The perceptual impact of different quantization schemes in G.719

BOTE LIU

Master’s Degree Project
Stockholm, Sweden May 2013

XR-EE-SIP 2013:001
Abstract

In this thesis, three kinds of quantization schemes, Fast Lattice Vector Quantization (FLVQ), Pyramidal Vector Quantization (PVQ) and Scalar Quantization (SQ) are studied in the framework of audio codec G.719. FLVQ is composed of an $RE_8$-based low-rate lattice vector quantizer and a $D_8$-based high-rate lattice vector quantizer. PVQ uses pyramidal points in multi-dimensional space and is very suitable for the compression of Laplacian-like sources generated from transform. SQ scheme applies a combination of uniform SQ and entropy coding. Subjective tests of these three versions of audio codecs show that FLVQ and PVQ versions of audio codecs are both better than SQ version for music signals and SQ version of audio codec performs well on speech signals, especially for male speakers.
Acknowledgements

I would like to express my sincere gratitude to Ericsson Research, which provides me with such a good thesis work to do.

I am indebted to my supervisor, Sebastian Näslund, for sparing time to communicate with me about my work every week and giving me many valuable suggestions. I am also very grateful to Volodya Grancharov and Eric Norvell for their advice and patience as well as consistent encouragement throughout the thesis. My thanks are extended to some other Ericsson researchers for attending the subjective listening evaluation in the thesis.

Finally, I want to thank my examiner, Professor Arne Leijon of Royal Institute of Technology (KTH) for reviewing my report very carefully and supporting my work very much.
# Contents

1 Introduction.................................................................................................................. 1
   1.1 A review of audio and speech coding................................................................. 1
   1.2 Objective............................................................................................................... 2
   1.3 Thesis outline ..................................................................................................... 3
2 Background.................................................................................................................. 4
   2.1 Audio and speech coding..................................................................................... 4
      2.1.1 Waveform coding......................................................................................... 4
      2.1.2 Parametric coding....................................................................................... 5
      2.1.3 Perceptual model......................................................................................... 6
   2.2 Quantization......................................................................................................... 7
      2.2.1 Scalar quantization..................................................................................... 7
      2.2.2 Vector quantization.................................................................................... 7
   2.3 Entropy coding..................................................................................................... 8
      2.3.1 Huffman coding......................................................................................... 8
      2.3.2 Range coding............................................................................................ 8
   2.4 Performance evaluation....................................................................................... 11
3 G.719 .......................................................................................................................... 12
   3.1 System description............................................................................................... 12
      3.1.1 Encoder overview ..................................................................................... 12
      3.1.2 Decoder overview .................................................................................... 13
   3.2 Encoder................................................................................................................ 13
      3.2.1 Transient detection..................................................................................... 13
      3.2.2 Adaptive time-frequency transform......................................................... 14
      3.2.3 Grouping of spectral coefficients.............................................................. 14
      3.2.4 Norm quantization.................................................................................... 15
      3.2.5 Bit-allocation............................................................................................. 17
      3.2.6 Shape quantization................................................................................... 18
      3.2.7 Noise level adjustment............................................................................. 18
   3.3 Decoder.................................................................................................................. 19
      3.3.1 Norm decoding......................................................................................... 19
      3.3.2 Spectral coefficient decoding................................................................. 19
3.3.3 Spectrum filling........................................................................................................19
3.3.4 Noise level adjustment..........................................................................................20
3.3.5 De-normalization .................................................................................................20
3.3.6 Inverse transform ...............................................................................................20
4 Quantization theory ....................................................................................................21
  4.1 Fast lattice vector quantization .............................................................................21
    4.1.1 Lattice .............................................................................................................21
    4.1.2 FLVQ encoder .................................................................................................22
    4.1.3 FLVQ decoder .................................................................................................24
  4.2 Pyramidal vector quantization ..............................................................................24
    4.2.1 Mapping between bits and pulses ..................................................................25
    4.2.2 Calculation of the size of the codebook .......................................................25
    4.2.3 PVQ Encoder .................................................................................................25
    4.2.4 PVQ Decoder .................................................................................................26
    4.2.5 Split ................................................................................................................27
5 Implementation ..........................................................................................................28
  5.1 Bit-stream format ....................................................................................................28
  5.2 G.719 with scalar quantization ............................................................................28
    5.2.1 Creation of a codebook ..................................................................................29
    5.2.2 Entropy coding implementation .....................................................................32
  5.3 G.719 with PVQ ....................................................................................................32
    5.3.1 Quantization of norms ...................................................................................32
    5.3.2 Bit-pulse conversion and pre-calculated table .............................................33
    5.3.3 Range coding .................................................................................................34
    5.3.4 Bit saved by range coding .............................................................................36
  5.4 Bit-allocation schemes .........................................................................................36
6 Results .......................................................................................................................38
  6.1 Objective evaluations ............................................................................................38
    6.1.1 Fixed bit-allocation coding .........................................................................38
    6.1.2 Flexible bit-allocation coding .................................................................39
  6.2 Subjective evaluations .........................................................................................40
    6.2.1 Original bit-allocation scheme ..................................................................41
1 Introduction

1.1 A review of audio and speech coding

In the last 20 years, as the rapid development of the public networks, like Internet, a great amount of digital data flowed over the networks unprecedentedly. Standards like World Wide Web, HTTP, HTML and FTP promoted this trend by providing convenient ways to access or download data from the public servers. In addition, because of the prevalence of the smartphones, vast amount of data is accessed or downloaded through wireless channels, whose transmission capacity is expected to expand and cost is quite expensive. One way to reduce data congestion in the communication channels is data compression, which can decrease the size of the data and at the same time acquire the same or high-fidelity quality of the original one.

Audio coding is one important branch of data compression, and is used to acquire efficient digital representation of audio signals. The bit-stream obtained from an audio encoder has a smaller size to be transmitted or stored than the original one, but the quality of the decoded signals is still acceptable.

In 1980’s the compact disk (CD) became popular because of its unprecedented high-fidelity. But the advantage came at the cost in high data rates. A general CD at that time was typically sampled at 44.1 kHz using pulse code modulation (PCM), and each sample was represented by 16 bits. This expensive mechanism leads to the data rate of 705.6 kbps for a monaural channel, and 1.41Mbps for a stereo respectively. Even if the applications of high data rates seem to be successful as the popularity of the CD and the digital audio tape (DAT), they are not tolerable any more considering the constraints of the bandwidth of the Internet or wireless networks. Therefore new multimedia digital applications have to reduce their data rates at the smallest cost of the audio quality, and this trend motivates considerable research and standardization work on the audio coding that has a reasonable trade-off between the quality and the corresponding cost of data rates.

The Moving Picture Expert Group (MPEG) is such an organization that developed a series of standards for audio and video compression and transmission. MPEG-1 describes the compression of audio signals and includes three kinds of coding schemes, called Layer -1, -2, -3, with increasing algorithm complexity and performance. MPEG-1 Layer -3 (MP3) is widely applied to encode audio signals and this format of audio files has been dominating the world of music. MPEG-2 defines a new audio coding method, which improves MPEG-1 by allowing the coding of audio signals with more than two channels and is back-compatible. MPEG-2 also specifies a non-back-compatible audio coding, called Advanced Audio Coding (AAC), which offers higher compression rates than the previous MPEG standards and is less complicated than MP3 to some extent. There are several new audio coding techniques introduced in MPEG-4, like Code-Excited Linear Prediction (CELP), Advanced Audio Coding (AAC), and MPEG-4 Scalable to Lossless (SLS, lossless audio coding), etc.

The work of speech coding research was started by Homer Dudley [1], of the Bell Telephone Laboratories. The research field of his work is the transmission of speech signals on the narrow bandwidth channel. Dudley proposed the first analysis-by-synthesis speech coding method, which analyzed the characteristics of speech signals and reconstructed the signals by exciting a
bank of periodic or random excitation signals in the decoder. The channel vocoder attracted much attention at its emergence due to its fine capacity to transmit encrypted speech. The work of formant pattern matching vocoder and the improved version of the channel vocoder were published [29] in the 1950’s and 1960’s.

The previous period of the research on analog speech coding is very meaningful, but the digital representation rapidly gained attention because of its advantages on communication safety, high-fidelity quality and the reduction of storage space. Pulse Code Modulation (PCM) is the simplest digital speech coding method, which is a combination of digital sampling and scalar quantization, but it does not exploit any redundancy in the signals. Some modified versions of PCM, like Delta Modulation (DM) [2], Differential PCM (DPCM) and Adaptive DPCM (ADPCM), can analyze the speech samples of each frame and remove the redundancy between them.

A more complicated speech representation based on Autoregressive (AR) model, in which the vocal tract is modeled as an all-pole filter, became a hot topic of the research on the digital speech coding. Itakura and Saito [3], and Atal and Schroeder [4] were the pioneers to research on speech with Linear Prediction (LP) method. An analysis-by-synthesis coding system based on LP was proposed by Atal and Hanauer [5], and Markel and Gray [6] did much work on the linear predictive coding (LPC). Another method, Homomorphic analysis, was also a good one to analyze the characteristics of the speech samples, and this technique was strongly supported by Oppenheim and Schafer [7].

As the development of the theory of digital signal processing, more improved schemes of speech coding appeared. Flanagan and Golden applied Short-Time Fourier Transform (STFT) to analyze speech signals in an analysis-synthesis system [8] and Portnoff [9] gave a theoretical support for the analysis with STFT. The research on linear prediction, transform coding and sub-band coding was continuously active in the mid to late 1970’s.

In the 1980’s and 1990’s, because of the demand for the narrow-band and secure communication in cellular, more efforts were placed on the low-rate speech coding for high-quality communication applications. Some important methods proposed in the 1980’s were: sinusoidal analysis-by-synthesis speech coder by McAulay and Quatieri [10], multiband excitation vocoder by Griffin and Lim [11], and vector quantization by Gersho and Gray [12]. Then a very famous speech coder, Code Excited Linear Prediction (CELP) [13], was proposed by Atal and Schroeder. The CELP coder transmits not only the LPC parameters but also the residuals of the waveform, and is a low-rate coder with relatively good communication quality.

In the late 1990’s and 2000’s more attention was attracted by the low-rate algorithm for the application of mobile cellular. For example, speech coders of Adaptive Multi-Rate (AMR) [14] and Adaptive Multi-Rate Wideband (AMR-WB) [15] were chosen as a standard in 3GPP and were deployed in GSM and UMTS. In the 2009, the speech codec SILK was released and applied in the software of Skype.

1.2 Objective

The goal of the thesis is to study the perceptual impact of different quantization schemes in the context of a MDCT (Modified Discrete Cosine Transform)-based audio codec, G.719 [34]. The quantizers under study are Fast Lattice Vector Quantization (FLVQ) [34], Pyramidal Vector Quantization (PVQ) [35] and Scalar Quantization (SQ).
FLVQ is the quantization scheme that G.719 is applying to. It is an 8-dimensional lattice vector quantizer consisting of two sub-quantizers for low and high bit-rates respectively.

PVQ is a fast and efficient approach to quantize Laplace distributed samples such as the normalized spectral coefficients from an audio or video compression system. PVQ has simple encoding and decoding procedures and need not store a large amount of codeword in a codebook. A PVQ codeword can be encoded and decoded by the computation of its position among all codewords in the corresponding PVQ codebook. The quantizer is then followed by range coding to remove the redundancy among the indices of PVQ outputs.

SQ is also selected as a quantization scheme in this thesis considering that it is the simplest method to quantize a data source of real numbers.

The results of the thesis are presented in three aspects:

Firstly, two quantizers being studied, FLVQ and SQ, are compared with un-constrained vector quantizer (UCVQ), an optimal vector quantization scheme, to see whether there is enough margin for these two quantizers to improve. This comparison is the starting point of the main work. If the margin is large enough, we have valid motivation to improve or replace the existing quantization schemes in the framework of G.719. An objective criterion, Segmental Signal-to-Noise Ratio (SEGSNR), is used to measure the performance of the different quantizers when quantizing normalized MDCT coefficients with fixed bit-rates.

Secondly, the performances of G.719 to which three quantization schemes under study are applied, are evaluated by the same objective criterion. This comparison can show to what extent the quantizers are matched to the other modules in the framework of G.719.

Finally, the modified versions of G.719 systems mentioned in the second aspect are evaluated with music and speech clips by MUSHRA (Multiple Stimuli with Hidden Reference and Anchor), which is one of the most common subjective evaluation methods for perceived quality of outputs from audio codecs.

1.3 Thesis outline

This report is organized as follows. In the second chapter, some background or related knowledge, like quantization, entropy coding and performance evaluation methods, is introduced.

The third chapter gives a detailed description of a full-band high-quality audio and speech coding standard, ITU-T G.719, which is the framework for studying quantization schemes in this thesis.

Chapter 4 provides more detailed theory about the quantization schemes studied in this thesis.

The implementation details of new quantization schemes in the G.719 framework are illustrated in the chapter 5.

