Entropy Coding of Adaptive Sparse Data Representations by Context Mixing

von

Thomas Wenzel
Matrikel-Nr. 5960073

geboren am
23.07.1987

Masterarbeit im Studiengang Mathematik
Universität Hamburg

2013

Gutachter: Prof. Dr. Armin Iske
Zweitgutachter: Dr. Laurent Demaret

Erklärung

Die vorliegende Arbeit habe ich selbständig verfasst und keine anderen als die angegebenen Hilfsmittel - insbesondere keine im Quellenverzeichnis nicht benannten Internet-Quellen - benutzt. Die Arbeit habe ich vorher nicht in einem anderen Prüfungsverfahren eingereicht. Die eingereichte schriftliche Fassung entspricht genau der auf dem elektronischen Speichermedium.

Abstract

In this thesis improvements on the video compression scheme AT3D, which relies on linear spline interpolation over anisotropic Delaunay tetrahedralizations, are presented by introducing a more sophisticated entropy coding stage and optimizing several intermediate computations to reduce compression time. A detailed literature summary on information theory basics and a detailed implementation summary of AT3D is presented. The original entropy coding stage is replaced by the core of the general purpose data compression scheme PAQ by Matt Mahoney, which employs a geometric context-mixing technique based on online convex programming for coding cost minimization. Novel models for pixel position- and grayscale value encoding are incorporated into the new AT3D_PAQ implementation and investigated numerically. A significant average improvement was achieved on a wide variety of test sequences, and in a comparison with H.264 a sequence class is presented, in which AT3D_PAQ outperforms H.264. Theoretical and numerical results additionally demonstrate the higher computational efficiency of AT3D_PAQ and show significantly reduced compression times in a variety of test environments.

Contents

1 Introduction **3**

1.1 Thesis organization and contributions 6

2 Fundamentals **8**

2.1 Notational conventions 8

2.2 A brief introduction to video compression 8

2.2.1 Summary and outlook 13

2.3 MPEG4-AVC/H.264 14

2.3.1 Overview 14

2.3.2 Coding units 14

2.3.3 Intra- and inter-frame prediction 16

2.3.4 Transform coding 17

2.3.5 Scan, quantization and entropy coding 18

2.3.6 In-loop deblocking filter 18

2.3.7 Rate control 19

2.3.8 Summary 19

2.4 Introduction to information theory 20

2.4.1 Basic definitions 20

2.4.2 Entropy 21

2.4.3 Linking entropy to data compression 25

2.4.4 Arithmetic coding 29

2.4.5 Separation of modeling and encoding 34

2.4.6 Summary 35

2.5 AT3D 36

2.5.1 Delaunay tetrahedralizations 37

2.5.2 Linear splines over Delaunay tetrahedralizations 38

2.5.3 Pre-processing 40

2.5.4 Adaptive thinning 41

2.5.5 Post-processing: Pixel exchange 43

2.5.6 Post-processing: Best approximation 45

2.5.7 Quantization 46

2.5.8 Implementation of arithmetic coding 47

2.5.9 Entropy coding: Pixel positions 47

2.5.10 Entropy coding: Grayscale values 51

2.5.11 Numerical results regarding stationarity 54

3 Development of AT3D_PAQ **55**
3.1 PAQ 56
3.1.1 PAQ8 - Building blocks 57
3.1.2 Context mixing in PAQ1-3 60
3.1.3 Context mixing in PAQ4-6 61
3.1.4 Context mixing in PAQ7-8: Logistic/geometric mixing 64
3.1.5 PAQ8 - Modeling tools 67
3.1.6 PAQ8 - Adaptive probability maps 68
3.2 Integration of PAQ into AT3D 69
3.3 Modifications of PAQ after the integration 70
3.3.1 Modeling pixel positions 71
3.3.2 Modeling grayscale values 75
3.4 Numerical results: AT3D_PAQ vs. AT3D 79
3.4.1 Pixel position encoding results 81
3.4.2 Grayscale value encoding results 85
3.4.3 Total compression results for selected sequences 85
3.4.4 Summary and analysis of overall results 90
3.4.5 Compression time comparison: AT3D_PAQ vs. AT3D 90
3.5 Numerical results: AT3D_PAQ vs. H.264 91
3.5.1 Summary and analysis of results 98
3.6 Conclusion 98
3.7 Outlook 99

