Comparative Analysis of Scalable Audio Coding Algorithms

Universidade do Minho

Luís M. R. Guimarães

June 2022

Abstract

The goal of this research was to compare different scalable audio coding algorithms/techniques. Different methods were generally studied and compared. Some were observed to provide benefits over others depending on the intended use. Scalable audio coding was observed to be able to offer competitive quality when compared to fixed rate audio coding and more modern algorithms than the ones studied might be able to provide even better performance/quality.

1 Introduction

With the expansion of the digital world, digital audio has become the standard for communication and audio applications. However, these applications still provide restrictions and uncompressed digital audio files tend to be large. There has been a need for digital audio codecs for some time now, and modern audio codecs are extremely efficient and developed.

Codec stands for coder-decoder. A coder (or encoder) takes some data and converts it to a specific format, usually a more convenient one. A decoder "translates" the file back from its compressed (or coded) format. For example, a codec is required in order to convert to the MP3 format as well as to recover the audio signal from the MP3 format.

From lossy (e.g. the popular MP3, AAC) to lossless (e.g. FLAC, Monkey's Audio), different audio codecs might be optimized for different uses. A lossy codec does not have the compromise of having to be able to recover the original signal, which means that there is a bigger window to reduce the file size. A lossless codec on the other hand, is capable of recovering **exactly** the original signal which might be preferable when storing the original signal is the ultimate goal but a smaller file size would still be preferable (e.g. archival). Lossless compression is also preferable when the compressed audio is expected to be post-processed, since post-processing may alter the spectral characteristics making the compression noise (introduced by a lossy encoder) no longer inaudible ([1]).

Technology is not as big of a constraint nowadays in terms of transferring audio files, but very small file sizes (and lossy compression) are still useful today. Music streaming, audio sharing/transferring and internet telephony still benefit from the small sizes that lossy audio codecs can provide and state-of-the art lossy audio codecs can also reduce the file size by magnitudes while maintaining transparent quality, i.e. indistinguishable from the original.

Techniques that allow truncating the coded file at any point, passing it to the decoder and getting a less accurate version of the compressed audio signal have also been developed. This feature is called *scalability* and is particularly interesting for client-server applications in heterogeneous networks, where clients can have diverse and also varying bitrate constraints.

This paper describes the basic ideas of audio coding and scalable audio coding and also different types of techniques used in scalable audio codecs/algorithms. Different scalable audio codecs/algorithms are also presented and compared.

2 Audio Coding

At the most basic level, digital audio signals are represented as binary digits (or bits).
Lossless compression is about organizing a sequence of bits (or a bitstream) in a different way in order to remove redundancy while still being able to retrieve the original sequence. Huffman coding and arithmetic coding are examples of lossless compression methods. Lossless codecs make use of this kind of techniques.
As for lossy codecs, it is assumed that the final destination is the human ear and so psychoacoustic principles are used in order to reduce the file size even further. This is called **perceptual audio coding**. The main purpose of perceptual audio coding is to maximize the perceived quality while minimizing the amount of information required to represent the audio signal.

In this section, basic principles of perceptual audio coding will be explained.

2.1 PCM

PCM stands for Pulse Code Modulation and refers to a way of representing digital signals where an analog audio signal is **sampled**, i.e. converted from continuous to discrete (the amount of samples per second is called the *sample-rate*) and **quantized**, i.e. converted from an infinite range of possible values to a limited one (the range of possible values is given by $2^{bit\text{-}depth}$).

Most modern digital audio formats are encoded in PCM. Figure 1 shows a visual representation of the PCM coding process.

Note that if the sample-rate and the bit-depth are high enough, the decoded signal is basically identical to the original (unlike what the figure might suggest).

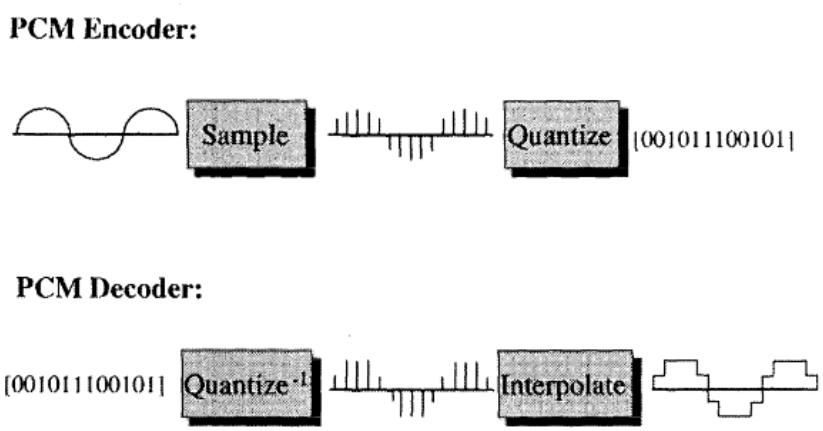

Figure 1: PCM codec (from [2])

2.2 Basic Structure

A perceptual audio codec involves several building blocks. Figure 2 shows a basic structure of an encoder:

- A PCM audio signal goes in.
- In the Time to Frequency Mapping stage, we use a time-frequency transform like the commonly used modified discrete cosine transform (MDCT). This transform allows us to represent the audio signal as a sum of cosine waves at different frequencies with different amplitudes.
- The Psychoacoustic Model stage analyzes the input signal, determines the masking levels, i.e. what frequency components (or coefficients) are inaudible (hence redundant) and sends that information to the Allocation and Coding stage.
- In the Allocation and Coding stage, we first do the bit allocation in a way to minimize the number of bits used overall. Since this process introduces errors (i.e. quantization noise), it is done in a way that the noise introduced is not perceivable (i.e. it stays below the masking threshold) or is as less perceivable as possible (i.e. more audible/important frequency components are prioritized) depending on the bit budget available. Second, we do noiseless coding which means employing lossless compression techniques on the information we have.
- In the last stage Bitstream Formatting, auxiliary data such as metadata may be added to the bitstream.

Figure 2 shows a basic structure of a decoder which is the inverse process of the encoder except for the Psychoacoustic Model stage.
After the final stage, the decoder outputs the decoded PCM audio signal.

Even though lossy compression involves reducing the objective quality of the audio signal by introducing mathematical errors, subjective (perceptual) quality can still be preserved or suffer very small perceivable changes depending on the codec and the target bitrate. Advanced Audio Coding (AAC) is able to achieve transparent quality (i.e. indistinguishable from the original) at just 64kbps per channel ([2], [3]).

