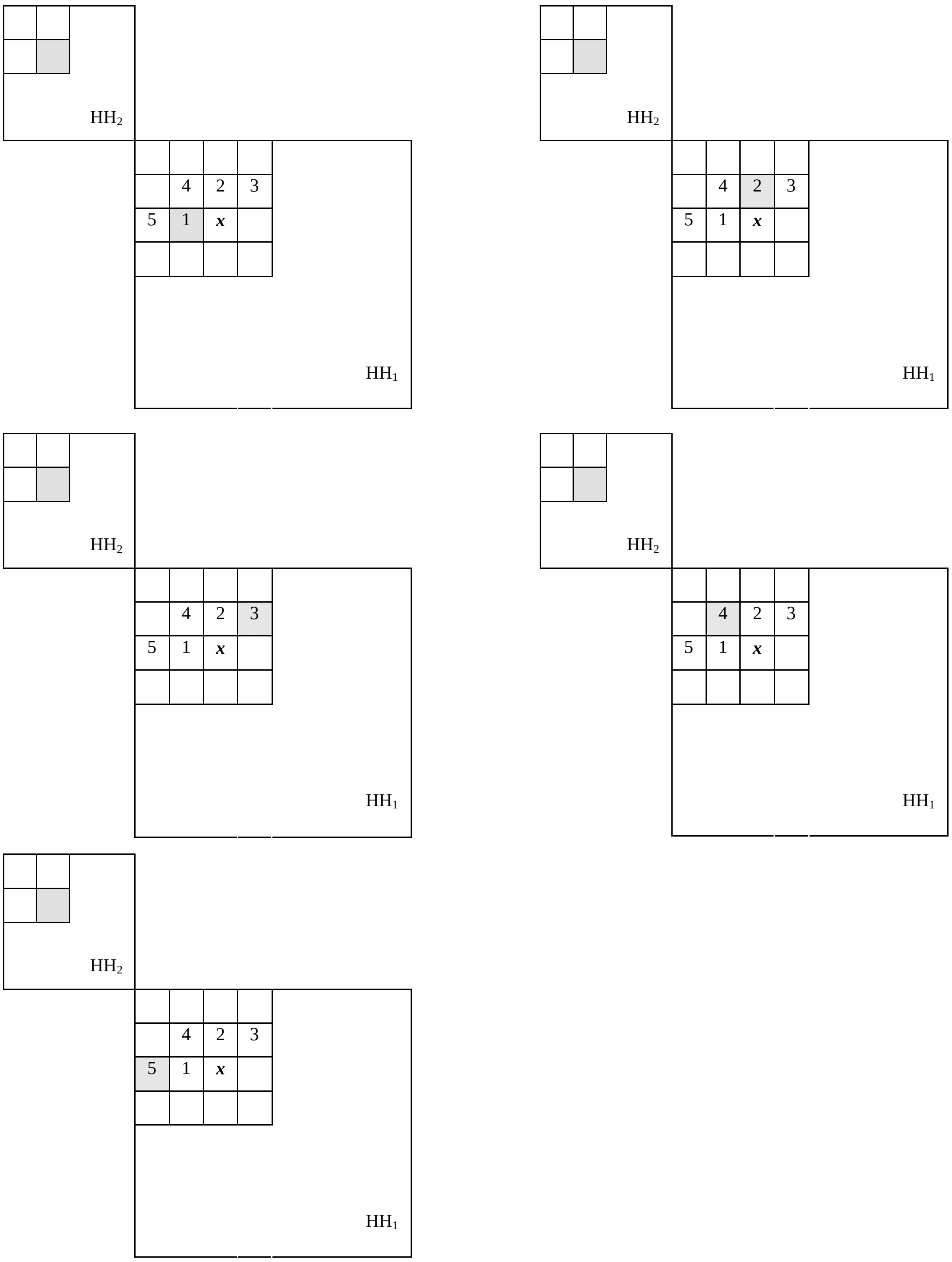


Figure 9-5. Templates based on DWT coefficients with one conjugate, context size is 5.

The formation concept of such tamplates is replacing of neighboring coefficient on conjugate up-level DWT coefficient. Here we use principle, that 1 coefficient from up-level has 4 conjugate coefficients from low-level. For replacing we use coefficient corresponded to coding coefficient on position *x*.

For testing natural smooth images and medical images were used. Results are in Table 9-5. Results are given in bit-rate and evaluated in bits per pixel.