4 Runtime optimization **100**
4.1 Improvements of the AT3D implementation 101
4.1.1 Assignment of pixels to tetrahedra 101
4.1.2 Custom queue implementation 117
4.1.3 Further improvements 119
4.2 Numerical results 120
4.3 Conclusion 127
4.4 Outlook 127
4.5 Further code improvements 128

Appendices **129**

A List of used hard- and software **129**

B Test video sequences **130**

C Detailed compression results **133**

D Detailed H.264 results **137**

E Detailed runtime results **139**

1 Introduction

Given a sequence S of symbols from an alphabet $\mathcal{A}$, that is to be stored on a medium or transmitted via a channel, it is natural to ask which is the shortest representation, that allows an exact reconstruction of S, and how to obtain it. This requires the process of lossless data ***compression*** or ***entropy coding***, a transformation of an input sequence into a preferably shorter output sequence. First answers were found by Shannon in 1948, when he created the field of information theory. Shannon introduced the concept of ***entropy*** of a sequence S. Entropy may be measured in bits per symbol and refers to the expected information content per symbol of a sequence with respect to a probability distribution P of the symbols in S. One of the most important results of his work was showing that entropy is a lower bound for the minimal length of an encoded sequence, which still allows exact reconstruction. We will present results from the information theory literature extending Shannon's results to ***sources*** $\mathcal{S}$, which are defined as random processes generating sequences S_i, and their ***entropy rates***, which describe the expected information content per symbol of a sequence emitted by $\mathcal{S}$. Those results show that a lower bound for the expected code-length per symbol of such a sequence S_i exists and that it is given by the entropy rate of $\mathcal{S}$, if $\mathcal{S}$ is stationary. The latter notion refers to the independence of joint probability distributions of arbitrary symbols from their time of emission. Even if $\mathcal{S}$ is not stationary, which is the case in our application, a lower expected code length bound per symbol is given by the ***joint entropy*** of a finite consecutive subsequence of random variables that $\mathcal{S}$ consists of. It is calculated based on the joint probability distribution of those random variables. Unfortunately, in practical applications all of these lower bounds are not computable. The above concepts and results will later be introduced in detail and provide a theoretical background for data compression and lead to a better understanding of our later goals.

We will focus on a specific kind of data to compress: video data. It is a digital representation of a sequence of images. The need for efficient video compression schemes is due to the large sizes of typical video sequences and the variety of fields, where video sequences have to be stored or transmitted. Most video compression schemes, and in particular the ones considered in this thesis, are ***lossy*** schemes, which introduce an error in the reconstructed video data. In 1989 the scheme H.262/MPEG-1 was introduced, which created the basis of today's state-of-the-art general purpose compression scheme ***H.264/MPEG4-AVC***. Both rely on intra-frame compression via a discrete cosine transform (or a closely related transform respectively) and translational inter-frame motion compensation of pixel-blocks. Besides a vast number of small improvements, H.264 additionally introduced an in-loop deblocking filter, which improved the visual quality of its output significantly by reducing the tendency to produce visible block-artifacts. A detailed introduction to the specification of H.264 is given in a dedicated chapter.

In this thesis we focus on the improvement of a different compression scheme, yet we will compare it to H.264 later: ***AT3D***. The latter was developed by Demaret, Iske, and Khachabi, based on the two-dimensional image-compression scheme AT (adaptive thinning). Given data is interpreted as a 3d scalar field with domain X, and the scheme then relies on linear spline interpolation over anisotropic Delaunay tetrahedralizations. The linear spline space is indirectly determined by the greedy iterative adaptive thinning algorithm, which selects a sparse set X_n of ***significant pixels*** from the initial grid by removing one least significant pixel or edge in each iteration. We call a tuple $p = (p_x, p_y, p_z) \in \mathbb{R}^3$ a pixel. The significance of a pixel is determined by simulating its removal from the tetrahedralization and calculating the incurred mean squared interpolation error. The pixels in each set X_k span a unique Delaunay tetrahedralization, which in turn yields a linear spline space, which contains an interpolant and a unique best approximation to the original video data. The removal of a pixel can be efficiently calculated, because it is a local operation only affecting its cell of adjacent tetrahedra, when performing an interpolation. The calculation of the best approximation is a global operation, thus it is only obtained once for the final set X_n. The option to remove two pixels connected by a tetrahedral edge at once further improves the compression. The resulting pixel set is further optimized by a local pixel exchange algorithm before the final calculation of the best approximation on X_n. The output of adaptive thinning is a set of pixels $Y \subset \mathbb{N}_0^3$ and a set E_V, which contains the grayscale values at each of the significant pixels. The theoretical complexity of this whole process is $\mathcal{O}(N \log N)$, where N denotes the number of removed pixels.