The results from objective and subjective tests are presented in Chapter 6, and the last two chapters are discussion and conclusion.
2 Background

2.1 Audio and speech coding

An audio codec is used to reduce transmission bandwidth and storage space of audio signals and can be implemented in a lossy or lossless way. The decoded signal from a lossless audio codec is exactly the same as the input one, and the compression rate is around 50%-60% of the original size [16]. Most of the data or signals in nature have statistical redundancy, and the lossless compressor can encode them without losing information by removing the redundancy. Lossy audio codecs have a higher compression rate and are also widely applied in practice. Besides statistical redundancy, lossy audio codecs also consider perceptual masking of human ear and remove the spectrum that is not easily perceived by people. The outputs of a lossless and a lossy audio codec may be quite different in some frequency bands, but the difference is very hard to hear at high bit-rate.

The audio coding methods can also be classified based on the types of data transmitted in the channel. Some audio signals are coded based on their waveform, while others are processed to extract some parameters representing the properties of the signal. Therefore two kinds of methods, waveform coding and parametric coding, are designed based on this idea.

2.1.1 Waveform coding

Pulse-Code Modulation (PCM) is the simplest version of waveform coding. Uniform PCM is similar to an AD converter and quantizes samples with scalar quantizer of a constant step-size. PCM does not exploit the statistical redundancy across samples, so its compression ratio is the worst compared to other quantizers. Non-uniform PCM has different quantization resolution and can adjust its quantizing step-size based on the statistical model of samples. If many samples are located in one range, then the samples in that range will be quantized with high precision, and vice versa. Non-uniform PCM can also work in the logarithmic domain and two standard companding algorithms, A-law and μ-law (well described in [17]), are based on this method. Adaptive PCM (APCM) is a variant of PCM and it may adjust its quantizing step-size based on the dynamic range variations of samples. The quantizing step-size is estimated from the previous samples and is transmitted to the decoder as side information in the adaptive PCM system with forward estimation. A more useful quantization is differential PCM (DPCM), which lowers the correlation of samples by predicting one sample with the previous ones. The simplest version of DPCM is the one with a first-order predictor which predicts a sample with only the previous one. In practice, DPCM usually makes the dynamic range of the residual signal reduced when compared with that of the input signal. Another kind of coding method integrated with the characteristics of APCM and DPCM is called ADPCM [18], [19] which predicts and quantizes adaptively based on the real-time statistics of speech signal. The prediction parameters can be obtained through open-loop or closed-loop adaption. In the open-loop adaption, the parameters of prediction are acquired from the current speech signal and have to be transmitted to the decoder to recover the signal. In the closed-loop, the prediction parameters are obtained from the previous
encoded speech samples and the decoder can also estimate them itself. Therefore the parameters are not needed to transmit in the closed-loop adaption in ADPCM.

The sub-band coder (SBC) [20], [21] and transform coder [22], [23] use transforms to convert signal from time-domain to frequency-domain, and exploit the redundancy in the transform domain. SBC acquires frequency bands of one signal by applying a bank of band-pass filters on it. Then the samples from the filters are sub-sampled and encoded. In the decoder, the received bit-stream is decoded, up-sampled and processed by the synthesis filter bank to recover the original signal. The signal can be perfectly reconstructed in the decoder if the distortion of quantization is ignored. SBC has a merit that it can exploit a perceptual model to allocate bits to different sub-bands in order to increase the efficiency of bits used. Besides, SBC can have different resolution on sub-bands and in speech coding low frequency sub-bands are given high resolution in order to keep key information like pitch period and formants.

Transform coders can efficiently compress data based on the fact that the unitary transform [24] decorrelates the input samples. Unitary transforms do not change the length of an \( N \)-dimensional input vector, actually it just rotates the vector in the \( N \)-dimensional vector space. An appropriate angle of rotation is selected to make the vector of samples uncorrelated to the greatest extent. Therefore the transform coder can reduce bit-rate by removing the redundancy of samples before quantization. There are some common transforms [24] in the transform coder, like the Karhunen-Loève Transform (KLT), the Discrete Fourier Transform (DFT), the Discrete Cosine Transform (DCT), etc. KLT is optimal unitary transform [25], [26] to de-correlate a source of samples. The transform matrix of KLT is composed of eigenvectors of the autocorrelation matrix of the source signal and therefore KLT is a source-dependent unitary transform. The transform parameters of DFT and DCT are independent of the source signal and can be calculated efficiently using a Fast Fourier Transform (FFT) [27], [28]. The performance of DCT is inferior to the optimal KLT, but is better than that of DFT. The decorrelating capacity of DFT approaches that of KLT and DCT as the size of the vector to be compressed increases [26].

### 2.1.2 Parametric coding

Parametric coding methods for speech coding are called vocoders from “voice encoder”. These coders can achieve very low bit-rate and at the same time provide an acceptable quality of speech reconstruction. The reason that vocoders have so high compression ratio is that the speech coder does not recover the waveform of the original signal but transmit some parameters representing the characteristics of a speech frame. The decoder then generates the corresponding output speech signal based on these parameters. The reconstruction signal sounds quite artificial because the residual, the details of the speech signal, is not sent to the decoder. Therefore when the bit-rate is higher than some value the quality of the decoded signal will not improve as the bit-rate increases. There is an analysis procedure for extracting speech parameters of a frame in the encoder and a synthesis procedure to reconstruct signal in the decoder. These speech parameters possibly include pitch period, gain, voiced/unvoiced switch, filter parameters, etc. If one frame is flagged as voiced, a sequence of periodic impulses is used as the input of the synthesis procedure. Otherwise, the decoder adopts a noise signal as the excitation signal.

The elementary vocoder is called channel vocoder [29], which has some channels to transmit quantized gains of different frequency bands to the decoder. After band-pass filter the gains of each frequency band are estimated by calculating the short-term energy of the corresponding band. Considering the relatively slow change of the envelope, the parameters of gains are sub-
sampled before transmitting. In the decoder, the reconstruction signal can be obtained by shaping the locally generated excitation signal with the quantized gains of frequency bands in the encoder. Another vocoder variant is the formant vocoder [30]. The difference of formant vocoder from channel vocoder is that the former estimates the spectral envelope by the formant frequencies and formant bandwidths. The formant vocoder may suffer from huge performance degradation if two formants stay too close. The third variant is linear predictive coding (LPC) vocoder, one of the most popular speech codec, based on the all-pole model. Like DPCM, LPC also has a linear predictor. However LPC does not transmit the residual of the input and the predictive values, but the parameters representing the characteristics of the frame. The reconstruction signal is created with these parameters and the locally generated excitation signal in the decoder. One of the applications of the vocoder is LPC-10, [31] which is mainly used for encrypted transmission in private analog telephone networks.

2.1.3 Perceptual model

The perceptual model [32] of human hearing is also an important topic in audio signal compression and related knowledge is used to remove the redundancy in this domain. The human audibility is determined by the frequency and sound pressure of the sound, [32], see Figure 1. To achieve a high compression rate, signal in the frequency bands which are not easy to perceive may be removed to save bits for transmission. The most sensitive frequency band is between 3 and 4 kHz because of the resonance of the outer ear canal. Then the bits allocated for each frequency band are adjusted considering the perceptual influence of human hearing. The frequency bands that are very sensitive to human ear are supposed to get more bits assigned. However, if the energy of signal in one frequency band is not high enough to make the sound audible, the samples of the signal may not be quantized at all. Another important factor of the psychoacoustic models is the masking effect: a signal may be masked by another adjacent high energy signal in temporal or frequency domain.

![Figure 1 - Hearing area. Adapted from Zwicker [32].](image)
2.2 Quantization

2.2.1 Scalar quantization

Scalar quantization (SQ) is to quantize a single number to the nearest quantization level from a pre-determined finite set of numerical values. Precisely, a scalar quantizer can be seen as a mapping \( P : R \rightarrow C \), where \( R \) is the set of real numbers and

\[
C = \{ y_1, y_2, \ldots, y_N \} \subset R
\]

is the output set of \( N \) pre-determined quantization levels.

Scalar quantization is the simplest form of quantization and SQ quantizes one sample each time. This leads to one disadvantage that SQ cannot exploit the statistical redundancy among samples to quantize. This problem can be solved using a more complex quantization method, vector quantization that will be introduced in the next section.

2.2.2 Vector quantization

The vector quantization is a mapping from an \( M \)-dimensional vector space \( R^M \) to a finite set \( C \) of reproduction points [40]:

\[
P : R^M \rightarrow C
\]

where \( C = \{ y_1, y_2, \ldots, y_N \} \) and \( y_i \in R^M \), \( i \in \{1, 2, \ldots, N\} \). The finite set \( C \) is called codebook, and each element of \( C \) is an \( M \)-dimensional vector in \( R^M \).

Equivalently, vector quantization can also be seen as the procedure that \( M \) values forming an \( M \)-dimensional vector \( x = (x_1, x_2, \ldots, x_M) \) are allocated to one of \( N \) pre-determined \( M \)-dimensional quantization cells, and are quantized as the corresponding representation vector \( \hat{x}_i = (\hat{x}_{i,1}, \hat{x}_{i,2}, \ldots, \hat{x}_{i,M}) \) [40]. All representation vectors are indexed from 1 to \( N \) and are assigned to the same or different sizes of the \( M \)-dimensional space. Figure 2 shows a 2-dimensional vector quantization with uniform and non-uniform resolution [41].

For a given codebook, vector quantization replaces a \( M \)-dimensional vector \( x \) with its nearest representation vector \( \hat{x}_{\text{near}} \), and the index of the representation vector is determined by

\[
j = \arg \min_i d(x, \hat{x}_i)
\]

where \( d \) is the distance between its two parameters.
2.3 Entropy coding

2.3.1 Huffman coding

Huffman coding [42] is one kind of commonly used entropy encoding algorithms for lossless compression. A source symbol encoded by this algorithm will use a variable length code table which is derived based on the statistics probability of each possible element in the source symbol.

A specific approach to choose the representation for each symbol is adopted in Huffman coding, leading to a prefix code in which one codeword representing for a symbol is never a prefix of the one representing any other symbol. The idea of Huffman coding is that the symbols with the larger probability of occurrence are supposed to use shorter codeword than those that does not commonly appear.

Huffman coding algorithm is optimal for a source consisting of unrelated symbols with a known probability distribution. But if the symbols in the stream are correlated, or the probability mass function of symbols in the stream is not known or not identically distributed, Huffman coding is not optimal any more.

However, some improved Huffman coding methods can overcome these restrictions in an adaptive way and it can still have a good performance facing unknown or context-dependent symbol source [43].

2.3.2 Range coding

Range coding [44] [45] is a kind of entropy coding, and is quite similar to the arithmetic coding. Range coding may use larger basis other than bits, which is used as basis in the arithmetic coding. The principle of the range coder to encode or decode a symbol $k$ is based on its frequency count.

Figure 2 - 2-dimensional vector quantization: uniform (left), non-uniform (right). Adapted from [41].
\( f[k] \), which represents the probability of occurrence of the symbol \( k \). For a source of symbols, the context of a symbol \( k \) is described with a three-tuple \((fl[k], fh[k], ft)\), with \( 0 \leq fl[k] \leq fh[k] \leq 2^{32} - 1 \). The frequency count of symbol \( k \) is the difference between \( fh[k] \) and \( fl(k) \), and \( ft \) is the sum of frequency counts of all distinct symbols in the source. To decrease the complexity of the calculation in the process of encoding and decoding, the value of \( ft \) is supposed to be less than \( 2^{32} \). The relation of the members of the three-tuple can be expressed according to:

\[
\begin{align*}
fl[k] &= \sum_{i=0}^{k-1} f[i], \\
fh[k] &= f[k] + fl[k], \\
ft &= \sum_{i=0}^{n-1} f[i]
\end{align*}
\] (4)

The range coding adopted in this thesis is introduced below. The key parameters in this range coding scheme are specified considering the trade-off between compressing performance and calculation complexity.

### 2.3.2.1 Range encoder

The range encoder maintains two 32-bit unsigned internal states: \( val \) and \( rng \) (shown in Figure 3), which represents the lower end and the size of the current range respectively.

![Figure 3 - Internal states of the encoder](image)

First the internal states \( val \) and \( rng \) are initially assigned to 0 and \( 2^{31} \). Then range encoder updates its internal states with the context of the symbol \( k \) to be encoded as follows:

\[
val = val + rng \frac{fl[k]}{ft}
\] (5)

If \( fl[k] \) is larger than zero,

\[
rng = rng \frac{fh[k] - fl[k]}{ft}
\] (6)

Otherwise,

\[
rng = rng - rng \frac{fh[k] - fl[k]}{ft}
\] (7)

The division operation in the equations above is integer division.
Finally the range is normalized and the operations of carry propagation and output buffering follows afterwards. The details of the implementation about the renormalization and carry propagation can be found in [44], [45].

2.3.2.2 Range decoder

The range decoder also maintains two 32-bits state variables: $val$ and $rng$ shown in Figure 4. The variable $rng$ has the same meaning as that in the encoder. But the variable $val$ represents the difference of the upper end of the current range and the position of the encoded value, minus one, which is different from that in the encoder.