Table 9-5. Bit-rate for algorithm GCT-W with context template with conjugate DWT coefficient, $D = 5$ (bits per pixel)

File name	GCT-W algorithm					
	0 coef.	1 coef.	2 coef.	3 coef.	4 coef.	5 coef.
Barb	5.074	5.199	5.112	5.068	5.068	5.068
Billsface	4.194	4.290	4.206	4.205	4.205	4.205
Boat	4.621	4.690	4.624	4.622	4.621	4.621
Cman	5.261	5.329	5.256	5.247	5.251	5.251
Einstein	5.069	5.102	5.099	5.088	5.088	5.088
Elaine	5.141	5.173	5.157	5.155	5.155	5.155
Goldhill	4.992	4.985	4.986	4.984	4.985	4.985
Lena	4.492	4.584	4.512	4.508	4.507	4.508
Man	5.687	5.729	5.664	5.663	5.663	5.663
Mri	4.905	4.917	4.835	4.839	4.833	4.833
Peppers	4.799	4.859	4.810	4.806	4.806	4.806
Zelda	4.146	4.162	4.159	4.159	4.159	4.159
Average	**4.865**	**4.918**	**4.868**	**4.862**	**4.862**	**4.862**
Hip	2.648	2.697	2.674	2.648	2.645	2.644
Blood	4.853	4.969	4.869	4.857	4.857	4.857
Child	3.715	4.046	4.096	4.033	4.032	4.032
5week	4.035	3.737	3.809	3.721	3.721	3.718
Ultrasound	5.210	5.436	5.282	5.260	5.260	5.260
Average	**4.090**	**4.177**	**4.146**	**4.104**	**4.103**	**4.102**

First column contains results of GCT-W compression method, which use simple template without replacing, it corresponds to 0 replacing pixel from template (see Figure 6-3). Other columns contain results for templates, where some coefficient is replacing on conjugate coefficient from up DWT level (see Figure 9-5).

Results show that using of coefficient from up DWT level does not improve compression efficiency. Bit-rate is the same as for context modeling based on DWT coefficients without using conjugate coefficient from up-level. Reason of this event is that correlation between coding coefficient and neighboring coefficients is approximately same as between coding coefficient and coefficient from up-level.

During developing algorithm of context modeling based on local gradients we need to define quantization range $[p, q]$ for differences g_1, g_2, g_3, which defined by formula (7.3) (see Section 7) and notated as local gradient. All such differences need to be quantized by applying uniform scalar quantizer (7.5). Our experiment was about detecting of optimal quantization range $[p, q]$. So we defined this range by using results of experiments. In our algorithm (see Section 7) we are applying context modeling based on quantized local gradients. Results are in Table 9-6. For experiment was used test set of natural smooth images

Table 9-6. The compression efficiency for GCT-G method in bits per pixel

File name	GCT-G algorithm with quantized region		
	[0, 42]	[0, 66]	[0, 127]
Barb	5.183	5.196	5.233
Billsface	4.141	4.131	4.111
Boat	4.500	4.487	4.495
Cman	4.974	5.004	4.972
Einstein	4.890	4.868	4.916
Elaine	5.201	5.193	5.235
Goldhill	4.923	4.914	4.918
Lena	4.504	4.485	4.496
Man	5.559	5.568	5.602
Mri	4.816	4.776	4.806
Peppers	4.770	4.752	4.761
Zelda	4.203	4.185	4.181
Average	**4.805**	**4.797**	**4.810**

In first column of Table 9-6 is bit-rate of compressed files, where quantized range for differences of local gradient were [0, 42], it means 43 different values for each quantized values $\tilde{q}_1, \tilde{q}_2, \tilde{q}_3$ of g_1, g_2, g_3, for second and third columns [0, 66] and [0, 127] respectively.

After experiment we see that optimal range is [0, 66], it is optimal balance between quality of compression and compression time. Context tree extended after increasing of quantization range $[p, q]$, it took more time for processing and more resources for storing such structure. Difference in bit-rate for different ranges is in thousandth, so we took middle value as optimal.

For estimation GCT-G method we made following experiment: encode of pixel value (or intensity) in context of quantized local gradient (GCT-G-intensity), encode of prediction error in context of image samples (GCT-I-error). Results are in Table 9-7.

Table 9-7. Comparable estimation of GCT-G method (bits per pixel)

File name	GCT-G	GCT-I	GCT-G-intensity	GCT-I-error
Barb	5.196	5.751	7.190	6.990
Billsface	4.131	4.606	6.510	7.100
Boat	4.487	4.812	6.087	6.090
Cman	5.004	5.246	6.162	6.472
Einstein	4.868	5.240	6.515	6.813
Elaine	5.193	5.423	7.229	7.423
Goldhill	4.914	5.513	6.581	6.329
Lena	4.485	4.796	7.041	7.215
Man	5.568	5.768	7.218	7.387
Mri	4.776	5.151	6.350	6.284
Peppers	4.752	4.981	7.004	7.125
Zelda	4.185	4.460	6.379	6.485
Average	**4.797**	**5.146**	**6.690**	**6.809**

Result from Table 9-7 show high bit-rate for GCT-G-intensity and GCT-I-error versions, because between gradients and sample value, prediction error and sample values correlation is very small. Correlation is necessary condition for context modeling. Actually, as we can see from Table 9-7 variations of our GCT-G and GCT-I algorithms, i.e. GCT-G-intensity and GCT-I-error methods, have higher bit-rate as results of GCT-G and GCT-I.