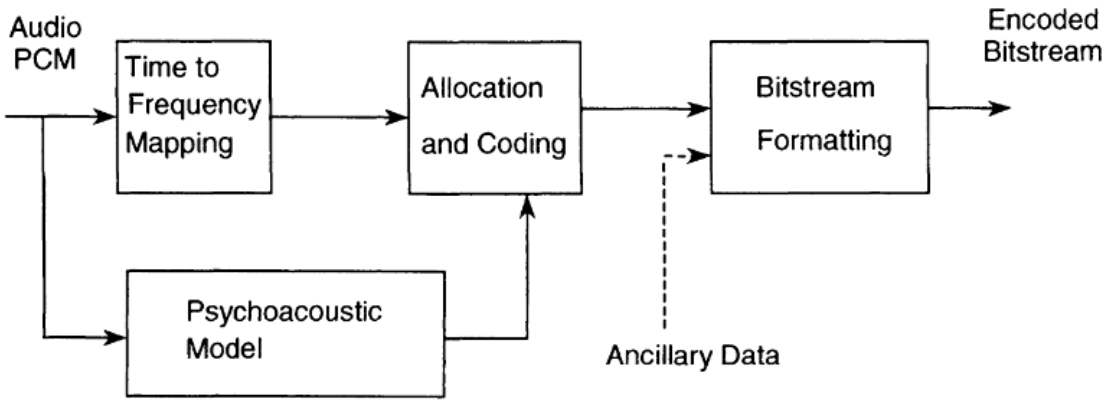

Figure 2: Encoder basic building blocks (from [2])

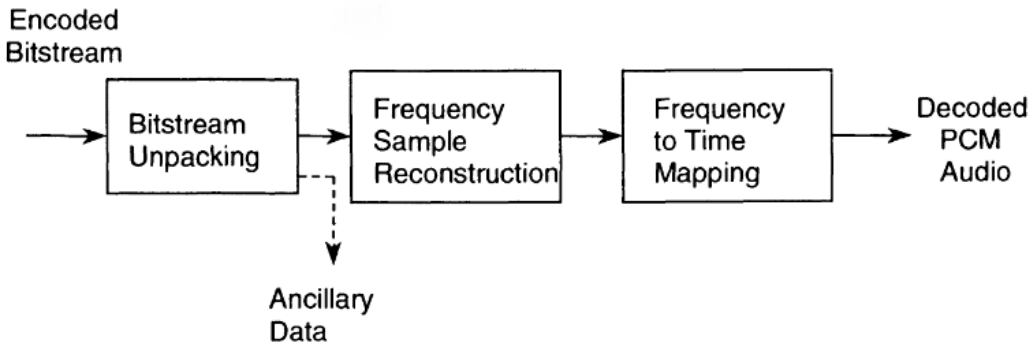

Figure 3: Decoder basic building blocks (adapted from [2])

3 Scalable Audio Coding

In client-server streaming applications that involve audio compression, when the client consumption rate is slower than the server transmission rate, the client asks for a smaller size according to their network limitations. Traditionally, the original audio signal is encoded at different bitrates and the biggest file which doesn't surpass the client's available bandwidth is transmitted. This is called *adaptive bitrate streaming* but requires the server to store several versions of the compressed audio.

Alternatively, a scalable audio codec can be used to provide bitrate scalability, allowing the same compressed file to be truncated at different points, avoiding the need to store several encoded versions of the same audio signal. Scalable audio codecs are organized in layers where a base layer represents the smallest possible quality and every other layer adds detail to the previous ones. Scalability is obtained by sending more or fewer layers to the decoder which determines the quality of the decoded audio signal. The resulting bitstream is referred to as an *embedded bitstream*.

A scalable audio codec can feature *large step* or/and *fine grain* scalability.

3.1 Large Step Scalability

Large step scalability, as the name suggests, means that the step size is relatively large and neighbor layers carry more information that is new.

Large step scalability was implemented in version 1 of MPEG-4 by a method of "cascaded encoding" ([2]). The audio signal is first encoded at the lowest desired bitrate, then at every stage the original signal is encoded at a higher bitrate and the additional precision is added to the bitstream. The codec is told what has already been added to the bitstream so that it can add only the additional detail. Due to the amount of side information required, this implementation cannot efficiently provide small enhancements (i.e. fine grain scalability) ([4]).

Figure 4 shows a block diagram of this approach.

3.2 Fine Grain Scalibility

Fine grain scalability refers to scalability with small steps. In contrast with large step scalability, neighbor layers carry less information that is new.

3.2.1 Bit-plane Coding

Bit-plane (or bit-slice) coding was observed to be commonly used in scalable audio codecs/algorithms that provided fine grain scalability. In this coding scheme, the different **bits** of the quantized coefficients (after the bit allocation stage) are arranged by significance. This way, more significant bits can be posi-

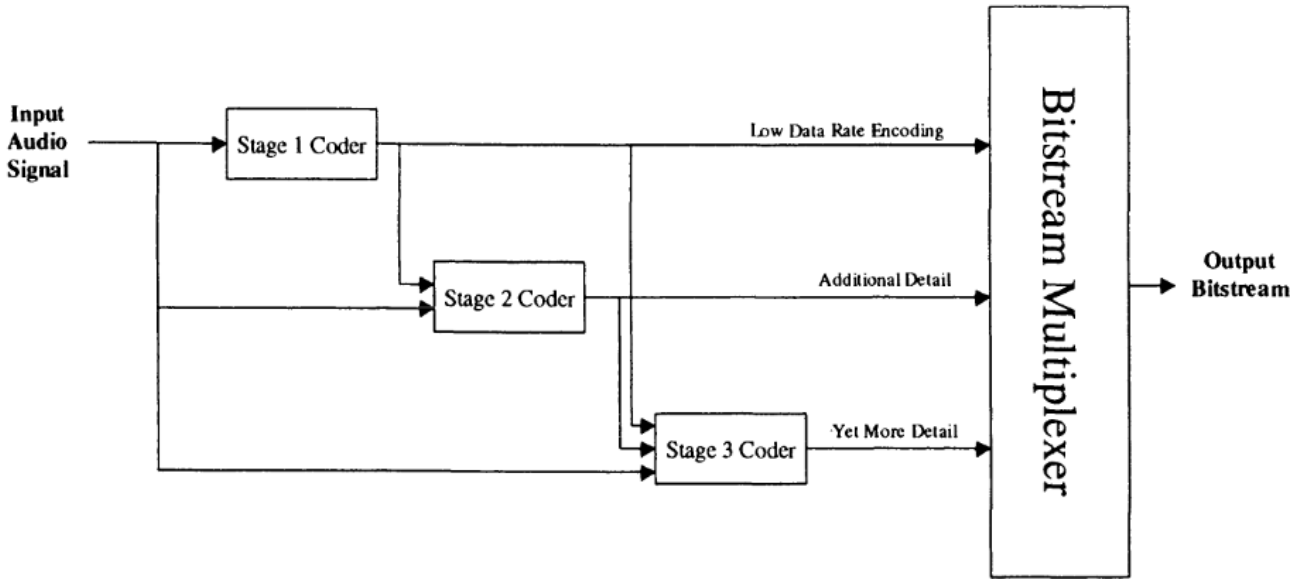

Figure 4: MPEG-4 Audio approach to large step scalable encoding (from [2])