![Figure 4 - Internal states of the decoder](image)

First, the range decoder initializes $rng$ to 128 and $val$ to $(127 - (b_0 >> 1))$, where $b_0$ is the first input byte from the bit-stream.

Then a 16-bit value $f_s$ which reflects the position of a symbol is calculated according to:

$$ f_s = ft - \min\left( \frac{val}{rng}, ft + 1, ft \right) $$

The decoder then searches for a symbol $k$ whose context includes the value $f_s$. After determining the symbol $k$, state variable $val$ is updated as

$$ val = val - rng \frac{ft - fh[k]}{ft} $$

If $fl[k]$ is larger than zero,

$$ rng = rng \frac{fh[k] - fl[k]}{ft} $$

Otherwise,

$$ rng = rng - rng \frac{ft - fh[k]}{ft} $$

The division operator in the equations above denotes integer division.
Finally update the decoder state with the symbol found. The decoder normalizes its range repeatedly until $rng > 2^{23}$, or does nothing if its range is already greater than $2^{23}$. More details of this process are described in [44], [45].

### 2.4 Performance evaluation

The most common objective evaluation criterion for a compression algorithm is the signal-noise-ratio (SNR), which usually is the ratio of the energy of the original signal and that of the difference between the original and the reconstruction in $L_2$ norm. The SNR is calculated according to:

$$SNR = 10 \log_{10} \left( \frac{\sum_{n=0}^{M-1} s^2(n)}{\sum_{n=0}^{M-1} (s(n) - \hat{s}(n))^2} \right)$$

where $s(n)$ and $\hat{s}(n)$ are original and the reconstructed signal respectively. However, the formula of SNR above only represents the general performance of a compression coding over a long time and cannot reflect the temporal variations of the quality of the reconstructed signal. Another objective criterion called segmental SNR (SEGSNR), representing the average SNR of the signal frame by frame, can detect the short-time variation of the reconstruction quality and is given by

$$SEGSNR = \frac{1}{M} \sum_{i=0}^{M-1} 10 \log_{10} \left( \frac{\sum_{j=0}^{N-1} s^2(iN + j)}{\sum_{j=0}^{N-1} (s(iN + j) - \hat{s}(iN + j))^2} \right)$$

where $M$ and $N$ are the number of frames and samples in each frame respectively. Objective measure method is effective to evaluate things that have a clear standard, like voltage, temperature, the sizes of machine parts, etc. But it is not enough to evaluate the performances of audio coding methods using objective criteria only. Instead, subjective tests are widely used and are recognized as the most reliable way to evaluate low bit-rate audio codecs. MUSHRA (Multiple Stimuli with Hidden Reference and Anchor) is one of the common subjective evaluation methods for the perceived quality of the output from lossy audio and speech codecs. In MUSHRA each listener is presented with a reference, some audio outputs from different codecs, a hidden reference and at least one hidden anchor. An anchor represents a low-pass filtered version of the original signal and is used to avoid rating minor artifacts as having very bad quality. The range of the scores given by listeners is from 0 to 100 and therefore it provides a high discrimination in very small differences.
3 G.719

G.719 [34] is an ITU-T standard audio codec which provides low-delay, high quality full-band speech and audio signal coding at 32-128 kbit/s. The codec is based on transform coding, and adopts the Modified Discrete Cosine Transform (MDCT) to convert signal from time domain to frequency domain. After transform, the spectral coefficients are normalized and quantized by lattice vector quantization.

3.1 System description

3.1.1 Encoder overview

The block diagram of the encoder is shown in Figure 5. The sampled input signal is first processed by a transient detector where a decision if a frame is transient mode or stationary mode is made. Then the input signal is transformed based on the mode of the frame. If the frame is stationary, a high frequency resolution transform is applied to the input signal. Otherwise, the input signal is divided into four 5-ms parts and a high time resolution transform is used on these sub-frames. After the adaptive transform, the spectral coefficients are divided into 44 sub-vectors (see Table 1) representing frequency bands up to 20kHz. The norms of these sub-vectors are calculated and quantized, and the spectral coefficients are normalized by the corresponding quantized norms. Next the bit-allocation algorithm allocates the remaining bits to the 44 sub-vectors based on their norms and perceptual importance. Then the normalized spectral
coefficients are quantized by lattice vector quantization, with the result from the bit-allocation algorithm. Noise level is estimated and quantized when a frame is in stationary mode.

### 3.1.2 Decoder overview

![Figure 6 - Block diagram of G.719 decoder [34]. See text for explanations.](image)

Figure 6 shows the structure of the decoder. First, the norm of each sub-vector is decoded and the bit-allocation algorithm is carried out based on the norms and the perceptual model used in the encoder. Then the normalized spectral coefficients are decoded according to the result from the bit-allocation algorithm. In case some sub-vectors are not quantized in the encoder, the decoder fills spectrum for them with a method called spectrum filling (see 3.3.3). A spectral codebook is created by the decoded transform coefficients, and the spectrum filling method fills the non-coded spectrum with elements from the codebook. Next, the spectral coefficients in sub-vectors can be obtained by the multiplication of norms and the normalized spectral coefficients. Finally the full-band audio signal is reconstructed by the inverse transform performed on the spectral coefficients based on the flag that indicates whether a frame is transient or stationary.

### 3.2 Encoder

#### 3.2.1 Transient detection

![Figure 7 - Transient detection [34].](image)

The structure of the transient detection is shown in Figure 7. The sampled input signal is first processed by a high-pass filter to remove undesired low frequency signal. The high-pass filter is given by [34]:

\[ H(f) = \begin{cases} 
1 & \text{if } f > f_c \\
0 & \text{otherwise} 
\end{cases} \]
\[ H_{hp}(z) = \frac{0.7466(1-z^{-1})}{1-0.4931z^{-1}} \] (14)

Then the high-pass filtered signal is divided into four 5-ms sub-frames, and we calculate the energy \( E(m), m = 0,1,2,3 \) of each sub-frame.

The long-term energy \( E_{LT}(m) \) of four sub-frames is calculated as follows [34]:

\[ E_{LT}(m) = 0.75E_{LT}(m-1) + 0.25E(m), m = 0,\ldots,3 \] (15)

\( E_{LT}(-1) \) can be acquired from the last sub-frame \( E_{LT}(3) \) of the previous frame. A transient is detected if the ratio of the short-term energy \( E(m) \) and the long-term energy \( E_{LT}(m) \) in any sub-frame is above a threshold [34]:

\[ \frac{E(m)}{E_{LT}(m)} \geq 7.8dB, m = 0,1,2 \text{ or } 3 \] (16)

If a transient is detected on one frame, the next one is also considered as transient because the operation of the adaptive time-frequency transform performs on the consecutive two frames.

### 3.2.2 Adaptive time-frequency transform

The adaptive time-frequency transform can change its time or frequency resolution based on the mode of the frame. If the frame is stationary, the transform provides a high frequency resolution in order to represent stationary signals. In the case of transient mode, the time resolution has to be increased to capture the rapid changes of the signal spectral features.

The adaptive time-frequency transform of both modes shares the same operation of the windowing and time aliasing. After that the transform of stationary mode performs a type IV discrete cosine transform (DCTIV). If the frame is transient, the transform will give four pieces of spectrum with better time domain resolution. More details of the transform can be found in [34].

#### 3.2.3 Grouping of spectral coefficients

For the stationary mode, the spectral coefficients are divided into four groups and each group has frequency bands in the same length. The grouping of the spectral coefficients of a frame in the stationary mode is shown in Table 1.

<table>
<thead>
<tr>
<th>Group</th>
<th>Length of sub-vectors</th>
<th>Number of sub-vectors</th>
<th>Number of coefficients</th>
<th>Indices of coefficients</th>
<th>Bandwidth (Hz)</th>
<th>Start (Hz)</th>
<th>End (Hz)</th>
</tr>
</thead>
<tbody>
<tr>
<td>I</td>
<td>8</td>
<td>16</td>
<td>128</td>
<td>0 - 127</td>
<td>3200</td>
<td>0</td>
<td>3200</td>
</tr>
<tr>
<td>II</td>
<td>16</td>
<td>8</td>
<td>128</td>
<td>128 - 255</td>
<td>3200</td>
<td>3200</td>
<td>6400</td>
</tr>
<tr>
<td>III</td>
<td>24</td>
<td>12</td>
<td>288</td>
<td>256 - 543</td>
<td>7200</td>
<td>6400</td>
<td>13600</td>
</tr>
<tr>
<td>IV</td>
<td>32</td>
<td>8</td>
<td>256</td>
<td>544 - 799</td>
<td>6400</td>
<td>13600</td>
<td>20000</td>
</tr>
<tr>
<td>Total</td>
<td>44</td>
<td>800</td>
<td>20000</td>
<td></td>
<td></td>
<td></td>
<td></td>
</tr>
</tbody>
</table>
There are 44 frequency bands in total. The sub-vectors in short length are for lower frequency bands to achieve a high frequency resolution, and the low frequency resolution is used for higher frequency bands. The grouping of the spectral coefficients considers the characteristics of human hearing and help the system utilize bits efficiently.

In the case of transient mode, four 5-ms sub-frame spectral coefficients are interleaved by reordering the coefficients of all 4 sub-frames in the frequency increasing order in the whole frame (shown in Figure 8). After interleaving, the coefficients are divided to four groups in the same way as that in the stationary mode.

![Figure 8 - Interleaving in the transient mode. The left half is a frame before interleaving, and coefficients of each sub-frame are in the frequency increasing order. The right half is a frame after interleaving, and coefficients of the same frequencies of all 4 sub-frames are adjacent to each other.](image)

### 3.2.4 Norm quantization

If a frame is transient, the norms of four 5-ms sub-frames are re-ordered for efficient differential encoding. The approach is to reorder the norms of the 1st and 3rd 5-ms sub-frames in the increasing order and the norms of the 2nd and 4th 5-ms sub-frames in the decreasing order. Figure 9 illustrates this process.
The norms, $N(p)$, of a sub-vector is calculated with the root-mean-square (RMS) of the sub-vector according to:

$$N(p) = \sqrt{\frac{1}{L_p} \sum_{k=s_p}^{e_p} y(k)^2}, \quad p = 0, \ldots, P-1$$  \hspace{1cm} (17)

where $p$ and $L_p$ represent the index and the size of the sub-vector, and $s_p$ and $e_p$ are the first and the last spectral coefficients respectively. Then a uniform logarithmic scalar quantizer is used in each frame to quantize these norms with 40 steps of 3dB, ranging from $2^{17.0}$ to $2^{-2.5}$.

The index of the first sub-vector is sent to the decoder directly after uniform logarithmic scalar quantization. The rest of the quantization indices are differentially coded and then followed by Huffman coding [48]. If the bits consumed for the differential indices encoded with Huffman coding is less than that without Huffman coding, then Huffman coding will be applied to these indices, and vice versa.

The method to quantize the norms of the sub-vectors is effective. Before differential coding, the norms of the sub-vectors are reordered so that the corresponding bands that are arranged at close positions are frequency adjacent. So the output of the differentiating coding is relatively concentrated in distribution, and that is very suitable to be encoded by Huffman coding.

At the bit-rate of 32Kbps, the sub-vectors of the normalized spectral coefficients in the group IV are only encoded if the largest norm of all frequency bands is located in the 34th band or any one in higher frequency.
3.2.5 Bit-allocation

The bit-allocation algorithm in G.719 distributes the available bits among all the frequency bands that people can perceive. The algorithm runs based on two features: the energy and the perceptual property of each band.

The norms adjusted based on the perceptual model are used in the bit-allocation procedure. The method how the norms are adjusted based on the perceptual model can be found in [34]. The result of the algorithm is stored in a bit-allocation vector. The elements of this vector represent the number of the bits allocated to each normalized coefficient in a sub-vector. The maximum value in the bit-allocation vector is upper limited to 9.

The elements in the bit-allocation vector are set to zeros initially. Then a loop is created and in each loop the band whose adjusted norm is the largest is found. This band is assigned one more bit per coefficient, and correspondingly the adjusted norm of this band is decreased by two. If the number of the bits allocated to one band achieves the upper limit, the adjusted norm of that band will be set to a minimum value and the bits for that band cannot increase any more. The loop will stop if the number of the remaining bits is not enough to assign to any band.

A more effective algorithm for bit allocation in the recommended code of G.719 is implemented in an indirect way. Before starting the loop of the allocation, reorder the adjusted norms of all bands in decreasing order. Then a floating window is created to indicate the candidates that participate in the competition of the resource of bits. The size of the floating window is set to one initially, and it is increased by one if the largest adjusted norm is the last member in the floating window. The termination condition is the same as the original one. Figure 10 shows the bit-allocation process in one instant.

![Diagram showing the bit-allocation process](image)

*Figure 10 – The band whose adjusted norm is the largest is allocated one more bit per coefficient, and correspondingly the adjusted norm of this band is decreased by two. If the*
largest adjusted norm is the last member in the floating window, the size of floating window is increased by one.