10. Conclusions

In this thesis was considered the problem of context modeling with using n-ary context tree structure for lossless compression. Grayscale images represent the area of interests in different types of digital images.

We have considered and studied known lossless compression methods JPEG2000, LOCO-I, CALIC. All of them are based on compression paradigm of “universal modeling and coding”. Being based on this paradigm two lossless compression algorithms have been developed during this work: General Context Tree based on DWT coefficients (GCT-W), General Context Tree based on local gradients (GCT-G). These algorithms are modernization of General Context Tree based on Intensity method (GCT-I).

Were developed and analyzed the different context templates based on DWT coefficients, local gradients and intensity of adjacent pixels. For context modeling the unique approach of n-ary context tree with incomplete and optimal structure was used.

The proposed new methods, GCT-I, GCT-W and GCT-G, were compared with other known lossless compression methods (JPEG2000, LOCO-I, CALIC) and gave comparable results, in general the difference is about 1-2% in compression efficiency. For testing have been used following types of grayscale images: medical, natural smooth, noise and astronomical images.

The empirical results show that on medical images by applying GCT-I method, here we have 7% improvement in compression effectiveness. Also GCT-G algorithm has better compression results about 2% improvement than LOCO-I method on medical and astronomical test sets.

References

[Adams01] M. D. Adams, “The JPEG2000 Still Image Compression Standard”, ISO/IEC JTC 1/SC 29/WG 1N 2412, September 2001.

[AKF05] A. Akimov, A. Kolesnikov, P. Fränti, “Context tree modeling of chain codes for contour compression”, *Scandinavian Conference on Image Analysis (SCIA'05)*, Joensuu, Finland, LNCS vol. 3540, pp. 312-321, June 2005.

[BCW89] T. Bell, J. Cleary, I.H. Witten, “Modelling for text compression”, *ACM Computing Surveys*, vol. 21 (4), pp.557-591, December 1989.

[FA99] P. Fränti, E. Ageenko, “On the use of context tree for binary image compression”, *Proc. 1999 IEEE International Conference on Image Processing "ICIP'99"*, vol. 3, pp. 757-761, Kobe, Japan, 1999.

[Fränti00] P. Fränti, “*Image Compression*”, University of Joensuu, Department of Computer Science, Lecture notes, 2000.

[Golomb66] S.W. Golomb, “Run-length encoding”, *IEEE Transactions on Information Theory*, vol. 12 (3), pp. 399-401, July 1966.

[HV91] P. Howard, J. Vitter, “Analysis of arithmetic coding for data compression”, *Proc. 1991 IEEE Data Compression Conference (DCC '91)*, Snowbird, Utah, pp. 3-12, April 1991.

[Kopyl04] P.Kopylov, “*Processing and Compression of Raster Map Images*”, University of Joensuu, Computer Science, Dissertations 8, Joensuu, 2004.

[Lloyd82] S.P. Lloyd, “Least squares quantization in PCM”, *IEEE Transactions on Information Theory*, vol. 28 (2), pp. 129-137, March 1982.

[Mallat89] S. Mallat, “A Theory of Multiresolution Signal Decomposition: The Wavelet Representation”, *IEEE Transactions on Pattern Analysis and Machine Intelligence*, vol.11 (7), pp.674-693, July 1989.

[Mallat98] S. Mallat, "*A Wavelet Tour of Signal Processing*", Academic Press, 1998.

[Max60] J. Max, "Quantizing for minimum distortion", *IRE Transactions on Information Theory*, vol. 6 (1), pp. 7-12, March 1960.

[MB01] M.W. Marcellin, A. Bilgin, "JPEG2000: Highly Scalable Image Compression", *Proc. 2001 International Conference on Information Technology: Coding and Computing (ITCC2001)*, Las Vegas, Nevada, pp. 268-272, April 2001.

[MBGB00] M.W. Marcellin, A. Bilgin, M.J. Gormish and M.P.Boliek, "An Overview of JPEG-2000", *Proc. 2000 Data Compression Conference*, Snowbird, Utah, pp. 523-541, March 2000.

[MF90] M.W. Marcellin and T.R. Fisher, "Trellis coded quantization of memoryless and Gauss-Markov sources", *IEEE Transactions on Communications*, vol. 38 (1), pp. 82-93, January 1990.