tioned before less significant bits in the resulting bitstream.
Figure 5 provides a visual representation of how **[-1, 5, 14, -3, -12, -1]** can be bit-plane encoded.

	c_0	c_1	c_2	c_3	c_4	c_5
	-1	5	14	-3	-12	-1
Sign	0	1	1	0	0	0
B.P. 3	0	0	1	0	1	0
B.P. 2	0	1	1	0	1	0
B.P. 1	0	0	1	1	0	0
B.P. 0	1	1	0	1	0	1

Figure 5: Bit-planes (from [5])

3.2.2 Huffman Coding

Huffman coding is an entropy coding (also noiseless coding or lossless compression) technique which reduces the amount of bits required to represent a message by assigning more bits to less frequent words and less bits to more frequent ones. The following example is from [6]:

Consider a four-symbol alphabet **{a, b, c, d}** with relative frequencies $\frac{1}{2}$, $\frac{1}{4}$, $\frac{1}{8}$ and $\frac{1}{8}$ respectively. Table 6 shows a possible codeword definition according to the symbols' relative frequencies. Note that the codeword **0** is attributed to **a** and no other codeword starts with 0. This is a result of the *prefix* property present in this coding scheme which means that no codeword is the prefix of another.

Let us look at the decoding process of **0010110** with codewords according to table 6:

Symbol	*Codeword*
a	0
b	10
c	110
d	111

Figure 6: Huffman codewords example (from [6])

- The first symbol is **a** since it is the only symbol whose codeword starts with 0;
- The remaining code is **010110** so another **a** is decoded;
- Next, for **10110** the only codeword beginning with 10 is **b**'s;
- And the remaining **110** corresponds to **c**.

Our code has been decoded to **aabc** and used 7 bits in total which is less than the 8 bits required if we weren't entropy coding.

Huffman coding is able to achieve optimal compression if all the probabilities are negative powers of 2 ([7]) (or integer powers of $\frac{1}{2}$ ([8])).

3.2.3 Arithmetic Coding

An alternative to Huffman coding is arithmetic coding. A problem with Huffman coding is that it never uses less than one bit per symbol. This is specially problematic in models with probabilities close to 1. Arithmetic coding, however, does not have this restriction ([9], [8]).

For the following example, values and figures are from [6]:

Symbol	*Codeword*	*Probability p* (in binary)	*Cumulative probability P*
a	0	.100	.000
b	10	.010	.100
c	110	.001	.110
d	111	.001	.111

Figure 7: Arithmetic coding example (from [6])

Table 7 is an expansion of table 6 which shows the symbol probabilities (p) and cumulative probabilities

(P) in binary. In arithmetic coding, every interval $[P, P+p)$ corresponds to the symbol with the respective probabilities and the interval $[0, 1)$ (includes 0 but not 1) is recursively subdivided as can be seen in figure 8. The encoding result can be any number in the final interval.

The encoding process of **a a b** is as follows (illustrated in figure 8):

- The first **a** corresponds to the interval **[0, .1)**. At this point, our original interval is just subdivided according to symbol probabilities and their cumulative probabilities. Since **a** starts at 0, our interval is going to start at 0 and since our initial width is 1, the end of our interval is calculated by multiplying 1 by the probability of **a** which is .1;
- For the second **a**, the interval $[0, .1)$ is subdivided into the same proportions which means that **a a** corresponds to the interval **[0, .01)** ;
- The last symbol is **b** and if we subdivide $[0, .1)$ we get the value .001 for the end of **a** (which is .01 $\times$ .1). This corresponds to the beginning of our interval and the end is calculated by adding to it our current width (.01) times the probability of **b** which is also .01. This results in .0001 so our final interval is **[.001, .0011)**.
- We need to ensure that the message is decodable. For this example, we are going to encode the message length which is 3 and any number in the interval **[.001, .0011)** results in our message when decoded.

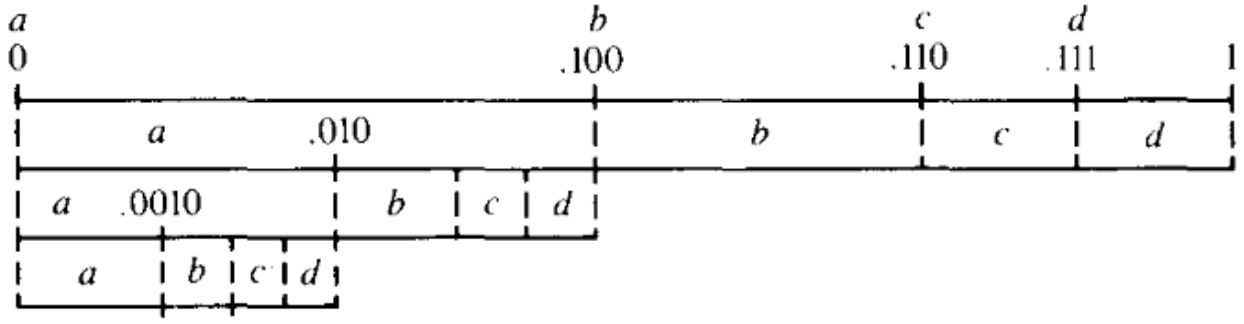

Figure 8: Arithmetic encoding process (from [6])

The information on table 7 is then passed to the decoder and for the value **.001** the decoding process can be done as follows:

- After subdividing **[0, 1)**, our code **.001** is in the interval corresponding to **a** (i.e. $[0, .1)$), so our first symbol is **a**. We can then readjust our code (and "undo" what the encoder did) by subtracting the beginning of our interval, in this case 0, and dividing by the probability of **a** which is .1. We then get **.01**.
- **.01** is more than 0 and less than .1 (where **b** starts) so the next symbol is another **a**. By repeating what we did previously we then get the code **.1**.