3.2.6 Shape quantization

The quantizer for the normalized spectral coefficients will be introduced in the chapter 4.

3.2.7 Noise level adjustment

The spectral coefficients in the sub-vectors which are allocated zero bits from bit-allocation algorithm will not be encoded. Although not all spectral coefficients are sent to the decoder, some useful features of these sub-vectors can be captured and help the decoder improve the perceived quality.

The non-coded signal level is estimated only in the case of stationary mode. The transition frequency, which is between the noise-filling region and the bandwidth extended region (see 3.3.3) of one frame, is determined before estimating the signal level of the frame. The transition frequency has to be determined at the decoder as well and it represents the transition from the noise-filling region to the bandwidth extended region.

One way to determine the transition frequency is to loop through all the sub-vectors from high to lower frequencies. If one sub-vector is not assigned any bits, it belongs to the bandwidth extended region. Once one coded band is found the frequency at the end of this band will be recorded as the transition frequency, and the bands below this transition frequency belongs to the noise-filling region. Figure 11 shows such a process.

\[
\text{NoiseLevel} = \frac{1}{|\mathcal{S}|} \sum_{k \in \mathcal{S}} \log_2 (y(k)^2) - \log_2 \left\{ \frac{1}{|\mathcal{S}|} \sum_{k \in \mathcal{S}} y(k)^2 \right\}
\]

where \( \mathcal{S} \) indicates the set of indices of non-quantized normalized spectral coefficients in the noise-filling region. A 2-bit scalar quantization codebook for quantizing the noise level calculated is shown in the Table 2.

<table>
<thead>
<tr>
<th>Index</th>
<th>Output quantized NoiseLevel</th>
</tr>
</thead>
<tbody>
<tr>
<td>0</td>
<td>0</td>
</tr>
<tr>
<td>Index</td>
<td>Output quantized NoiseLevel</td>
</tr>
<tr>
<td>-------</td>
<td>-----------------------------</td>
</tr>
<tr>
<td>1</td>
<td>–6</td>
</tr>
<tr>
<td>2</td>
<td>–12</td>
</tr>
<tr>
<td>3</td>
<td>–18</td>
</tr>
</tbody>
</table>

### 3.3 Decoder

#### 3.3.1 Norm decoding

If Huffman coding is applied to the indices of the norms of sub-vectors in the encoder, the quantization indices of these norms are recovered by the Huffman decoding followed by the differential decoding. Otherwise, the indices can be obtained directly by the differential decoding. The encoder transmits a flag bit to inform the decoder whether the Huffman coding is applied. The relevant information about the quantization codebook for the norms of sub-vectors is described in 3.2.4.

#### 3.3.2 Spectral coefficient decoding

The decoding of the normalized spectral coefficients will be covered in chapter 4.

#### 3.3.3 Spectrum filling

Before the spectrum filling, the transition frequency has to be estimated, and the method to do this is the same as that in the encoder (see 3.2.7). Spectrum filling comprises two parts. The first part, called noise filling, fills the non-coded spectrum between the lowest frequency and the transition frequency $f_t$; the second one, bandwidth extension, is to regenerate the spectrum above the transition frequency using the lower frequency spectrum.

The connection between these two components is illustrated in Figure 12. More details about spectrum filling can be found in [34].
3.3.4 **Noise level adjustment**

The spectral coefficients in the noise filling region are adjusted based on the *NoiseLevel* (see 3.2.7) estimated in the encoder. In the case of transient mode, the parameter *NoiseLevel* is set to 0dB by default. The *NoiseLevel* adjustment is done according to the equation below [34]:

\[
 y^\text{i}(k) = 2^{-\text{NoiseLevel}} y^\text{in}(k), \quad \text{for } k \text{ such that } R(p_k) = 0 \text{ and } k \leq f_i
\]  

(19)

where \( y^\text{i}(k) \) is the spectral coefficients generated in the noise filling process and \( R(p_k) \) is the number of bits allocated to the sub-vector \( p_k \), which includes the spectral coefficient at frequency \( k \). The noise level index is estimated with the non-coded transform coefficients below the transition frequency at the encoder side. This index reflects the degree of the flatness of these spectral coefficients. The flatter these non-coded spectral coefficients are, the less attenuation will be applied to the corresponding spectral coefficients in the decoder.

3.3.5 **De-normalization**

After the generation of the full-bandwidth audio spectral coefficients, these coefficients are de-normalized by multiplying the received quantized norms of sub-vectors. The rest of the spectral coefficients, which correspond to the frequency bands above 20kHz, the upper limit of human hearing capacity, are set to zeros.

3.3.6 **Inverse transform**

The inverse transform is the reverse operation of the adaptive time-frequency transform in 3.2.2. More related details can be found in [34].
4 Quantization theory

This chapter describes two kinds of important vector quantization methods, FLVQ and PVQ. FLVQ is 8-dimensional lattice vector quantization, and a vector is supposed to be quantized to the closest quantization vector in a specified lattice vector set by FLVQ. PVQ is different from FLVQ, and it is based on pulses.

4.1 Fast lattice vector quantization

The FLVQ scheme in G.719 is used to quantize 8-dimensional vectors acquired from sub-vectors. FLVQ consists of two components: an RE8 based lattice vector quantizer named LVQ1, which is adopted when an 8-dimensional vector is assigned one bit per normalized coefficients, and a D8-based lattice vector quantizer named LVQ2 [36], which is used when more than one bit per coefficient is allocated to an 8-dimensional vector of normalized coefficients. All 8-dimensional vectors in the same sub-vector will be allocated the same number of bits.

4.1.1 Lattice

A lattice is a combination of linear independent vectors of the same dimension. Suppose \(a_1, a_2, \ldots, a_n\) are \(m\)-dimensional independent vectors, then the set of these vectors

\[
\Lambda = u_1 a_1 + u_2 a_2 + \cdots + u_n a_n
\]

is called an \(n\)-dimensional lattice, where \(u_1, u_2, \ldots, u_n\) are arbitrary integers.

Lattice has another equivalent expression with its generator matrix. For example, a generator matrix [36] of the lattice \(D_8\) is

\[
G = \begin{bmatrix}
2 & 0 & 0 & 0 & 0 & 0 & 0 & 0 \\
1 & 1 & 0 & 0 & 0 & 0 & 0 & 0 \\
1 & 0 & 1 & 0 & 0 & 0 & 0 & 0 \\
1 & 0 & 0 & 1 & 0 & 0 & 0 & 0 \\
1 & 0 & 0 & 0 & 1 & 0 & 0 & 0 \\
1 & 0 & 0 & 0 & 0 & 1 & 0 & 0 \\
1 & 0 & 0 & 0 & 0 & 0 & 1 & 0 \\
1 & 0 & 0 & 0 & 0 & 0 & 0 & 1
\end{bmatrix}
\]

The lattice can also be expressed with the generator by:

\[
\Lambda = uG
\]

where \(u = (u_1, u_2, \ldots, u_n)\), and \(G\) is equivalent to \((a_1, a_2, \ldots, a_n)^T\).

The inverse of the generator matrix of a lattice is also useful in some practical cases, and the inverse matrix of \(G\) is
\[
G^{-1} = \begin{bmatrix}
0.5 & 0 & 0 & 0 & 0 & 0 & 0 & 0 \\
-0.5 & 1 & 0 & 0 & 0 & 0 & 0 & 0 \\
-0.5 & 0 & 1 & 0 & 0 & 0 & 0 & 0 \\
-0.5 & 0 & 0 & 1 & 0 & 0 & 0 & 0 \\
-0.5 & 0 & 0 & 0 & 1 & 0 & 0 & 0 \\
-0.5 & 0 & 0 & 0 & 0 & 1 & 0 & 0 \\
-0.5 & 0 & 0 & 0 & 0 & 0 & 1 & 0 \\
-0.5 & 0 & 0 & 0 & 0 & 0 & 0 & 1 \\
\end{bmatrix}
\]

(23)

\(D_8\) is one of the typical lattices and is defined as [36]

\[
D_8 = \{(y_0, y_1, y_2, y_3, y_4, y_5, y_6, y_7) \in \mathbb{Z}_8 \mid \sum_{i=0}^{7} y_i = \text{even}\}
\]

(24)

\(RE_8\) [36] is also a well-known lattice and is defined based on \(D_8\) as

\[
RE_8 = (2D_8) \bigcup (2D_8 + [1,1,1,1,1,1,1,1])
\]

(25)

The lattice \(RE_8\) is composed of the points on concentric spheres with radius of \(2\sqrt{2m}\) centered at the origin, where \(m\) is a non-negative integer. Every point \(x = (x_1, x_2, ..., x_8)\) of \(RE_8\) satisfies the condition:

\[
\sum_{i=1}^{8} x_i^2 = 8m
\]

(26)

A class of a lattice \(\Lambda\) is a set of vectors \(\{x : x \in \Lambda\}\) which represents permutations of the same group of elements.

### 4.1.2 FLVQ encoder

#### 4.1.2.1 Low bit rate

The lattice \(RE_8\) is composed of the points on concentric spheres with radius of \(2\sqrt{2m}\) centered at the origin, where \(m\) is a non-negative integer. These points on the concentric spheres are used as the representative in the codebook [36] for LVQ1.

The codebook of LVQ1 is composed of all 240 points on the \(RE_8\) when \(m = 1\) and 16 additional points which are used to quantize the input vectors with low energy. The representative points in the codebook of LVQ1 are arranged in a specified order in order to index fast, and can be found in [34].

An 8-dimensional vector that is assigned only one bit per coefficient can be quantized by LVQ1 as follows. First, add an offset to all the elements in the vector and scale them appropriately to match the range of the codebook. Next, sort the processed elements in decreasing order and find the best-matched vector \(v\) in Table 3, which lists leaders of the codeword in the codebook of LVQ1. Then obtain the best-matched codeword by reordering the elements of \(v\) in the original order. Finally compute the index of the codeword.
<table>
<thead>
<tr>
<th>Index</th>
<th>Leader</th>
</tr>
</thead>
<tbody>
<tr>
<td>0</td>
<td>0 0 0 0 0 0 0</td>
</tr>
<tr>
<td>1</td>
<td>2 0 0 0 0 0 0</td>
</tr>
<tr>
<td>2</td>
<td>0 0 0 0 0 0 -2</td>
</tr>
<tr>
<td>3</td>
<td>2 0 0 0 0 0 0</td>
</tr>
<tr>
<td>4</td>
<td>2 2 0 0 0 0 0</td>
</tr>
<tr>
<td>5</td>
<td>1 1 1 1 1 1 -1</td>
</tr>
<tr>
<td>6</td>
<td>1 1 1 1 -1 -1 -1 -1</td>
</tr>
<tr>
<td>7</td>
<td>1 1 -1 -1 -1 -1 -1 -1</td>
</tr>
<tr>
<td>8</td>
<td>-1 -1 -1 -1 -1 -1 -1 -1</td>
</tr>
<tr>
<td>9</td>
<td>1 1 1 1 1 1 1 1</td>
</tr>
</tbody>
</table>

4.1.2.2 High bit rate

When one 8-dimensional spectral coefficient vector is allocated 2 – 9 bits per coefficient, the vector can be quantized by LVQ2. The first step (offset and scaling) is the same as that in the LVQ1. Next, find the closest lattice point \( \mathbf{v} \) in \( D_8 \) to the scaled vector by the searching algorithm described in [36]. Then compute the index vector \( k = (k_1, k_2, ..., k_8) \) of \( \mathbf{v} \) in the codebook truncated by \( R(p) \) according to:

\[
k = vG^{-1} \mod r \quad \text{with} \quad r = 2^{R(p)}
\]  

(27)

where \( G \) is the generator matrix of \( D_8 \) described in Equation (21) and \( R(p) \) is the number of bits allocated for sub-band \( p \).

Finally, compute the code-vector \( \mathbf{c} \) with the index vector \( k \) by the algorithm described in [36], and compare \( \mathbf{v} \) with \( \mathbf{c} \). If \( \mathbf{v} \) and \( \mathbf{c} \) are the same, it means that the scaled vector is in the support of the truncated codebook limited by \( R(p) \), and then \( k \) is the index of the closest lattice point to the scaled vector. Otherwise, the input vector is an outlier of the truncated codebook, and it will be scaled down repeatedly until the scaled input vector is inside the support of the truncated codebook limited by \( R(p) \). Figure 13 illustrates the truncation in 2-dimensional vector.
After lattice vector quantization, a kind of entropy coding, Huffman coding, is optionally used to decrease the bits transmitted. If the bits transmitted for the indices of the shape quantization with Huffman coding is less than that without Huffman coding, this kind of entropy coding is applied when \( 2 \leq R(p) \leq 5 \), and the saved bits are allocated for the vectors which do not get any bits in the bit-allocation procedure. Otherwise Huffman coding is not adopted. The decision whether the Huffman coding is used is transmitted to the decoder as side information. The Huffman codes for the FLVQ indices can be found in [34].