[MF98] B. Martins, S. Forchhammer, "Tree coding of bi-level images", *IEEE Transactions on Image Processing*, vol. 7 (4), pp. 517-528, April 1998.

[MT02] M. W. Marcellin, D. S. Taubman, "*JPEG2000: image compression fundamentals, standards, and practice*", Kluwer Academic Publishers, 2002.

[Norhe94] R. Norhe, "*Topics in descriptive complexity*", PhD Thesis, University of Linköping, Sweden, 1994.

[Ris83] J. Rissanen, "A universal data compression system", *IEEE Transactions on Information Theory*, vol. 29 (5), pp. 656-664, September 1983.

[RL81] J. Rissanen, G. Langdon, "Universal modeling and coding", *IEEE Transactions on Information Theory*, vol. 27 (1), pp. 12-23, January 1981.

[Shan48] C.E. Shannon, "A mathematical theory of communication", *Bell System Technical Journal*, vol. 27 (3), pp. 398-403, July 1948.

[SN96] G. Strang, T. Nguyen, “*Wavelets and Filter Banks*”, Wellesley-Cambridge Press, 1996.

[WM97a] X. Wu, N. Memon, “Context-Based, Adaptive, Lossless Image Coding”, *IEEE Transactions on Communications*, vol. 45 (4), pp. 437-444, April 1997.

[WM97b] X. Wu, N. Memon “Recent Developments in Context-Based Predictive Techniques for Lossless Image Compression”, *The Computer Journal*, vol. 40 (2/3), pp. 127-136, 1997.

[WR95] M. Weinberger, J. Rissanen, “A universal finite memory source”, *IEEE Transactions on Information Theory*, vol. 41 (3), pp. 643-652, May 1995.

[WRA96] M. Weinberg, J. Rissanen and R. Arps, “Applications of universal context modeling to lossless compression of gray-scale images”, *IEEE Transactions on Image Processing*, vol. 5 (4), pp. 575-586, April 1996.

[WSG00] M. Weinberger, G. Seroussi, B. Carpentieri, “Lossless Compression of Continuous-Tone Images”, Hewlett-Packard Laboratories technical report HPL-2000-163, vol. 29, December 2000.

[WSS96] M. Weinberger, G. Seroussi, G. Sapiro, “LOCO-I: A Low Complexity, Context-Based, Lossless Image Compression Algorithm”, Tech. Rep. Hewlett-Packard Laboratories, Palo Alto, CA 94304, vol. 10, April 1996.

[WSS00] M. Weinberger, G. Seroussi, G. Sapiro, “The LOCO-I Lossless Image Compression Algorithm: Principles and Standardization into JPEG-LS”, *IEEE Transactions on Image Processing*, vol. 9 (8), pp. 1309-1324, August 2000.

Appendix 1

Results are given in bit-rate and evaluated in bits per pixel. Initial bit-rate for all images is 8 bits per pixel.

Table 1. Bit-rate of compression results on natural smooth images (bits per pixel).

File name	CALIC	LOCO-I	JPEG2000	GCT-G	GCT-W	GCT-I
Barb	4.656	4.863	4.781	5.196	5.069	5.751
Billsface	3.800	3.914	4.021	4.131	4.185	4.606
Boat	4.156	4.250	4.406	4.487	4.607	4.812
Cman	4.650	4.740	5.000	5.004	5.254	5.246
Einstein	4.525	4.602	4.813	4.868	5.060	5.240
Elaine	4.813	4.898	4.938	5.193	5.147	5.423
Goldhill	4.625	4.712	4.844	4.914	4.987	5.513
Lena	4.094	4.237	4.313	4.485	4.490	4.796
Man	5.125	5.267	5.500	5.568	5.669	5.768
Mri	4.375	4.518	4.600	4.776	4.885	5.151
Peppers	4.375	4.489	4.625	4.752	4.789	4.981
Zelda	3.875	4.005	4.000	4.185	4.150	4.460
Average	**4.422**	4.541	4.653	4.797	4.858	5.146

Table 2. Bit-rate of compression results on medical images (bits per pixel).

File name	CALIC	LOCO-I	JPEG2000	GCT-G	GCT-W	GCT-I
Hip	1.839	1.887	2.611	1.938	2.638	1.948
Child	3.634	3.683	3.830	3.772	3.714	3.599
5week	3.223	3.325	3.576	3.273	4.033	3.292
Blood	3.806	4.076	4.682	3.464	4.862	2.447
Ultrasound	4.573	4.679	4.886	5.015	5.192	5.019
Average	3.415	3.530	3.917	3.492	4.088	**3.261**

Table 3. Bit-rate of compression results on natural noise images (bits per pixel).