- The code **.1** is exactly where **b** starts so our next symbol is **b**.
- Since we got from the encoder the message length 3, the decoding process is complete and the decoded message is **a a b**.

Note that, in this example, had we not encoded the message length, **.001** could also have been decoded as **a a b a** or **a a b a a**, or a similar sequence with an arbitrary number of **a**'s at the end.
Two ways of ensuring decodability are ([9]):

- Encoding the length of the message (as done in the example);
- Using an End-of-File symbol (additionally).

Computers don't provide infinite precision but this coding method can also be implemented using finite-precision ([9], [6]) which has not been studied for this work.

3.2.4 MPEG-4 BSAC

The BSAC (bit-sliced arithmetic coding) scalable codec was introduced in MPEG-4 version 2 and comprises of the AAC codec up to the Huffman coding stage, where the bit-sliced arithmetic coding technique is used instead ([4]). This technique involves grouping the coefficients into frequency bands, slicing the quantized coefficients in bit-planes and coding them using arithmetic coding. Figure 12 shows a visual representation of the process. This allows fine grain scalability since lower bit-planes can be omitted.

BSAC is designed to support scalability with nearly transparent quality at 64kbps per channel ([3]) which is the rate at which AAC is able to achieve transparent quality. A comparison with AAC in terms of subjective quality is done in the Comparisons section.

3.2.5 Tree-based Significance Mapping

In order to achieve efficient bit-plane coding for a larger set of coefficients, it is also advantageous to describe the bit-planes with position information (instead of value information alone). This becomes particularly interesting for sparse data, i.e. when most of the coefficients are zero, since a large number of coefficients can be encoded as a single **0**.

A way of doing this is called tree-based significance mapping where the coefficient position information is mapped to node-location in the tree domain. A significance tree is generated by ordering all coefficients in the form of trees with the assumption that the coefficients closer to the roots of the trees will usually be more significant (i.e. larger in magnitude) than those at the leaves ([3]). Different trees can result in different levels of compression.

In the case of SPIHT ([10]), which was originally developed for image compression, *spatial orientation trees* are used for the significance mapping of transform coefficients. A *spatial orientation tree* defines spatial similarities between pixels which naturally occur in an image. Its decoding process is similar to

the encoding process, but with *outputs* changed to *inputs*. This means that the decoder is able to recover the coefficient ordering by having the same *execution path*.

3.2.6 DSTQ

The Dynamic Scalable Compression Scheme (DSTQ) ([3]) is a scalable compression algorithm with focus on dynamically variant 1-D signals, such as audio signals, that uses the tree-based significance mapping technique.

In audio coding, the audio signal is commonly divided into frames of fixed length. Unlike an image, the frequency content of an audio signal can drastically change over time, which means that there is no single tree that is able to capture the significance information of all frames equally well. For this reason, in each frame a significance tree is selected from a set of possible trees. If the signal is known *a priori*, this set is constructed using the input signal as the training set, if not, a training set from a database corresponding to the signal class can be used, e.g. for a speech signal, a speech set from a database can be used as the training set.

In [3], it is concluded that an optimal significance tree for a given 1-D coefficient vector can be derived by computing its sorting tree. The algorithm for generating K optimal significance trees is the following:

- First the sorting trees for every frame F are computed. The performance of these trees for every frame is measured by using the lengths of the resulting bitstreams and stored in an $F \times F$ matrix M.
- Then the K optimal significance trees are derived from M by choosing the best performing trees for every frame and then iteratively reducing the number of significance trees to K by substituting, for each frame, their significance trees with less optimal trees (with the smallest performance decrement) that are already being used in other frames.

This set of possible significance trees is then transmitted to the decoder.
This algorithm shares similarities with the SPIHT algorithm but is more appropriate for audio signals. DSTQ has an *initialization* stage, a *sorting pass* stage and a *refinement pass* stage. The encoder works as follows:

- At the *initialization stage*, a significance tree is selected from the set of possible trees and its index is outputted, the highest bit-plane is computed and its number outputted and the list of significant coefficients (LSC) is set as an empty list.
- The sorting pass and refinement pass stages run in a loop, for every bit-plane, until the lowest bit-plane has been processed (or a bit budget has been fully used).
- In the *sorting pass*, significance tests are conducted (by calling the *TreeSignificance* function) and new significant coefficients are added to the LSC.
- In the *refinement pass*, for coefficients in the LSC, except those added in the last sorting pass, their

bit value at the current bit-plane is outputted.

Figure 9 shows a visual representation of the algorithm.
Similarly to the SPIHT algorithm, the decoding process is the same as the encoding process except the *outputs* are changed to *inputs*.