### 4.1.3 FLVQ decoder

The decoding process of the spectral coefficients varies based on whether Huffman coding is applied on the indices of the normalized spectral coefficients. If Huffman coding is used and \( 2 \leq R(p) \leq 5 \), the received data will be Huffman decoded to acquire the indices of vectors. Otherwise, the indices of vectors can be directly read from the received bit stream.

The first step to decode an 8-dimensional vector is to find the code-vector based on its index and bit-allocation vector \( R \). Then scale and offset the code-vector, reversely operating as the instruction in the FLVQ encoder.

### 4.2 Pyramidal vector quantization

PVQ is another way to quantize the shape of spectrum. The idea of PVQ is to quantize an \( N \)-dimensional vector by allocating "pulses" to each element of the vector to match the shape of the target vector. "Pulse" is a quantization unit for each dimension in a vector. For example, suppose you have 3 pulses to quantize an 8-dimensional vector, which means you can distribute these 3 pulses to any dimension in this vector. These 3 pulses can be put in one dimension, like \((3,0,0,0,0,0,0,0)\) or \((0,0,-3,0,0,0,0,0)\). They can also be distributed on different dimensions, like \((1,-2,0,0,0,0,0,0)\) or \((0,1,-1,0,1,0,0,0)\). The number of pulses available can be calculated based on the number of dimensions of a vector and the number of bits available. The method to distribute the pulses depends on the energy of each dimension of a vector. The quantization becomes more accurate if more pulses are assigned.

A PVQ codebook is a combination of \( K \) signed pulses in an \( N \)-sample vector, where pulses at the same position are required to have the same sign. The size of this PVQ codebook is denoted as \( V(N,K) \).
4.2.1 Mapping between bits and pulses

The number of bits allocated for an $N$-dimensional vector limits the maximum size of the codebook to quantize this vector. The size of PVQ codebook to quantize a vector depends on the number of the dimensions of the vector and the number of available pulses. Based on the previous facts, there is a connection between the bits and the pulses if the number of dimensions of the vector to quantize is fixed.

To acquire good performance, the coder searches for the maximum number of pulses for PVQ based on the bits allocated for quantizing. To decrease running time, the search algorithm uses a pre-computed bit-to-pulse table, and only the number of pulses that makes $V(N, K)$ less than $2^{32}$ is valid to avoid complex calculations.

4.2.2 Calculation of the size of the codebook

$V(N, K)$ is the number of combinations of $K$ pulses in $N$ positions and is used to calculate the size of a PVQ codebook for quantizing an $N$-dimensional vector with $K$ pulses. The calculation of $V(N, K)$ can be performed in a recursive way with the relation [35]:

$$V(N, K) = \begin{cases} 
1, & K = 0 \\
0, & N = 0, K \neq 0 \\
V(N-1, K) + V(N, K-1) + V(N-1, K-1), & \text{otherwise}
\end{cases}$$

(28)

4.2.3 PVQ Encoder

There are some possible methods to search for the best matched code-vector with a tradeoff between performance and complexity. The approach introduced here has two steps.

The first step is to compute an initial code-vector $y_0$ by projecting a vector $X$ to the $K-1$ pulses pyramid according to:

$$y_0 = \frac{(K-1)X}{\sum |X|}$$

(29)

The initial codeword $y_0$ may have $K-1$ non-zero values at most.

The second step is to adjust the initial code-word with the remaining pulses, which are utilized in a way to maximize the normalized correlation between $X$ and $y$:

$$J = \frac{X^T y}{\|y\|}$$

(30)

According to these two steps, the best matched code-vector $Y$ is acquired, and is indexed [33] next. Suppose an $N$-dimensional vector $Y$ is encoded by PVQ with $K$ pulses, the indexing process is shown in the following pseudo-code:
\[ i = 0, k = 0 \]

\[ \text{for } j = N - 1: -1: 0 \]

\[ \text{if } k > 0 \]

\[ i = i + (V(N - j - 1, k - 1) + V(N - j, k - 1)) / 2 \]

\[ k = k + \text{abs}(Y[j]) \]

\[ \text{if } Y[j] < 0 \]

\[ i = i + (V(N - j - 1, k) + V(N - j, k)) / 2 \]

\[ \text{end} \]

where \( i \) is the index of \( Y \).

According to the encoding process of PVQ, we can deduce that the output vectors of PVQ are usually sparse. The reason is that the number of pulses is usually very small even if the number of bits is in a medium level. If the number of pulses is smaller than or close to the number of dimensions and the energy is concentrated in one or a couple of dimensions of the vector, the dimensions that do not have enough energy will not acquire pulses and be encoded as zeros, which leads to the sparsity of PVQ outputs. However, FLVQ do not suffer from this problem because most of lattice quantization points are not sparse.

### 4.2.4 PVQ Decoder

The indices of the code-vectors in a PVQ codebook are in the range from 0 to \( V(N, K) - 1 \), where \( V(N, K) \) is the size of this PVQ codebook. Suppose \( i \) is a vector index decoded from the bit-stream, so \( 0 \leq i \leq V(N, K) - 1 \), then the decoded vector can be recovered by the following pseudo-code:

\[ k = K \]

\[ \text{for } j = 0 : N - 1 \]

\[ p = (V(N - j - 1, k) + V(N - j, k)) / 2 \]

\[ \text{if } i < p \]

\[ \text{sgn} = 1 \]

\[ \text{else} \]

\[ \text{sgn} = -1, i = i - p \]

\[ \text{end} \]

\[ k0 = k, p = p - V(N - j - 1, k) \]

\[ \text{while } p > i \]

\[ k = k - 1, p = p - V(N - j - 1, k) \]

\[ \text{end} \]

\[ X[j] = \text{sgn}^*(k0 - k), i = i - p \]

\[ \text{end} \]

Finally the decoded vector \( X = (x_0, x_1, \ldots, x_{N-1}) \) is normalized to meet \( \sum_{i=0}^{N-1} x_i^2 = 1 \).
4.2.5 Split

The maximum size of PVQ codebooks is up to $2^{32}$, to avoid complex calculation. If the bits allocated to one band are allowed for a larger codebook, the sub-vector (see Table 1) representing this band is split into two smaller vectors of half size. A gain parameter is quantized to represent relative gains of these two half-size vectors, and this coding procedure is recursively performed. Figure 14 shows a possible split of an $N$-dimensional vector.

The PVQ splitting has to pay for the cost of the transmission of the parameter, gain, but this operation actually avoids more waste if the number of bits assigned to a vector is much larger than 32.
5 Implementation

The previous chapters introduce the background, the G.719 standard, and the theory of three different quantization schemes. This chapter will give some details about how they are implemented. The audio codec G.719, which is the framework of this study, is already fully elaborated in the previous chapter, and this chapter will show some details of the two modified versions of the audio codec, in which the original lattice vector quantization (FLVQ) is replaced with new quantization schemes (PVQ and SQ).

5.1 Bit-stream format

Figure 15 shows the bit-stream format of G.719 [34]. The transient flag indicates if the frame is transient and FlagL indicates if the fourth group is encoded. FlagN and FlagC label a frame if its norms and normalized spectral coefficients are Huffman coded. Then the bit-stream of norms and transform coefficients follows. In the stationary mode, 2 bits are used to encode the noise level index.

The other two modified versions of G.719 use the same bit-stream as that of G.719, though there are some small differences when forming and processing the bit-stream in the encoder and decoder. For example, FlagC is useless when SQ is used instead of the original lattice vector quantization, and “Norm code bits” are not processed in the decoder when PVQ is adopted for quantizing transform coefficients. More details can be found in the following sections.

<table>
<thead>
<tr>
<th>Transient flag</th>
<th>FlagL</th>
<th>FlagN</th>
<th>FlagC</th>
<th>Norm code bits</th>
<th>Transform coefficients code bits</th>
<th>Noise level index</th>
</tr>
</thead>
<tbody>
<tr>
<td>1 bit</td>
<td>1 bit</td>
<td>1 bit</td>
<td>1 bit</td>
<td></td>
<td>2 bits</td>
<td></td>
</tr>
<tr>
<td>a) Case of stationary frames</td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
</tr>
</tbody>
</table>

<table>
<thead>
<tr>
<th>Transient flag</th>
<th>FlagL</th>
<th>FlagN</th>
<th>FlagC</th>
<th>Norm code bits</th>
<th>Transform coefficients code bits</th>
</tr>
</thead>
<tbody>
<tr>
<td>1 bit</td>
<td>1 bit</td>
<td>1 bit</td>
<td>1 bit</td>
<td></td>
<td></td>
</tr>
<tr>
<td>b) Case of transient frames</td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
</tr>
</tbody>
</table>

Figure 15 - Bit-stream format of audio codecs [34].

5.2 G.719 with scalar quantization

SQ is a very common and basic way of quantizing. It is also well-known because this quantization approach is easy to design, and consumes a fairly low amount of memory.

In this section we design a modified version of G.719 in which the full-band audio codec uses a mid-rise scalar quantization instead of its original vector quantization (FLVQ). The other modules including pre-processing, transform, quantization of norms and bit-allocation algorithm, remain unchanged. A comparison of these two audio coders will be discussed in the next chapter.
5.2.1 Creation of a codebook

A uniform scalar quantization followed by entropy coding can achieve an optimal performance, and this method is used in this thesis. The first step to design the uniform scalar quantization to be integrated in G.719 is to create the codebook for quantizing the normalized spectral coefficients.

Considering each sub-vector can be allocated from 0 to 9 bits for each element in the bit-allocation algorithm, nine uniform scalar quantization codebooks are created to accommodate this. The maximum uniform scalar codebook has $2^9 = 512$ entries, while the minimum has only 2 entries. Any two consecutive intervals between entries in the same codebook have the same length.

For a codebook for a specified bit-rate, it is supposed to have a different range from the others for different bit-rates. If the ranges of codebooks for all bit-rates are the same, large distortions will occur. The codebooks designed for high bit-rate may lead to bad performance if the range is set too low, while the codebooks for low bit-rate are considered unreasonable if the range is set too high. Figure 16 shows the scalar quantization levels of the codebooks for different bit-rates. Therefore it is not wise to set the same range for uniform scalar quantizers for all bit-rates, and 9 different ranges are chosen for these codebooks. Once the number of entries and the range of a codebook are determined, the codebook is fixed.

One way to choose an appropriate range for a codebook for a certain bit-rate is to try all possible ranges for this codebook, and choose the one that leads to minimum distortion between the original signals and the quantized ones.

The first step to implement this idea is to choose a database consisting of a large amount of uncompressed digital audio signals. Then encode the audio signals of this database with a modified G.719 codec, which forces each sub-vector to be allocated a certain bit-rate $R$ for each element in the bit-allocation procedure. The modified G.719 codec also replaces its original vector quantization with the uniform scalar quantizer using the codebook for the selected bit-rate.
Finally a sufficient number of range values with a step-size \( r \) are tried in a loop to find the one that leads to the minimum distortion.

A practical problem in this implementation is to balance the running time of this range-finding algorithm and the accuracy of the range for the codebook. If the step-size of the range values is set too small in this algorithm, the running time is prohibitively long; if it is set too large, the result is not accurate. In our code an idea of an adaptive step-size is proposed to achieve an accurate result and a short running time at the same time.

The step-size of the range values is adaptive and it varies in different phases. When the range value in the loop is far from its best value, the step-size of the range value is large so that the algorithm can search quickly. Once the range value is close to the appropriate value, the step-size will become smaller repeatedly to achieve a sufficiently accurate result. Therefore this algorithm can obtain high precision with very little extra running time.

This adaptive step-size algorithm has one assumption that the distortion between the original and the quantized coefficients has only one signal optimum versus the range values of the codebook. This assumption holds in this algorithm theoretically. The normalized spectral coefficients to quantize are assumed to be zero-mean. The value of the range of SQ codebook is refined iteratively.

Suppose a source of normalized MDCT samples with maximum absolute value \( m \) is quantized by a uniform scalar quantizer, then the optimal range value (precision of \( d \) required) of its codebook consisting of \( n \) entries can be determined as follows:

1) Determine the initial step-size of the range values. Considering all samples are located in \([-m,m]\), 0.1 \( m \) or similar value could be the candidates of the initial step-size.

2) The maximum absolute value \( m \) of the source is assigned to the initial target value, which represents the direction that the range value to approach to decrease the distortion. Then initialize the range of the codebook to zero and start a loop to find the best value of the range.

3) In each loop the value of the range is increased or decreased by the step-size to approach the target value and determine the codebook of the uniform scalar quantizer with the current range value and the number of the entries in the codebook. Then encode and decode the source of samples with the modified G.719 audio codec, and calculate the distortion between the original and the reconstructed normalized MDCT coefficients.

4) If the distortion in the second iteration is larger than that in the first iteration, then stop the loop and choose a smaller initial step-size and go to 2). Otherwise, once the distortion begins to increase, record the range value \( v_0 \), \( v_1 \), \( v_2 \) in the current loop, the previous loop and the one before the previous loop, respectively.