File name	CALIC	LOCO-I	JPEG2000	GCT-G	GCT-W	GCT-I
Face	4.183	4.315	4.322	4.528	4.565	5.511
Fog	3.382	3.467	3.513	3.577	3.624	4.022
Girl	3.869	3.949	4.018	4.173	4.263	5.245
Port	6.541	6.682	6.780	7.111	6.843	7.079
Susy	5.196	5.288	5.403	5.585	5.474	5.932
Average	**4.634**	4.740	4.807	4.995	4.954	5.558

Appendix 2

Here are results of experiment for detecting the optimal value for the number of DWT levels and the size of context s in GCT-W algorithm. Results are given in bit-rate and evaluated in bits per pixel. Initial bit-rate for all images is 8 bits per pixel. For experiment we used two sets of images: smooth natural images, which were used in developing the existing ISO/IEC 10918-1 lossless JPEG standard (see Appendix 4), and medical images (see Appendix 5).

	DWT 3 levels				**DWT 4 levels**			
File name	$s = 2$	$s = 3$	$s = 4$	$s = 5$	$s = 2$	$s = 3$	$s = 4$	$s = 5$
Barb	5.066	5.068	5.068	5.068	5.076	5.078	5.078	5.078
Billsface	4.205	4.205	4.205	4.205	4.199	4.199	4.199	4.199
Boat	4.621	4.621	4.621	4.621	4.607	4.607	4.607	4.607
Cman	5.247	5.251	5.251	5.251	5.253	5.253	5.253	5.253
Einstein	5.088	5.088	5.088	5.088	5.066	5.066	5.066	5.066
Elaine	5.154	5.155	5.155	5.155	5.144	5.143	5.144	5.144
Goldhill	4.984	4.985	4.985	4.985	4.980	4.980	4.980	4.980
Lena	4.508	4.507	4.508	4.509	4.493	4.494	4.494	4.495
Man	5.663	5.663	5.663	5.663	5.687	5.688	5.688	5.688
Mri	4.833	4.833	4.833	4.833	4.889	4.889	4.889	4.889
Peppers	4.806	4.806	4.806	4.806	4.787	4.786	4.786	4.786
Zelda	4.159	4.159	4.159	4.159	4.145	4.146	4.146	4.146
Average	4.861	**4.862**	4.862	4.862	4.861	**4.861**	4.861	4.861

	DWT 5 levels				**DWT 6 levels**			
File name	$s = 2$	$s = 3$	$s = 4$	$s = 5$	$s = 2$	$s = 3$	$s = 4$	$s = 5$
Barb	5.066	5.074	5.074	5.074	5.063	5.071	5.071	5.071
Billsface	4.194	4.194	4.194	4.194	4.183	4.183	4.183	4.183
Boat	4.621	4.621	4.621	4.621	4.623	4.623	4.623	4.623
Cman	5.261	5.261	5.261	5.261	5.257	5.257	5.257	5.257
Einstein	5.069	5.069	5.069	5.069	5.069	5.069	5.069	5.069
Elaine	5.142	5.143	5.141	5.141	5.146	5.146	5.144	5.144
Goldhill	4.992	4.992	4.992	4.992	4.995	4.995	4.995	4.995
Lena	4.491	4.493	4.492	4.492	4.500	4.498	4.498	4.498
Man	5.686	5.687	5.687	5.687	5.675	5.673	5.673	5.673
Mri	4.905	4.905	4.905	4.905	4.895	4.895	4.895	4.895
Peppers	4.800	4.799	4.799	4.799	4.791	4.792	4.792	4.792
Zelda	4.146	4.146	4.146	4.146	4.145	4.143	4.143	4.143
Average	4.864	**4.865**	4.865	4.865	4.862	**4.862**	4.862	4.862

	DWT 7 levels				DWT 8 levels			
File name	*s* = 2	*s* = 3	*s* = 4	*s* = 5	*s* = 2	*s* = 3	*s* = 4	*s* = 5
Barb	5.073	5.083	5.083	5.083	5.059	5.069	5.069	5.069
Billsface	4.186	4.186	4.186	4.186	4.185	4.185	4.185	4.185
Boat	4.616	4.615	4.615	4.615	4.609	4.607	4.607	4.607
Cman	5.255	5.258	5.258	5.258	5.255	5.254	5.254	5.254
Einstein	5.063	5.063	5.063	5.063	5.060	5.060	5.060	5.060
Elaine	5.149	5.146	5.148	5.148	5.147	5.147	5.146	5.146
Goldhill	4.989	4.990	4.990	4.990	4.987	4.987	4.987	4.987
Lena	4.488	4.489	4.492	4.489	4.489	4.490	4.490	4.490
Man	5.669	5.668	5.668	5.668	5.672	5.669	5.669	5.669
Mri	4.885	4.885	4.885	4.885	4.885	4.885	4.885	4.885
Peppers	4.796	4.793	4.793	4.793	4.787	4.789	4.789	4.789
Zelda	4.150	4.149	4.149	4.149	4.150	4.150	4.150	4.150
Average	4.860	**4.860**	4.861	4.861	4.857	**4.858**	4.858	4.858