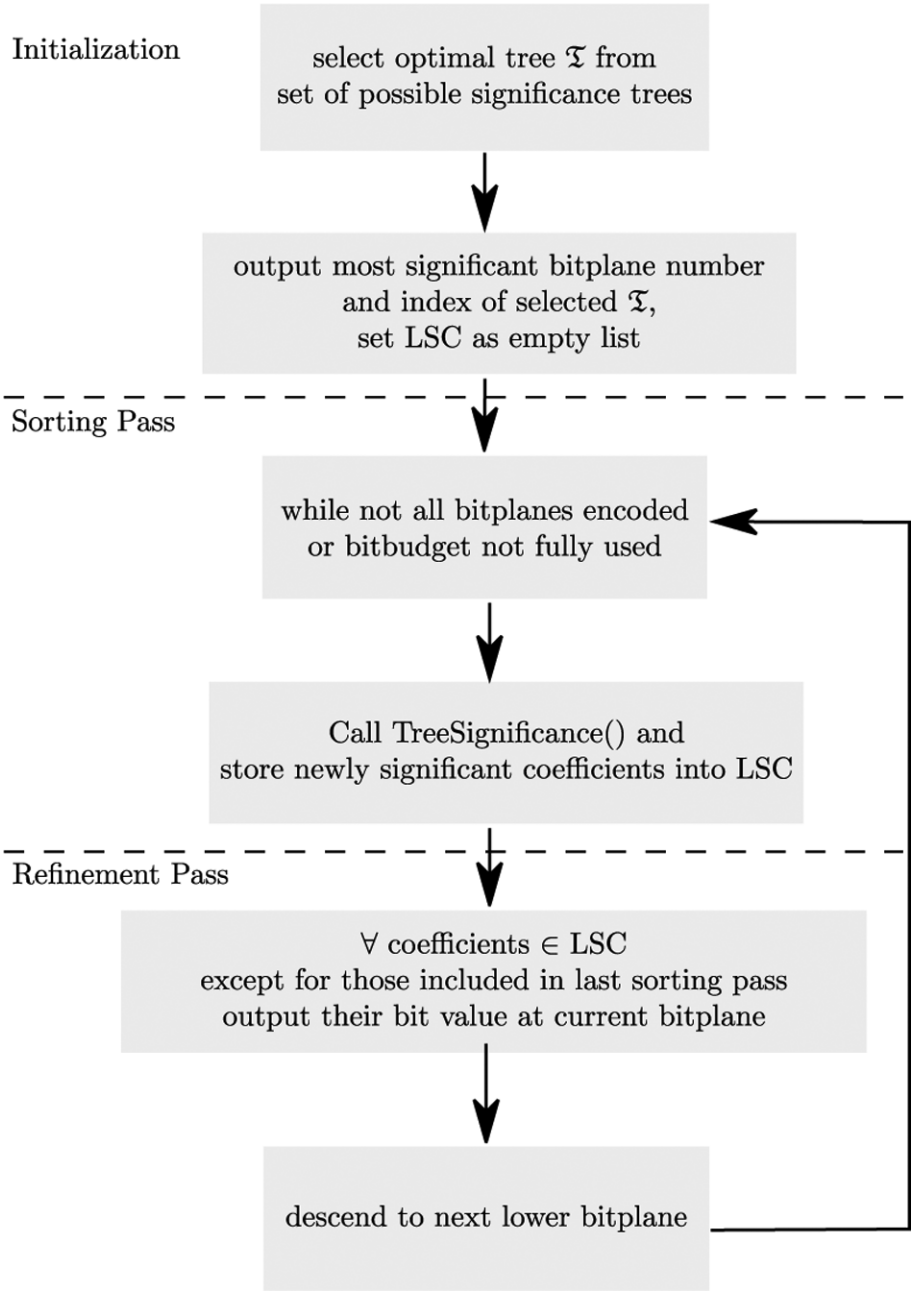

Figure 9: DSTQ algorithm (from [3])

The function that does the significance tests is called *TreeSignificance* and is defined as follows:
For non-leaf nodes/coefficients, its tree is tested for significance and if it is insignificant, i.e. it has no significant coefficient, **0** is outputted. In case it is significant, **1** is outputted and then the node is tested for significance (in the case of a leaf node, it is directly tested for significance). If it is significant, **1** and

sign bit are outputted, otherwise **0** is outputted. The significance test then proceeds to repeat the test for all this node's subtrees.
Note that after a tree or sub-tree tests as insignificant, it is encoded as a single **0** which is an important feature of the tree-based significance mapping technique.

Figure 10 shows a visual representation of *TreeSignificance*[1].

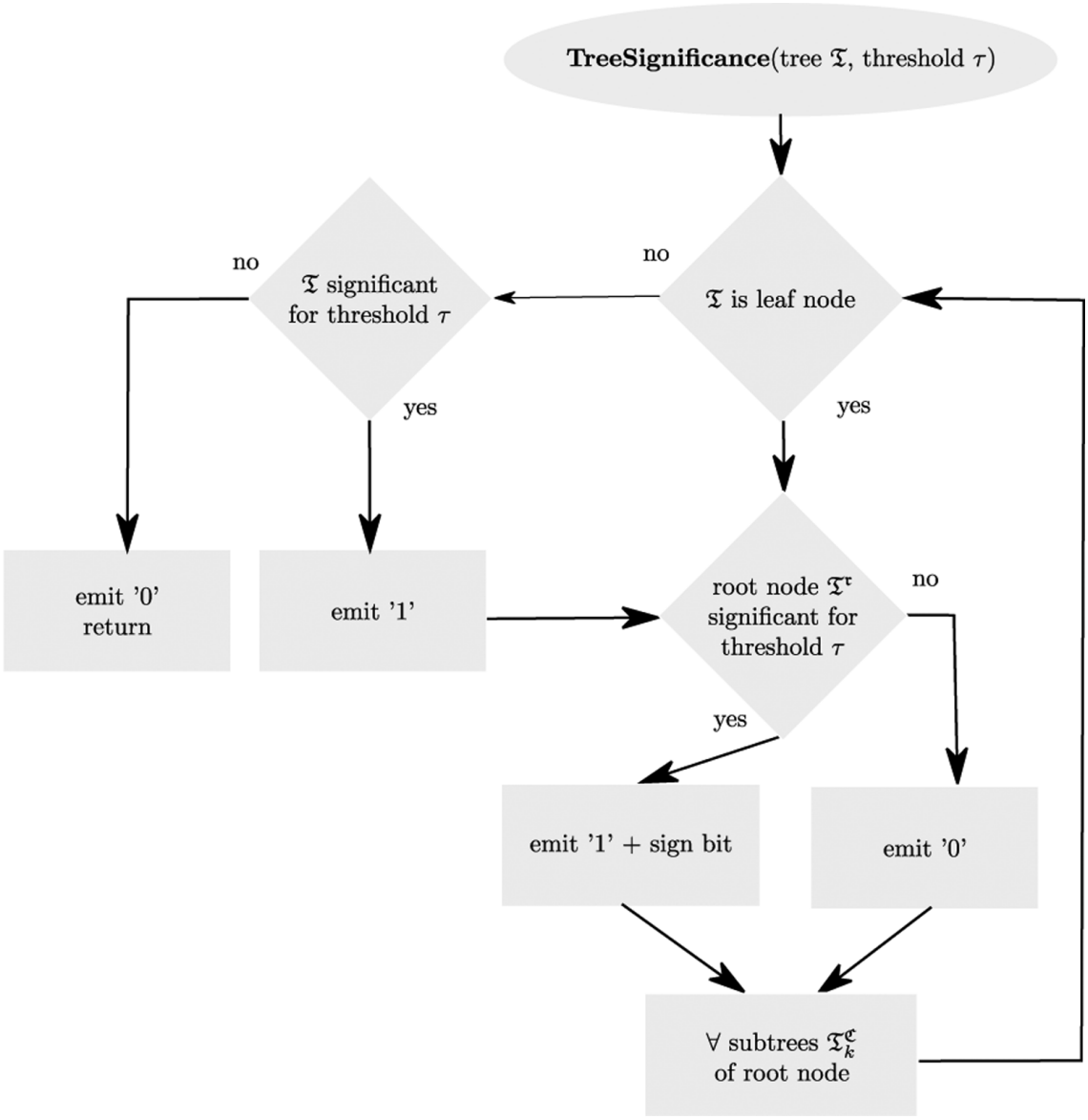

Figure 10: TreeSignificance (from [3])

3.2.7 Lossless Scalable Coding and MPEG-4 SLS

Codecs which provide scalability and exact reconstruction of the original signal were also found, in particular the MPEG-4 Scalable to Lossless Solution(SLS).