5) Compute the distortion when the range of the codebook is \( v_1 + d \) and \( v_1 - d \) respectively. Either \( v_0 \) or \( v_2 \), the one closer to the range value \( (v_1 + d \text{ or } v_1 - d) \) that leads to a smaller distortion becomes the new target value, and the other one is abandoned. The current range value is back to \( v_1 \).
6) Half the step-size of the range value in the codebook and go to 3), the loop will stop if the step-size is less than the required precision of $d$.

![Figure 17 - Adaptive step-size algorithm. Each diamond point is a temporary range value obtained in one loop and the digits labeling these points represent the indices of the loops.](image)

Figure 17 shows a simple process of the algorithm. Each diamond point is a temporary range value obtained in one loop and the digits labeling these points represent the indices of the loops. The process of this algorithm in Figure 17 is illustrated as follows:

First, the initial step-size of the range value of the codebook and the target value are chosen as the algorithm described above. Then a loop is started and the point 1, 2, 3, 4, 5 are selected to be the range value based on the initial step-size. Notice the point 5 leads to the first increase of the distortion so we go back and focus on point 5, 4, 3. Then we calculate the distortion of two points, which have the same distance $d$ to the point 4, where $d$ is the required precision of the range value. The step-size is always larger than $d$ until the end of the loop. From the figure we know that the left one of these two points has smaller distortion, and therefore the point 3 is chosen as the next target value and the point 5 is abandoned. The current range value is back to point 4 and the step-size is halved. Next, the range value moves from point 4 to the target, point 3 with the update step-size. The next routine is point 4, point 6 and point 3. The point 3 is first increase in this routine and so we pay attention to point 3, point 6 and point 4. Calculate the distortion of the two points around point 6 respectively as we did last time and determine that point 4 is the next target and point 6 is the current range value. The following routine is point 6, point 7 and point 4 and the process continues until the step-size is smaller than $d$.

The results based on this algorithm are shown in Table 4.

<table>
<thead>
<tr>
<th>Bit-rate (bits per sample)</th>
<th>The number of entries in the codebook</th>
<th>Codebook ranges</th>
<th>Step-size</th>
</tr>
</thead>
<tbody>
<tr>
<td></td>
<td></td>
<td></td>
<td></td>
</tr>
</tbody>
</table>
5.2.2 Entropy coding implementation

To simplify the implementation, we do not adopt any kind of entropy coding in this version of the codec. Instead it is estimated that entropy coding could reduce the bit-rate for shape quantization by 20%, so this amount of bits are added to the bit-allocation in advance.

5.3 G.719 with PVQ

In this section another version of audio codec is designed. In this modified version the original FLVQ is replaced with PVQ to quantize the vectors of normalized coefficients. A few changes are introduced to make the new quantizer work in the G.719 framework. These changes are described in the following sections, and the rest of the system is left unchanged.

5.3.1 Quantization of norms

When estimating and quantizing the norm of each sub-vector $Y = (y_{p_1}, \ldots, y_{p_L})$ in G.719, the norm is defined as the RMS value of the sub-vector according to:

$$N(p) = \sqrt{\frac{1}{L_P} \sum_{k=1}^{L_P} y(k)^2}, \quad p = 0, \ldots, P-1$$

(31)

where $L_p$ is the size of the $p$-th sub-vector. Rewrite the expression and we see that the sum of each normalized sample squared is equal to:

$$\sum_{k=1}^{L_p} \left( \frac{y(k)}{N(p)} \right)^2 = L_p$$

(32)

However in PVQ, this sum is strictly equal to 1. So a small modification has to be made in this version of the codec when a sub-vector is normalized or de-normalized to resolve the mismatch.
When normalizing a sub-vector in encoder, we define the root-square value of the sub-vector as its norm as follows:

\[ N(p) = \sqrt{\sum_{k=p}^{e_p} y(k)^2}, \quad p = 0, \ldots, P-1 \]  

(33)

We apply the new norms defined above to the PVQ version of the codec to ensure that the sum of the squared normalized samples in one vector equals 1, which satisfies the requirement of PVQ.

Another problem in the implementation of this version of the codec is also about the normalization. In PVQ it is required that the sum of the squared normalized samples is strictly equal to 1, while in G.719 this sum can be around a specified value in a narrow range. The reason for this is that the FLVQ in G.719 not only quantizes the normalized coefficients, but also has a limited capacity to adjust the gains for them. But the PVQ based on pulses only quantizes the normalized coefficients without any adjustment.

The consequence of this problem is that the PVQ have to apply an almost perfect quantization for norms, otherwise a possible large distortion (usually 0 ~ 20%) due to coarse normalization will happen. However, G.719 does not suffer from it because the distortion generated by the coarse quantization of norms is refined by FLVQ.

So far we have not designed a new quantization scheme for the norms in the PVQ version of codec. We just transmit the norms of transparent quality to the decoder and subtract the same number of bits consumed in the quantization of norms in G.719 from the bit-allocation budget. Therefore the “Norm code bits” (see Figure 15) in the bit-stream are not processed when decoding.

### 5.3.2 Bit-pulse conversion and pre-calculated table

To avoid complex calculation, the number of the pulses, \( K \), is not allowed to make \( V(N, K) \) (the size of a codebook of \( N \)-dimensional vectors, see 4.2.2) exceed \( 2^{32} = 4,294,967,296 \). In order to meet this requirement, the maximum number of pulses for a certain dimension has to be found to satisfy this condition. For example, for 8-dimensional vector, \( V(8,36) = 4,066,763,520 \) is slightly less than \( 2^{32} \), but \( V(8,37) \) is larger than \( 2^{32} \). So 36 is the largest number of pulses for the 8-dimensional vector. Similarly, the largest numbers of pulses for the other dimensions can also be calculated in this way, and the corresponding results are shown in Table 5.

### Table 5 - Maximum number of pulses for 32-bit codebook of different dimensions

<table>
<thead>
<tr>
<th>Dimension</th>
<th>Maximum pulses</th>
<th>Dimension</th>
<th>Maximum pulses</th>
</tr>
</thead>
<tbody>
<tr>
<td>2</td>
<td>128</td>
<td>32</td>
<td>7</td>
</tr>
<tr>
<td>4</td>
<td>128</td>
<td>16</td>
<td>12</td>
</tr>
<tr>
<td>6</td>
<td>88</td>
<td>24</td>
<td>9</td>
</tr>
<tr>
<td>8</td>
<td>36</td>
<td>32</td>
<td>7</td>
</tr>
</tbody>
</table>
To simplify the calculation in the algorithm, the number of pulses is upper limited by 128, which is the reason that the maximum numbers of pulses for 2-dimensional and 4-dimensional are truncated to 128. However not all the integers are valid to be the number of pulses, another variable \( q \) representing quantization quality is used to lower its resolution when \( K \) is larger than 16. Table 6 shows the mapping between \( q \) and \( K \).

Table 6 - Mapping between quantization quality \( q \) and the number of pulses \( K \)

<table>
<thead>
<tr>
<th>( q )</th>
<th>( K )</th>
<th>( q )</th>
<th>( K )</th>
<th>( q )</th>
<th>( K )</th>
<th>( q )</th>
<th>( K )</th>
</tr>
</thead>
<tbody>
<tr>
<td>1</td>
<td>1</td>
<td>11</td>
<td>11</td>
<td>21</td>
<td>26</td>
<td>31</td>
<td>60</td>
</tr>
<tr>
<td>2</td>
<td>2</td>
<td>12</td>
<td>12</td>
<td>22</td>
<td>28</td>
<td>32</td>
<td>64</td>
</tr>
<tr>
<td>3</td>
<td>3</td>
<td>13</td>
<td>13</td>
<td>23</td>
<td>30</td>
<td>33</td>
<td>72</td>
</tr>
<tr>
<td>4</td>
<td>4</td>
<td>14</td>
<td>14</td>
<td>24</td>
<td>32</td>
<td>34</td>
<td>80</td>
</tr>
<tr>
<td>5</td>
<td>5</td>
<td>15</td>
<td>15</td>
<td>25</td>
<td>36</td>
<td>35</td>
<td>88</td>
</tr>
<tr>
<td>6</td>
<td>6</td>
<td>16</td>
<td>16</td>
<td>26</td>
<td>40</td>
<td>36</td>
<td>96</td>
</tr>
<tr>
<td>7</td>
<td>7</td>
<td>17</td>
<td>18</td>
<td>27</td>
<td>44</td>
<td>37</td>
<td>104</td>
</tr>
<tr>
<td>8</td>
<td>8</td>
<td>18</td>
<td>20</td>
<td>28</td>
<td>48</td>
<td>38</td>
<td>112</td>
</tr>
<tr>
<td>9</td>
<td>9</td>
<td>19</td>
<td>22</td>
<td>29</td>
<td>52</td>
<td>39</td>
<td>120</td>
</tr>
<tr>
<td>10</td>
<td>10</td>
<td>20</td>
<td>24</td>
<td>30</td>
<td>56</td>
<td>40</td>
<td>128</td>
</tr>
</tbody>
</table>

The results of the mapping between quantization quality and the number of bits can be calculated based on the method mentioned at the beginning of this part and the relation between \( q \) and \( K \). The Appendix part gives these results for vectors of different dimensions.

### 5.3.3 Range coding

The indices of the vectors of normalized spectral coefficients are entropy coded by range coding, which is introduced in section 2.3.2. Considering the complexity of the algorithm, not all the indices are entropy coded, and actually some bits from the indices are transmitted transparently to the decoder (see 5.3.3.1 and 5.3.3.2). These raw bits are packed at the end of the bit-stream of each frame, see Figure 18.
Figure 18 - Structure of the bit-stream of normalized spectral coefficients in one frame. Range coding bits are packed from the start of the bit-stream, while raw bits are placed at the end of the bit-stream.

The usage of the bits saved by range coding will be described in 5.3.4.

5.3.3.1 Range encoder

The indices of $N$-dimensional vectors of normalized spectral coefficients encoded by PVQ of $K$ pulses are assumed to be uniformly distributed from 0 to $V(N,K)$-1. So the three-tuple $(ft[k], fh[k], fi)$ of an index of PVQ on normalized spectral coefficients has the relation

$$f[k] = fh[k] - ft[k] = 1$$  \hspace{1cm} (34)

and

$$ft = V(N,K) \leq 2^{32} - 1$$  \hspace{1cm} (35)

Therefore the three-tuple of a PVQ index can be simplified by $(t,t+1,ft)$, where $t$ represents the index to be coded and satisfies $0 \leq t < ft$.

Assume an index $t$ is to be range coded, and $n$ is the number of bits to store the value $(ft-1)$ in two’s complement notation. If $n$ is 8 or less, index $t$ is encoded fully by range coding with the three-tuple $(t,t+1,ft)$. Otherwise, only the top 8 bits of $t$ are encoded by range coding with three-tuple $(t >> (n-8), (t >> (n-8)) + 1, ((ft-1) >> (n-8)) + 1)$, and the remaining lower bits are just packed at the end of the bit-stream without any processing.

5.3.3.2 Range decoder

The decoding process is introduced in section 2.3.2.2, and we use $rc\_decode$ to denote the function of decoding. Before a vector is decoded, the number of dimensions of the vector and the number of available pulses are known as $(N,K)$ according to the information from the bit-allocation process and other related modules. Therefore the size of the codebook for this vector can be calculated by $ft = V(N,K)$.

Let $n$ denotes the number of bits to store the value $(ft-1)$ in two’s complement notation. If $n$ is 8 or less, the index $t$ can be decoded by $rc\_decode(ft)$ and the internal states are update with the three-tuple $(t,t+1,ft)$. Otherwise, the top 8 bits of the index $t$ are decoded with $t = rc\_decode(((ft-1) >> (n-8)) + 1)$, and the decoder state is updated with the three-tuple.
\((t, t + 1, ((ft - 1) >> (n - 8)) + 1)\). The remaining bits of index \(t\) can be decoded from the raw bits packed at the end of the bit-stream.

### 5.3.4 Bit saved by range coding

The number of bits allocated and the number of bits used may be different in the shape encoding procedure because of the range coding applied after PVQ.

The difference is accumulated to the “balance” that is responsible for adjusting the allocation for the following bands. So the bits that are used for each band consists two parts: the bits from the bit-allocation procedure and one third of the balance. The only exception is that for the band before the last band and the last band, half of the balance and the whole balance are used as the second part. This process is illustrated in Figure 19.

![Figure 19 - Utilization of the bits saved in entropy coding. The bits allocated for one band are consumed in PVQ, and the remainder is stored in “Balance” for the following bands to use. The arrows represent the flows of bits.](image)

### 5.4 Bit-allocation schemes

The current bit-allocation scheme in the standard G.719 well matches FLVQ, but it may cause some problems when used in the other two versions of codec. The output of PVQ is likely to be too sparse if the input vectors are not assigned to enough bits, and the sparse spectrum of audio signals may cause a bad subjective experience. In addition, a scalar quantizer with low bit-rate provides a limited number of quantization levels, which may lead to a terrible performance.