	DWT 3 levels				DWT 4 levels			
File name	*s* = 2	*s* = 3	*s* = 4	*s* = 5	*s* = 2	*s* = 3	*s* = 4	*s* = 5
Hip	2.657	2.652	2.644	2.645	2.654	2.645	2.639	2.640
Child	3.723	3.721	3.718	3.718	3.728	3.722	3.722	3.724
5week	4.033	4.032	4.032	4.032	4.048	4.048	4.048	4.048
Blood	4.857	4.857	4.857	4.857	4.854	4.854	4.854	4.854
Ultrasound	5.260	5.260	5.260	5.260	5.226	5.226	5.226	5.226
Average	4.106	4.104	**4.102**	4.102	4.102	4.099	4.099	**4.098**

	DWT 5 levels				DWT 6 levels			
File name	*s* = 2	*s* = 3	*s* = 4	*s* = 5	*s* = 2	*s* = 3	*s* = 4	*s* = 5
Hip	2.651	2.645	2.638	2.648	2.649	2.651	2.633	2.633
Child	3.718	3.718	3.718	3.715	3.722	3.722	3.722	3.722
5week	4.034	4.035	4.035	4.035	4.033	4.033	4.033	4.033
Blood	4.853	4.853	4.853	4.853	4.861	4.861	4.861	4.861
Ultrasound	5.210	5.210	5.210	5.210	5.198	5.198	5.198	5.198
Average	4.093	**4.092**	4.092	4.090	4.093	4.093	**4.091**	4.091

	DWT 7 levels				DWT 8 levels			
File name	*s* = 2	*s* = 3	*s* = 4	*s* = 5	*s* = 2	*s* = 3	*s* = 4	*s* = 5
Hip	2.654	2.648	2.641	2.641	2.652	2.655	2.638	2.640
Child	3.714	3.718	3.715	3.714	3.717	3.718	3.716	3.716
5week	4.033	4.031	4.031	4.033	4.033	4.031	4.031	4.031
Blood	4.862	4.862	4.862	4.862	4.862	4.862	4.862	4.862
Ultrasound	5.192	5.192	5.192	5.192	5.192	5.192	5.192	5.192
Average	4.091	4.090	4.089	**4.088**	4.091	4.090	**4.088**	4.088

Appendix 3

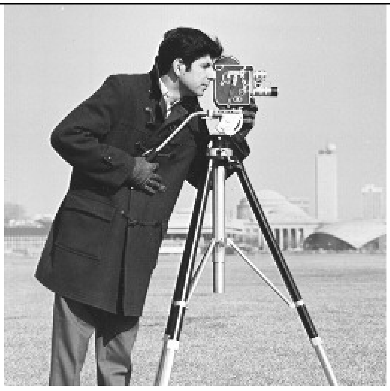
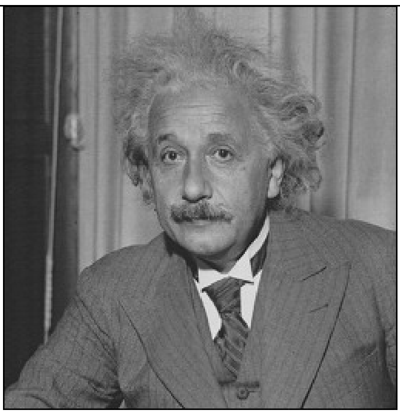
Results of experiment based on noise images (see Appendix 6). Results are given in bit-rate and evaluated in bits per pixel. Initial bit-rate for all images is 8 bits per pixel.

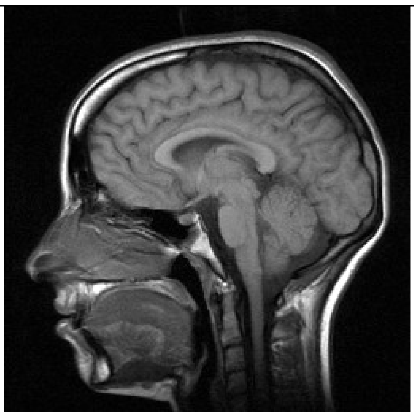
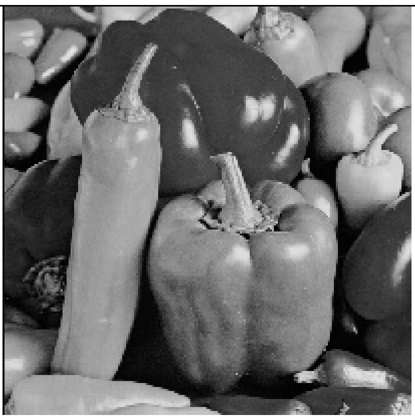
Table 1. Bit-rate of compression results on natural noise images (bits per pixel)