[1] In the actual implementation, a modified version is used that only emits test results for nodes that have not become significant in a previous bit-plane.

The IntMDCT (Integer Modified Discrete Cosine Transform) is a lossless transform based on the MDCT which outputs integer values instead of floating point and allows perfect reconstruction ([11]). By excluding floating point errors and allowing perfect reconstruction, the IntMDCT is a more attractive choice for lossless coding.

In the SLS, the IntMDCT is used as the time-frequency transform. SLS has a two-layer structure where the first layer (also the base layer) is an AAC core and the second layer is a lossless enhancement layer (LLE) as figure 11 shows. Having a core layer AAC means that the bitstream is backwards compatible with the AAC codec and also that perceptual audio coding is involved which allows improved quality for bitrates closer to the base layer's.

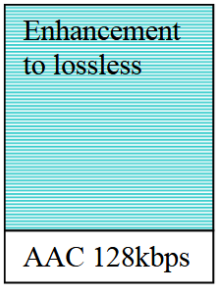

Figure 11: SLS layers (from [1])

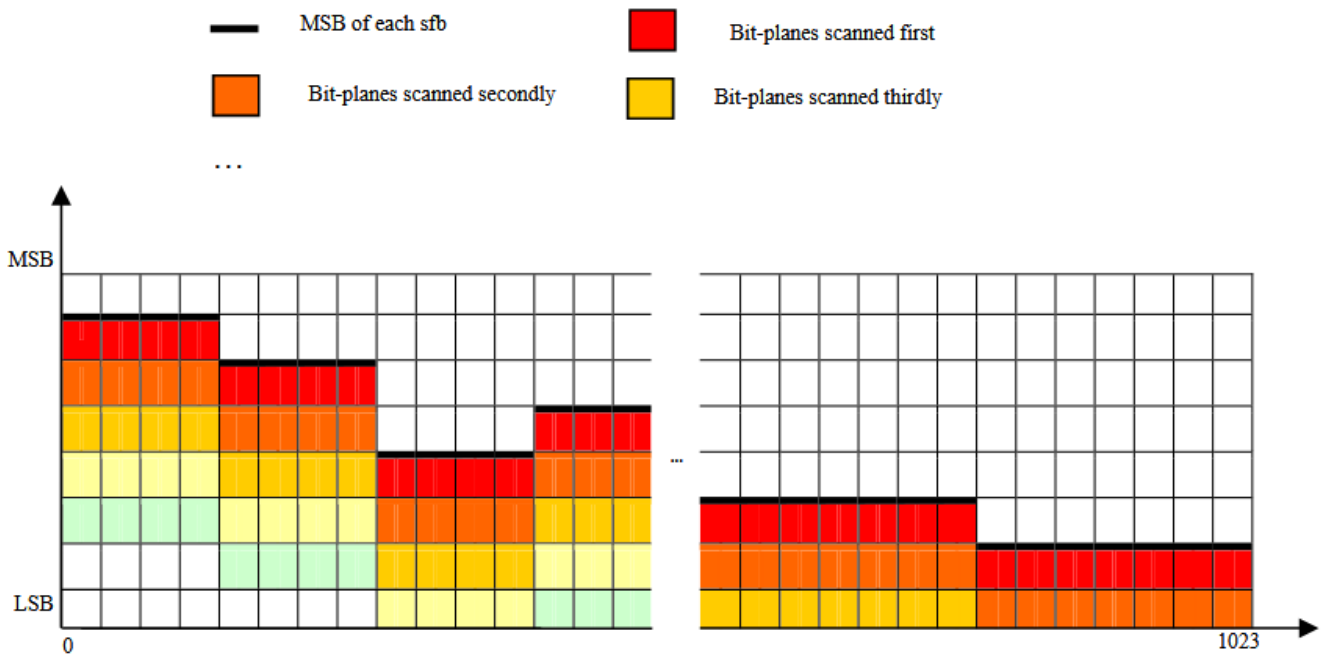

Figure 12: SLS bit-plane order (from [1])

The LLE layer is obtained by subtracting the AAC spectral data from the original spectrum and is then bit-plane encoded[2]. The IntMDCT data of the LLE is divided in bands and the bit-plane coding process

[2]In the actual implementation, more complex techniques are employed which consider different distributions of the co-

starts from the first non-zero bit-plane of every band and progressively moves to lower bit-planes ([1]) as shown in figure 12. This results in the spectral shape of the quantization noise from the core AAC being preserved for lower bitrates and better perceptual the higher the bitrate of the LLE is ([1]).

SLS also has a non-core mode where the core AAC is disabled and the bitstream becomes scalable from 0kbps to lossless. However, by not using a perceptual codec, the spectral shape of the LLE roughly follows the shape of the original signal spectrum which is far from optimal ([12]).

4 Comparisons

Figure 13 shows the results of a comparison test between BSAC and AAC using the method [13]. The BSAC file used for this test was encoded at 96kbps per stereo channel. Note that figure 13 shows the **stereo** bitrate.