In order to solve the problem of mismatching between the original bit-allocation scheme and the two quantizers (PVQ and SQ) under study, a new bit-allocation scheme (to be verified) is proposed by constraining low bit-allocation for SQ and PVQ. The principle of the new bit-allocation scheme is the same as the original one except that the new scheme does not allow allocating 1 bit per coefficient. That means in the new scheme, sub-vectors are supposed to be allocated either no bits or at least 2 bits per sample. The new bit-allocation scheme decreases the
use of very low bit-rate PVQ and SQ, which might possibly bring better results. A comparison of these two schemes is given in the next chapter.
6 Results

This chapter shows the performances of quantization schemes in objective and subjective evaluation. All the implementation details that may influence experimental results in this chapter are described in Chapter 5.

6.1 Objective evaluations

6.1.1 Fixed bit-allocation coding

First we compare the SEGSNR of 8-dimensional unconstrained vector quantizer (UCVQ), SQ and FLVQ when their bit-rates are given. As we know, UCVQ is the optimal vector quantization. Therefore we can see how close the FLVQ and SQ are to the UCVQ and decide if it is worth spending time on improving these two quantization schemes. The input of these three quantizers is normalized MDCT-domain spectral coefficients of an about 5-min audio signal sampled at 48 kHz. The input audio file includes about one dozen clips of voices or instruments or both, and each clip has a length ranging from several seconds to half a minute. All the normalized spectral coefficients are quantized using the same amount of bits in one quantization process.

For the 8-dimensional UCVQ one has to create the codebook before it is used. The codebook of the 8-dimensional code-vectors is trained by the General Lloyd-Max Algorithm algorithm [40] and its size $M$ is a power of two, ranging from 2 to $2^8$. All the spectral coefficients are grouped to form 8-dimensional vectors and an initial codebook can be acquired by randomly selecting $M$ vectors from them. Then the initial codebook is trained iteratively. In each iteration, all the 8-dimensional vectors of spectral coefficients search for the nearest code-vector in $L_2$ norm and are assigned to the corresponding group. The total quantization distortion of vectors quantized by UCVQ with the current codebook is recorded. Then each code-vector in the codebook is updated to the average of the vectors belonging to its group. The iteration will stop if the difference between the current total quantization distortion and the previous one is smaller than a specified threshold. The threshold here is set to $10^{-5}$ of current distortion. After all 16 UCVQ codebooks are well trained, the actual quantization starts. The sizes of the codebooks and the corresponding SEGSNRs are recorded. The bit-rate of a UCVQ is the binary logarithm of the size of its codebook.

The scalar quantization scheme here is the combination of the uniform scalar quantizer and arithmetic coding [40]. The uniform scalar quantizer also needs a trained codebook, and the training method is the same as that described in 5.2.1. The scalar quantizer is followed by arithmetic coding to exploit the redundancy among the encoding indices of code-vectors. The number of each code-vector used is counted and these counts are used as the parameters of arithmetic coding.

FLVQ is relatively easy to evaluate because we can utilize the G.719 reference code to get the results. We enforce the bit-allocation algorithm to assign the same number of bits to each normalized spectral coefficient and fetch the output just after the FLVQ decoder.
According to Figure 20, UCVQ has an obvious advantage over the other two quantizers. It is because UCVQ can exploit the redundancy among samples in one vector and has a better capacity of space filling than scalar quantization.

We also find that the combination of scalar quantization and arithmetic coding has a similar performance to that of FLVQ. This result may be a little surprising because usually a vector quantizer has better performance than a scalar quantizer under the same condition. But in this case the FLVQ scheme in G.719 is not perfect. First, Huffman coding following FLVQ is only performed on the indices of vectors which are assigned 2-5 bits per coefficients. In addition, the entropy coding applied only exploits the redundancy of the samples in the same position of vectors, ignoring that among samples in the same vector. The imperfection is the trade-off between the complexity and the performance of the algorithm. A Huffman coding scheme that can remove the redundancy among samples in one vector needs a huge table including the mapping of all the indices of code-vectors. This mapping table inevitably leads to larger storage space and more search time.

6.1.2 Flexible bit-allocation coding

In this part we compare SEGSNR of G.719 with FLVQ, PVQ and SQ at the bit-rate of 32Kbps and 64Kbps. The results also cover comparisons with the new bit-allocation scheme (see 5.4) for PVQ and SQ. Considering SEGSNR is an objective measuring criterion, we did not apply
perceptual model and spectrum filling technique. The input audio file is the same as that used in 6.1.1. Table 10 and 11 shows the results for 32 and 64 kbps.

Table 7 – SEGSNR (dB) of different audio codec at the bit-rate of 32Kbps (the word “original” or “new” mean that the audio codec apply original or new bit-allocation scheme, similarly hereafter)

<table>
<thead>
<tr>
<th></th>
<th>G.719+FLVQ</th>
<th>G.719+PVQ original</th>
<th>G.719+PVQ new</th>
<th>G.719+SQ original</th>
<th>G.719+SQ new</th>
</tr>
</thead>
</table>

Table 8 – SEGSNR (dB) of different audio codec at the bit-rate of 64Kbps

<table>
<thead>
<tr>
<th></th>
<th>G.719+FLVQ</th>
<th>G.719+PVQ original</th>
<th>G.719+PVQ new</th>
<th>G.719+SQ original</th>
<th>G.719+SQ new</th>
</tr>
</thead>
<tbody>
<tr>
<td>SEGSNR</td>
<td>27.5192</td>
<td>30.5908</td>
<td>30.2162</td>
<td>27.8989</td>
<td>27.9848</td>
</tr>
</tbody>
</table>

According to Table 7 and Table 8, we can see that in both bit-rates, PVQ has the highest SEGSNR of the different variants. The reason is that the indices of code-vectors used in PVQ are subject to uniform distribution when quantizing normalized MDCT-domain coefficients, and the entropy coding in PVQ, range coding, utilizes this information to remove redundancy among these indices. But FLVQ in G.719 does not fully exploit this kind of redundancy.

In addition, we find that FLVQ has a similar performance to that of SQ, which may result from the same fact: entropy coding scheme after FLVQ does not fully exploit the redundancy of the input signal.

Finally we notice that the new scheme of bit-allocation algorithm does not improve SEGSNR for PVQ, and this conclusion may be used to analyze the results of subjective tests.

6.2 Subjective evaluations

In the subjective evaluation, namely MUSHRA, 10 music (including castanets, glockenspiel, kick-drum, percussion, orchestra, trumpet, pop, etc) and 10 speech (5 male and 5 female in different languages) clips are prepared and their outputs from three versions of audio codec at bit-rate of 32kbps and 64kbps are collected. For each music or speech clip, the listeners, which are composed of eight Ericsson researchers, evaluate and give scores for five versions of the clip, including the outputs of the three versions (FLVQ, PVQ and SQ) of audio codec, a hidden reference (lossless quality) and an anchor, a 7 kHz low-pass version of the reference. All versions of clips are scored on a scale from 0 to 100.

The tests were done with both the original G.719 bit-allocation scheme and the new version described in section 5.4.
6.2.1 Original bit-allocation scheme

6.2.1.1 Music

Figure 21 - MUSHRA results for music with the original bit-allocation scheme at 32kbps (left) and 64kbps (right). (Ref represents the score for the lossless audio signal and Anch7 is the score for the low-pass signal with the cut-off frequency of 7kHz. Error bars reflect the score ranges of different listeners. These terms also apply to the following related figures.)

Figure 21 shows the results for music clips at the bit-rate of 32Kbps and 64Kbps with the original bit-allocation scheme. We find that SQ performs worse than both VQ schemes. This might be due to the fact that music signals often have a wider spectrum which makes the bit distribution spread out over more bands and allocate a lower number of bits in each band. Therefore SQ quantizes coefficients in a vector with very few levels at low bit-rates and this might lead to its output sounds noisy if it is done on large portions of the signal. However, the VQ schemes can still focus on the highest energy coefficients in a vector, even at low bit-rates.

6.2.1.2 Speech
Figure 22 - MUSHRA results for speech with the original bit-allocation scheme at 32kbps (left) and 64kbps (right)

In the Figure 21 and Figure 22, it seems that PVQ has a slightly small edge over FLVQ on music and speech at both bit-rates.

Figure 23 - MUSHRA results of 10 speech clips with the original bit-allocation scheme at 32kbps (left) and 64kbps (right). The left five materials are from female speakers, and the right five ones are from male speakers in each plot.

According to the results for speech clips (Figure 22 and Figure 23), we see that SQ performs well on speech. It is equivalent with PVQ and FLVQ at 32kbps and better at 64kbps. Besides, SQ seems to have an advantage over the other two VQ schemes for male speakers.

6.2.2 New bit-allocation scheme
Figure 24 - MUSHRA results for music (top) and speech (below) with the new bit-allocation scheme at 32kbps (left) and 64kbps (right)

Figure 24 shows a series of MUSHRA results with the new bit-allocation scheme and we can compare it with the original scheme. We notice that the new bit-allocation algorithm does not improve the performance of SQ or PVQ at any bit-rates, and in fact it makes the PVQ perform worse on music. This means that our assumption that a sparse PVQ outcome may cause degradation does not seem to hold.
7 Discussion

The original bit-allocation scheme matches FLVQ in G.719, but it may not be optimal for PVQ and SQ. In all schemes integer numbers of bits are assigned to a vector, which may limit the performance of PVQ and SQ. Due to the mapping between the number of bits and pulses is not from integers to integers, it is not reasonable to assign integer numbers of bits to a vector in PVQ. Actually, PVQ needs a higher resolution (like 1/8 bit) bit-allocation scheme to make use of bits in a more efficient way. The similar problem also happens to SQ. Because of the integer number of bits allocated scheme, the size of a SQ codebook must be the powers of two. However, for low bit-rate case, non-powers of two, like 3, 7 or 11 may be a more appropriate size for SQ codebook.

A further adjustment of norms based on perceptual models of human hearing in the bit-allocation algorithm and the spectrum filling algorithm for non-coded spectral coefficients in G.719 are both optimized for FLVQ, which inevitably gives FLVQ an extra advantage over the other two quantization schemes when they are compared in the framework of audio codec G.719.

When the bit-rate is low, the output vectors of PVQ are often too sparse and SQ has too few quantization levels. In order to solve this problem, one new bit-allocation scheme is proposed (see 5.4). According to Chapter 6, the SEGSNR of the PVQ version of audio codec decreases when the original bit-allocation scheme is replaced by the new one. Moreover, in the subjective evaluation the new bit-allocation algorithm does not improve the performance of SQ and PVQ at any bit-rate. On the contrary, it makes PVQ perform worse on music. Therefore we think that the low-energy sub-band is worth quantizing because it improves the objective evaluation results, though it leads to sparse outcomes at the same time.

Next, some shortcuts applied in the implementation are summed up to see how they exert an impact on the audio codecs.

The entropy coding actually is not implemented in SQ version of audio codec, and instead 20% of bits for shape quantization are used to compensate the performance loss. The number of bits compensated is estimated by simulations done in advance and the result is an average value through vast amount of experiments. In general case, the compression efficiency is not stable all the time. An entropy coding may have a much higher compression ratio than its average sometimes and in the other occasions it may have a rather lower one or even generates longer bit-stream than the input. In the implementation of SQ version of audio codec, a constant compression ratio of entropy coding is given, which may lead to stable but impractical effects on the subjective tests.

PVQ has a slightly small edge over FLVQ on music and speech at both bit-rates, according to the results shown in the previous chapter. However, considering the shortcut described in 5.3.1, such a slight difference cannot mean the superiority of PVQ. More bits may be needed for PVQ to refine the quantization of its norms, so a complete PVQ version of audio codec would suffer from a little performance degradation after the shortcut is replaced with a practical solution.
8 Conclusion

In this thesis, the perceptual impact of FLVQ, PVQ and SQ is studied in the framework of the audio codec G.719. The study of the three quantization schemes is on 3 levels: how they perform (1) on normalized MDCT coefficients at fixed rates;
(2) when they are applied to the framework of G.719 in the objective view;
(3) when they are applied to the framework of G.719 in the subjective view.

Corresponding programs and simulations are reasonably designed based on these targets and some important conclusions are summarized as follows:

(1) FLVQ and SQ have lower SEGSNR than the optimal vector quantization (UCVQ) at fixed rates, and the margin achieves about 3dB when the bit-rate is 2 bits/sample.

(2) The audio codec G.719 with PVQ outperforms that with the other two quantization schemes at bit-rate 32Kbps and 64Kbps in the objective evaluation.