File name	**CALIC**	**LOCO-I**	**JPEG2000**	**GCT-G**	**GCT-W**	**GCT-I**
Face	4.183	4.315	4.322	4.528	4.565	5.511
Fog	3.382	3.467	3.513	3.577	3.624	4.022
Girl	3.869	3.949	4.018	4.173	4.263	5.245
Port	6.541	6.682	6.780	7.111	6.843	7.079
Susy	5.196	5.288	5.403	5.585	5.474	5.932
Average	**4.634**	4.740	4.807	4.995	4.954	5.558

Appendix 4

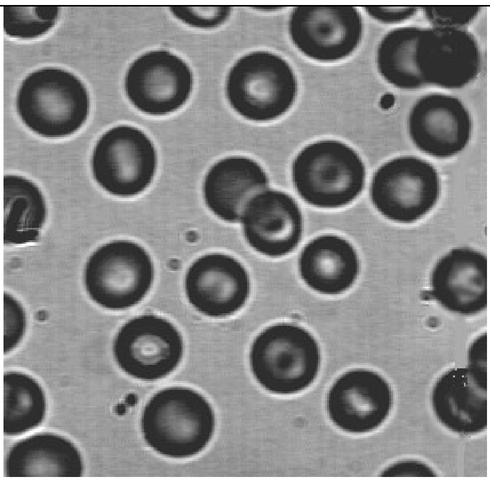
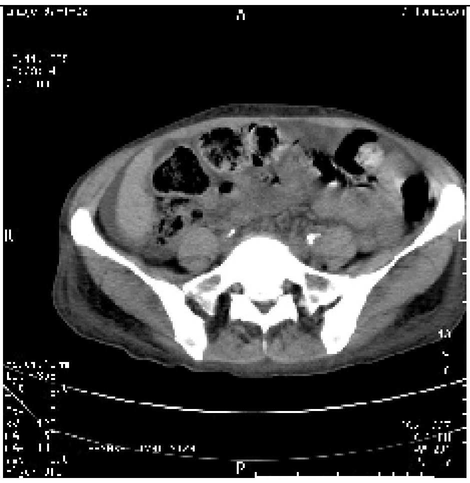
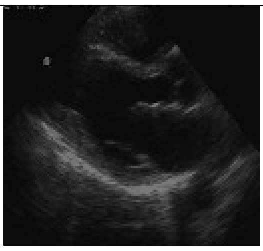
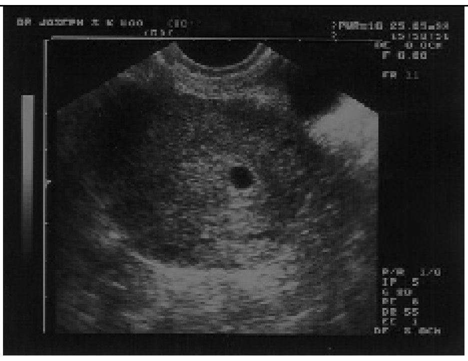
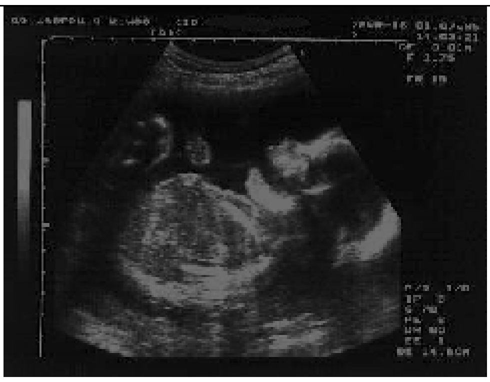
Original JPEG image test set. Set of natural smooth grayscale images.

Barb.pgm, size 512x512, 256col.	Billsface.pgm, size 309x240, 256col.
Boat.pgm, size 512x512, 256col.	 Cman.pgm, size 256x256, 256col.
 Einstein.pgm, size 256x256, 256col.	Elaine.pgm, size 512x512, 256col.

Goldhill.pgm, size 512x512, 256col.	Lena.pgm, size 512x512, 256col.
Man.pgm, size 512x512, 256col.	 Mri.pgm, size 256x256, 256col.
 Peppers.pgm, size 512x512, 256col.	Zelda.pgm, size 512x512, 256col.