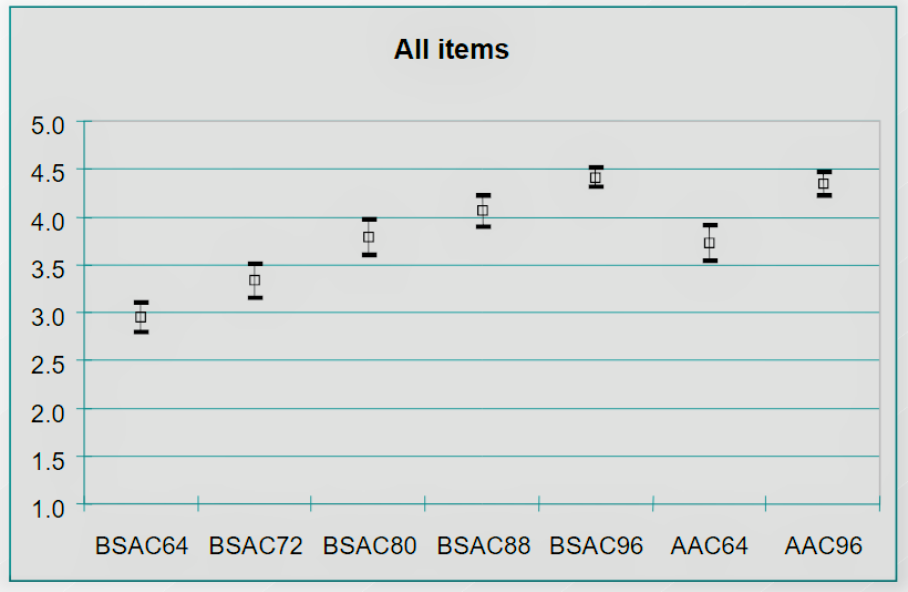

Figure 13: BSAC vs AAC (from [14])

Fixed bitrate codecs like the AAC are optimized differently for different bitrates (using different techniques based on psychoacoustic principles). By simply omitting bit-planes, we obtain a lower quality signal, with more quantization noise, but the spectral shape of the originally encoded signal is preserved which is suboptimal as can be seen in figure 13.

Figure 14 shows quality test results that include the AAC (the Nero implementation) and DSTQ. The DSTQ algorithm was tested by using an AAC codec up to the coding stage, and replacing Huffman coding with the DSTQ algorithm. The tests used the method described in [15]. Note that in terms of evaluation scores, 1.0 and 5.0 in figure 13 correspond to 20 and 100 in figure 14 respectively.

efficients and provide better compression ratios.

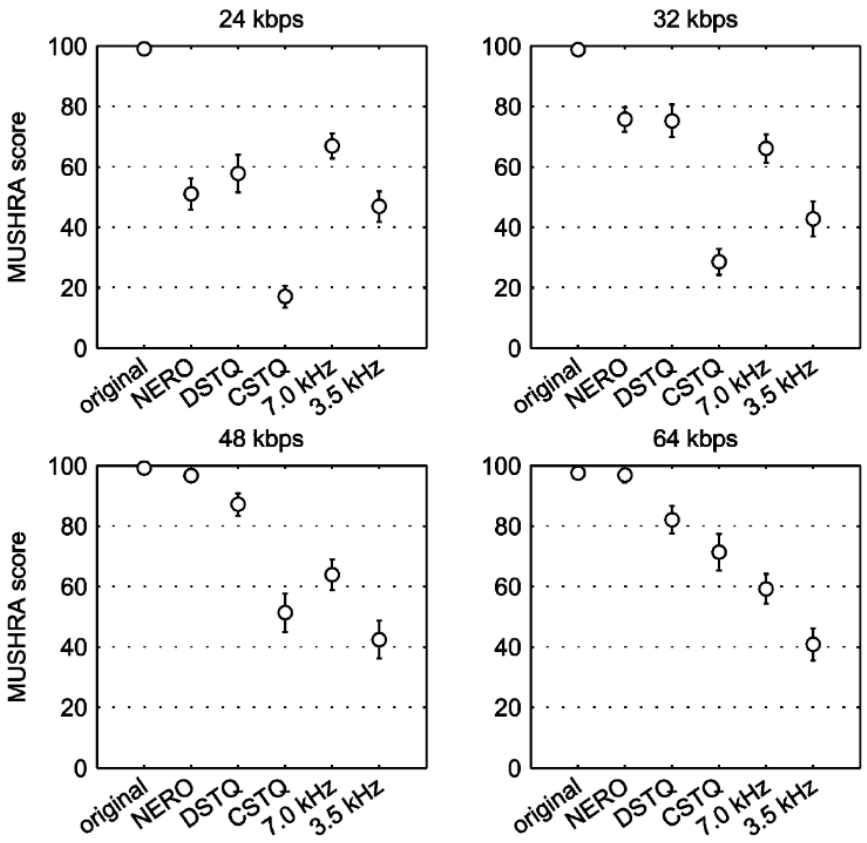

Figure 14: DSTQ vs AAC (NERO refers to the AAC implementation used, CSTQ is an older algorithm by the same authors, 7.0 kHz and 3.5 kHz refer to low-pass filters chosen as anchor points) (from [3])

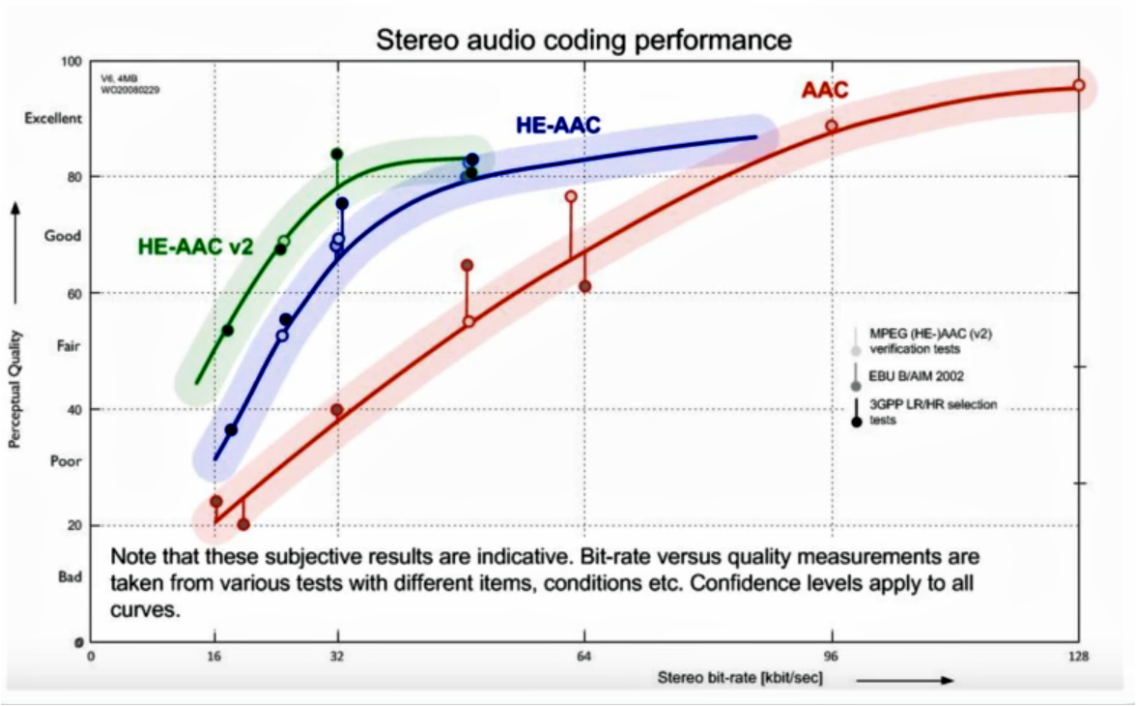

Figure 15: AAC family performance (from [16])

In figure 13 the audio signal was encoded at 48kbps and at that bitrate (untruncated) provided similar quality to AAC. BSAC is also expected to provide similar perceptual quality to AAC at 64kbps. For this reason, BSAC seems to perform slightly better than DSTQ for higher bitrates (48-64kbps).