(3) In subjective evaluation, FLVQ and PVQ are better than SQ when quantizing normalized MDCT coefficients for music signals in the framework of G.719. But SQ performs well on speech. MUSHRA scores of SQ are close to those of PVQ and FLVQ at 32kpbs but beat them at 64kpbs. Besides, SQ has an obvious advantage when quantizing spectral coefficients of male speakers’ speech, especially at the bit-rate of 64kpbs.
Reference


[16] Comparison of lossless codecs on the FLAC’s website ([http://flac.sourceforge.net/comparison.html](http://flac.sourceforge.net/comparison.html))


Appendix

In PVQ, the mappings between quantization quality (q, see 5.3.2) and the number of needed bits (with resolution of 1/8 bit) in vectors of different dimensions are shown below.

Table 9 - Conversion between quantization quality and the corresponding number of bits needed in a 2-dimensional vector in PVQ. The “q” represents quantization quality (see 5.3.2), and the number of bits needed is counted with a unit of 1/8 bit, similarly hereafter.

<table>
<thead>
<tr>
<th>q</th>
<th>1</th>
<th>2</th>
<th>3</th>
<th>4</th>
<th>5</th>
<th>6</th>
<th>7</th>
<th>8</th>
</tr>
</thead>
<tbody>
<tr>
<td>1/8 bit</td>
<td>15</td>
<td>23</td>
<td>28</td>
<td>31</td>
<td>34</td>
<td>36</td>
<td>38</td>
<td>39</td>
</tr>
<tr>
<td>q</td>
<td>9</td>
<td>10</td>
<td>11</td>
<td>12</td>
<td>13</td>
<td>14</td>
<td>15</td>
<td>16</td>
</tr>
<tr>
<td>1/8 bit</td>
<td>41</td>
<td>42</td>
<td>43</td>
<td>44</td>
<td>45</td>
<td>46</td>
<td>47</td>
<td>47</td>
</tr>
<tr>
<td>q</td>
<td>17</td>
<td>18</td>
<td>19</td>
<td>20</td>
<td>21</td>
<td>22</td>
<td>23</td>
<td>24</td>
</tr>
<tr>
<td>1/8 bit</td>
<td>49</td>
<td>50</td>
<td>51</td>
<td>52</td>
<td>53</td>
<td>54</td>
<td>55</td>
<td>55</td>
</tr>
<tr>
<td>q</td>
<td>25</td>
<td>26</td>
<td>27</td>
<td>28</td>
<td>29</td>
<td>30</td>
<td>31</td>
<td>32</td>
</tr>
<tr>
<td>1/8 bit</td>
<td>57</td>
<td>58</td>
<td>59</td>
<td>60</td>
<td>61</td>
<td>62</td>
<td>63</td>
<td>63</td>
</tr>
<tr>
<td>q</td>
<td>33</td>
<td>34</td>
<td>35</td>
<td>36</td>
<td>37</td>
<td>38</td>
<td>39</td>
<td>40</td>
</tr>
<tr>
<td>1/8 bit</td>
<td>65</td>
<td>66</td>
<td>67</td>
<td>68</td>
<td>69</td>
<td>70</td>
<td>71</td>
<td>71</td>
</tr>
</tbody>
</table>

Table 10 - Conversion between quantization quality and the corresponding number of bits needed in a 4-dimensional vector in PVQ.

<table>
<thead>
<tr>
<th>q</th>
<th>1</th>
<th>2</th>
<th>3</th>
<th>4</th>
<th>5</th>
<th>6</th>
<th>7</th>
<th>8</th>
</tr>
</thead>
<tbody>
<tr>
<td>1/8 bit</td>
<td>23</td>
<td>39</td>
<td>51</td>
<td>60</td>
<td>67</td>
<td>73</td>
<td>79</td>
<td>83</td>
</tr>
<tr>
<td>q</td>
<td>9</td>
<td>10</td>
<td>11</td>
<td>12</td>
<td>13</td>
<td>14</td>
<td>15</td>
<td>16</td>
</tr>
<tr>
<td>1/8 bit</td>
<td>87</td>
<td>91</td>
<td>94</td>
<td>97</td>
<td>100</td>
<td>102</td>
<td>105</td>
<td>107</td>
</tr>
<tr>
<td>q</td>
<td>17</td>
<td>18</td>
<td>19</td>
<td>20</td>
<td>21</td>
<td>22</td>
<td>23</td>
<td>24</td>
</tr>
<tr>
<td>1/8 bit</td>
<td>111</td>
<td>115</td>
<td>118</td>
<td>121</td>
<td>124</td>
<td>126</td>
<td>129</td>
<td>131</td>
</tr>
<tr>
<td>q</td>
<td>25</td>
<td>26</td>
<td>27</td>
<td>28</td>
<td>29</td>
<td>30</td>
<td>31</td>
<td>32</td>
</tr>
<tr>
<td>1/8 bit</td>
<td>135</td>
<td>139</td>
<td>142</td>
<td>145</td>
<td>148</td>
<td>150</td>
<td>153</td>
<td>155</td>
</tr>
<tr>
<td>q</td>
<td>33</td>
<td>34</td>
<td>35</td>
<td>36</td>
<td>37</td>
<td>38</td>
<td>39</td>
<td>40</td>
</tr>
<tr>
<td>1/8 bit</td>
<td>159</td>
<td>163</td>
<td>166</td>
<td>169</td>
<td>172</td>
<td>174</td>
<td>177</td>
<td>179</td>
</tr>
</tbody>
</table>
Table 11 - Conversion between quantization quality and the corresponding number of bits needed in a 6-dimensional vector in PVQ.

<table>
<thead>
<tr>
<th>q</th>
<th>1</th>
<th>2</th>
<th>3</th>
<th>4</th>
<th>5</th>
<th>6</th>
<th>7</th>
<th>8</th>
</tr>
</thead>
<tbody>
<tr>
<td>1/8 bit</td>
<td>28</td>
<td>49</td>
<td>65</td>
<td>78</td>
<td>89</td>
<td>99</td>
<td>107</td>
<td>114</td>
</tr>
<tr>
<td>q</td>
<td>9</td>
<td>10</td>
<td>11</td>
<td>12</td>
<td>13</td>
<td>14</td>
<td>15</td>
<td>16</td>
</tr>
<tr>
<td>1/8 bit</td>
<td>120</td>
<td>126</td>
<td>132</td>
<td>136</td>
<td>141</td>
<td>145</td>
<td>149</td>
<td>153</td>
</tr>
<tr>
<td>q</td>
<td>17</td>
<td>18</td>
<td>19</td>
<td>20</td>
<td>21</td>
<td>22</td>
<td>23</td>
<td>24</td>
</tr>
<tr>
<td>1/8 bit</td>
<td>159</td>
<td>165</td>
<td>171</td>
<td>176</td>
<td>180</td>
<td>185</td>
<td>189</td>
<td>192</td>
</tr>
<tr>
<td>q</td>
<td>25</td>
<td>26</td>
<td>27</td>
<td>28</td>
<td>29</td>
<td>30</td>
<td>31</td>
<td>32</td>
</tr>
<tr>
<td>1/8 bit</td>
<td>199</td>
<td>205</td>
<td>211</td>
<td>216</td>
<td>220</td>
<td>225</td>
<td>229</td>
<td>232</td>
</tr>
<tr>
<td>q</td>
<td>33</td>
<td>34</td>
<td>35</td>
<td>-</td>
<td>-</td>
<td>-</td>
<td>-</td>
<td>-</td>
</tr>
<tr>
<td>1/8 bit</td>
<td>239</td>
<td>245</td>
<td>251</td>
<td>-</td>
<td>-</td>
<td>-</td>
<td>-</td>
<td>-</td>
</tr>
</tbody>
</table>

Table 12 - Conversion between quantization quality and the corresponding number of bits needed in an 8-dimensional vector in PVQ.

<table>
<thead>
<tr>
<th>q</th>
<th>1</th>
<th>2</th>
<th>3</th>
<th>4</th>
<th>5</th>
<th>6</th>
<th>7</th>
<th>8</th>
</tr>
</thead>
<tbody>
<tr>
<td>1/8 bit</td>
<td>31</td>
<td>55</td>
<td>75</td>
<td>91</td>
<td>105</td>
<td>117</td>
<td>128</td>
<td>138</td>
</tr>
<tr>
<td>q</td>
<td>9</td>
<td>10</td>
<td>11</td>
<td>12</td>
<td>13</td>
<td>14</td>
<td>15</td>
<td>16</td>
</tr>
<tr>
<td>1/8 bit</td>
<td>146</td>
<td>154</td>
<td>161</td>
<td>168</td>
<td>174</td>
<td>180</td>
<td>185</td>
<td>190</td>
</tr>
<tr>
<td>q</td>
<td>17</td>
<td>18</td>
<td>19</td>
<td>20</td>
<td>21</td>
<td>22</td>
<td>23</td>
<td>24</td>
</tr>
<tr>
<td>1/8 bit</td>
<td>200</td>
<td>208</td>
<td>215</td>
<td>222</td>
<td>229</td>
<td>235</td>
<td>240</td>
<td>245</td>
</tr>
<tr>
<td>q</td>
<td>25</td>
<td>-</td>
<td>-</td>
<td>-</td>
<td>-</td>
<td>-</td>
<td>-</td>
<td>-</td>
</tr>
<tr>
<td>1/8 bit</td>
<td>255</td>
<td>-</td>
<td>-</td>
<td>-</td>
<td>-</td>
<td>-</td>
<td>-</td>
<td>-</td>
</tr>
</tbody>
</table>

Table 13 - Conversion between quantization quality and the corresponding number of bits needed in a 12-dimensional vector in PVQ.

<table>
<thead>
<tr>
<th>q</th>
<th>1</th>
<th>2</th>
<th>3</th>
<th>4</th>
<th>5</th>
<th>6</th>
<th>7</th>
<th>8</th>
</tr>
</thead>
<tbody>
<tr>
<td>1/8 bit</td>
<td>36</td>
<td>65</td>
<td>89</td>
<td>110</td>
<td>128</td>
<td>144</td>
<td>159</td>
<td>173</td>
</tr>
<tr>
<td>q</td>
<td>9</td>
<td>10</td>
<td>11</td>
<td>12</td>
<td>13</td>
<td>14</td>
<td>15</td>
<td>16</td>
</tr>
<tr>
<td>1/8 bit</td>
<td>185</td>
<td>196</td>
<td>207</td>
<td>217</td>
<td>226</td>
<td>234</td>
<td>242</td>
<td>250</td>
</tr>
</tbody>
</table>
Table 14 - Conversion between quantization quality and the corresponding number of bits needed in a 16-dimensional vector in PVQ.

<table>
<thead>
<tr>
<th>q</th>
<th>1</th>
<th>2</th>
<th>3</th>
<th>4</th>
<th>5</th>
<th>6</th>
<th>7</th>
<th>8</th>
</tr>
</thead>
<tbody>
<tr>
<td>1/8 bit</td>
<td>39</td>
<td>71</td>
<td>99</td>
<td>123</td>
<td>144</td>
<td>164</td>
<td>182</td>
<td>198</td>
</tr>
</tbody>
</table>

<table>
<thead>
<tr>
<th>q</th>
<th>9</th>
<th>10</th>
<th>11</th>
<th>12</th>
<th>-</th>
<th>-</th>
<th>-</th>
<th>-</th>
</tr>
</thead>
<tbody>
<tr>
<td>1/8 bit</td>
<td>214</td>
<td>228</td>
<td>241</td>
<td>253</td>
<td>-</td>
<td>-</td>
<td>-</td>
<td>-</td>
</tr>
</tbody>
</table>

Table 15 - Conversion between quantization quality and the corresponding number of bits needed in a 24-dimensional vector in PVQ.

<table>
<thead>
<tr>
<th>q</th>
<th>1</th>
<th>2</th>
<th>3</th>
<th>4</th>
<th>5</th>
<th>6</th>
<th>7</th>
<th>8</th>
</tr>
</thead>
<tbody>
<tr>
<td>1/8 bit</td>
<td>44</td>
<td>81</td>
<td>113</td>
<td>142</td>
<td>168</td>
<td>192</td>
<td>214</td>
<td>235</td>
</tr>
</tbody>
</table>

<table>
<thead>
<tr>
<th>q</th>
<th>9</th>
<th>-</th>
<th>-</th>
<th>-</th>
<th>-</th>
<th>-</th>
<th>-</th>
<th>-</th>
</tr>
</thead>
<tbody>
<tr>
<td>1/8 bit</td>
<td>255</td>
<td>-</td>
<td>-</td>
<td>-</td>
<td>-</td>
<td>-</td>
<td>-</td>
<td>-</td>
</tr>
</tbody>
</table>

Table 16 - Conversion between quantization quality and the corresponding number of bits needed in a 32-dimensional vector in PVQ.

<table>
<thead>
<tr>
<th>q</th>
<th>1</th>
<th>2</th>
<th>3</th>
<th>4</th>
<th>5</th>
<th>6</th>
<th>7</th>
<th>-</th>
</tr>
</thead>
<tbody>
<tr>
<td>1/8 bit</td>
<td>47</td>
<td>87</td>
<td>123</td>
<td>155</td>
<td>184</td>
<td>212</td>
<td>237</td>
<td>-</td>
</tr>
</tbody>
</table>