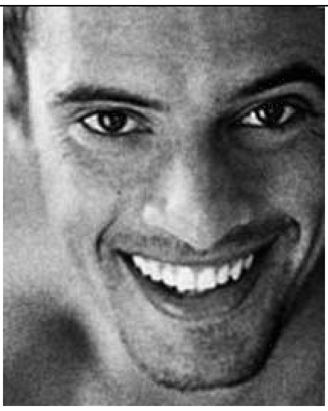
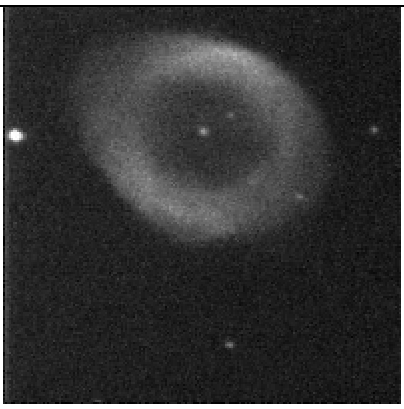
Appendix 5

Set of medical images.

 Blood.pgm, size 272x265, 256col.	 Hip.pgm, size 512x512, 256col.
 Ultrasound.pgm, size 128x120, 256 col.	 5week.pgm, size 356x270, 256col.
 Child.pgm, size 378x290, 256 col.	

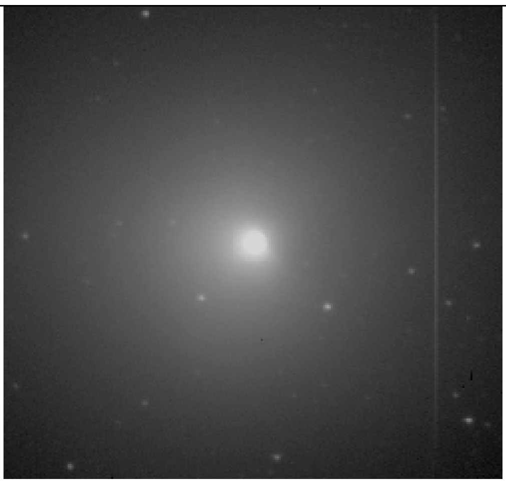
Appendix 6

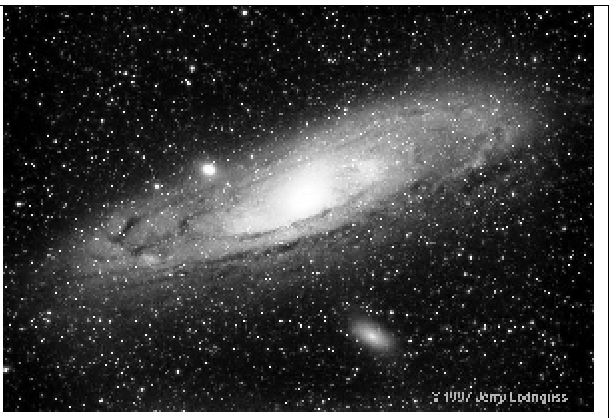
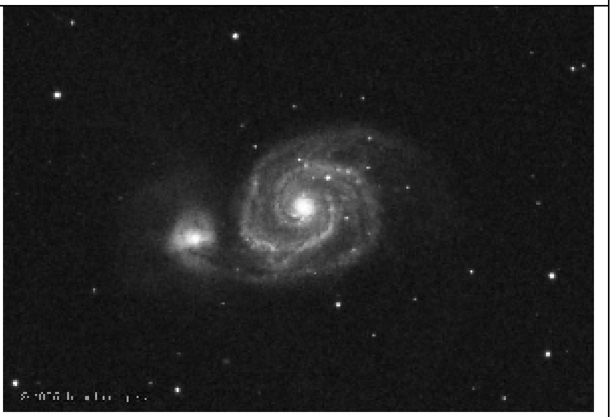
Test set of natural noise images.

 Face.pgm, size 211x264, 256col.	 Fog.pgm, size 512x512, 256col.
Girl.pgm, size 204x264, 256col.	Port.pgm, size 320x408, 256col.
Susy.pgm, size 352x288, 256col.	

Appendix 7

Test set of astronomical images.

Astr1.pgm, size 827x552, 256col.	Asrt2.pgm, size 576x396, 256col.
 Astr3.pgm, size 512x512, 256col.	Asrt4.pgm, size 524x364, 256col.
Astr5.pgm, size 827x552, 256col.	Astr6.pgm, size 827x552, 256col.

Astr7.pgm, size 576x386, 256col.	 Astr8.pgm, size 576x396, 256col.
Astr9.pgm, size 827x552, 256col.	 Astr10.pgm, size 576x396, 256col.