State-of-the-art fixed bitrate codecs are optimized differently for different bitrates (using different techniques based on psychoacoustic principles) which means that there is a tendency that scalability comes at the cost of perceptual quality. Interestingly, DSTQ showed worse perceptual quality above 32kbps than fixed bitrate AAC, but managed to perform better for bitrates below 32kbps. DSTQ showed very good perceptual quality overall and more recent scalable algorithms/codecs might exist which could perform even better than DSTQ.

SLS performance tests done in [1] show that the SLS can provide competitive performance as a lossless codec while providing fine-grain scalability. While the BSAC technique and the DSTQ algorithm are designed to be used with a perceptual audio codec (which is lossy), the SLS is able to provide lossless compression of the original audio signal and is also able to provide perceptual audio coding for lower bitrates (with the AAC core). Subjective quality tests were not found, but SLS is expected to perform worse than DSTQ for bitrates 24-64kbps in a similar test as 14, with the core AAC at 24kbps. This is because at 24kbps the same quality as the AAC is achieved, but for every increment in bitrate, the spectral shape does not build towards that of the AAC at 64kbps, instead the spectral shape of the core AAC at 24kbps is roughly preserved for lower bitrates and slowly approximates that of the original signal.

Extensions of the AAC have been developed for low bitrate coding which are able to provide better perceptual quality than regular AAC. They include techniques like Spectral Band Replication (SBR), which involve coding the lower frequencies alone and reconstructing the higher frequencies at the decoder stage using perceptually shaped noise derived from the lower frequencies, and also Parametric Stereo which involves encoding the stereo signal as a mono signal plus a set of parameters characterizing the stereo image ([16]).

HE-AAC (High Efficiency AAC) features a combination of SBR and the AAC codec and HE-AACv2 features a combination of Parametric Stereo, SBR and the AAC codec ([16]). Figure 15 shows a perceptual quality comparison between AAC, HE-AAC and HE-AACv2 (again, note that the bitrate corresponds to the stereo bitrate). These extensions are expected to perform better than DSTQ for bitrates below 24kbps (per channel), however, they don't offer scalability and are limited to a smaller range of bitrates.

5 Conclusion

In this paper, different types of scalable audio coding mechanisms were described and compared. The basic ideas of audio coding and scalable audio coding were presented, techniques used in scalable audio coding, in particular bit-plane coding, arithmetic coding and significance trees, were described as well as different codecs/algorithms that make use of these techniques.

Further research for comparing scalable audio codecs/algorithms could include complexity and storage requirements comparisons as well as more modern codecs/algorithms.

6 References

[1] R. Yu, R. Geiger, S. Rahardja, J. Herre, X. Lin, and H. Huang, "MPEG-4 scalable to lossless audio coding," in *Proc. 117th AES conv*, 2004, pp. 1–14.

[2] M. Bosi and R. E. Goldberg, *Introduction to digital audio coding and standards*, vol. 721. Springer Science & Business Media, 2002.

[3] S. Strahl, H. Hansen, and A. Mertins, "A dynamic fine-grain scalable compression scheme with application to progressive audio coding," *IEEE transactions on audio, speech, and language processing*, vol. 19, no. 1, pp. 14–23, 2010.

[4] I. J. N2803, "MPEG-4 audio version 2 (final committee draft 14496-3 AMD1)." 1999 [Online]. Available: https://sound.media.mit.edu/resources/mpeg4/audio/documents/w2803.html

[5] E. Ravelli, G. Richard, and L. Daudet, "Extending fine-grain scalable audio coding to very low bitrates using overcomplete dictionaries," in *2007 IEEE workshop on applications of signal processing to audio and acoustics*, 2007, pp. 195–198.

[6] G. G. Langdon, "An introduction to arithmetic coding," *IBM Journal of Research and Development*, vol. 28, no. 2, pp. 135–149, 1984.

[7] "How huffman trees work." Computerphile, 2013 [Online]. Available: https://www.youtube.com/watch?v=umTbivyJoiI

[8] I. H. Witten, R. M. Neal, and J. G. Cleary, "Arithmetic coding for data compression," *Communications of the ACM*, vol. 30, no. 6, pp. 520–540, 1987.

[9] mathematicalmonk, "(IC 5.2) arithmetic coding - example #1." 2011 [Online]. Available: https://www.youtube.com/watch?v=7vfqhoJVwuc

[10] A. Said and W. A. Pearlman, "A new, fast, and efficient image codec based on set partitioning in hierarchical trees," *IEEE Transactions on circuits and systems for video technology*, vol. 6, no. 3, pp. 243–250, 1996.

[11] R. Geiger, T. Sporer, J. Koller, and K. Brandenburg, "Audio coding based on integer transforms," in *Audio engineering society convention 111*, 2001.

[12] T. Li, S. Rahardja, and S. N. Koh, "Frequency region-based prioritized bit-plane coding for scalable audio," *IEEE transactions on audio, speech, and language processing*, vol. 16, no. 1, pp. 94–105, 2007.

[13] "Methods for the subjective assessment of sound quality - general requirements," *Recommendation ITU-R BS. 1284*, 1996.

[14] R. Sperschneider, F. Feige, and S. Quackenbusch, “Report on the MPEG-4 audio version 2 verification test,” *ISO/IEC JTC1/SC29/WG11*, vol. 3075.

[15] “Method for the subjective assessment of intermediate quality level of coding systems,” *Recommendation ITU-R BS. 1534*, 2001.

[16] S. U. Marina Bosi CCRMA, “Keynote: How perceptual audio coding has shaped our lives.” 2018 [Online]. Available: https://youtu.be/3BHMykq5PTU?t=2606