Those two sets are then entropy coded. One of the two central results of this thesis will be the improvement of the entropy coding stage of AT3D by replacing it with a general-purpose lossless data compression scheme called ***PAQ***, which was then adapted to the specific characteristics of the adaptive thinning output. The new implementation, which is the result of this thesis, will be denoted by ***AT3D_PAQ***.

In AT3D an arithmetic encoder was used in the entropy coding stage. It compresses a given sequence $S = (s_0, s_1, \dots, s_m)$ of symbols from an alphabet $\mathcal{A} = \{a_0, \dots, a_n\}$ by starting with the interval $I_0 = [0, 1]$ and dividing it into $n + 1$ subintervals I_k with lengths according to the occurrence probabilities of the symbols a_i in the current iteration step. When the whole sequence was processed, the number $x \in I_m$ with the shortest binary fractional representation is chosen and stored. A binary fraction is stored by saving the bit-stream representing its numerator, while its denominator has to be a power of 2. The probabilities used for interval partitioning may vary in the encoding process and are based on the probability distribution model of its source $\mathcal{S}$. In this case, the arithmetic coder is called ***adaptive***. If the source emits stochastically dependent random variables, the use of conditional probabilities is advantageous and leads to better compression results, because they allow modeling the dependencies between random variables without explicitly knowing their joint probability distribution. In this case it is called a ***contextual*** arithmetic coder. It will be shown later, that the compression achieved in our practical application entirely depends on the employed source models, and that a more accurate model directly results in improved compression and vice versa.

In AT3D pixel positions are encoded by iterating through all positions in the original pixel grid row by row and encoding a bit for each position, that is set if it contains a significant pixel. It uses a context-based arithmetic coder for pixel position coding, where contexts are based on significant pixel counts in ***projected 2d-context boxes***. When considering a pixel position y to be encoded, a context box refers to all previously encoded pixel positions x in the same frame as y satisfying $\|y - x\|_\infty < r$. A projected 2d-context box is the projection of y into the first and last frame of a video sequence, where the counts of significant pixels are summed up in the corresponding 2d-context box. Therefore, the first and the last frame of a sequence are encoded first, before encoding the remaining frames in their original order. This is necessary for the decoder to be able to reconstruct the bit-stream. This approach is obviously tailored to a very specific kind of adaptive thinning output, which allows for a significant improvement.
Grayscale values are encoded by an adaptive arithmetic coder. The employed probabilities are directly deducted from the count of a grayscale value's occurrence, which also suggests the possibility for improvements.

One of the major drawbacks of the original AT3D implementation is its limitation to at most one class of contexts. If a video sequence has properties, that are not incorporated into the probability model of that context class, compression will suffer. Therefore, AT3D_PAQ features the integration of the general purpose lossless data compression scheme PAQ, which was originally developed by Mahoney and first presented in [Mah02]. It relies on a technique called ***context mixing***, which combines predictions from numerous context classes into a resulting probability distribution, that in general performs at least as good as the best individual distribution. Besides giving a detailed introduction of PAQ we will describe the context classes we implemented in AT3D_PAQ later.
In our numerical investigations of AT3D_PAQ we will consider a large number of examples, which will demonstrate that our efforts resulted in improved compression for all test sequences. In particular it is remarkable, that this thesis was not focused on improving special cases, but on achieving a general improvement, which was successful. Compression results are improved by 15.04 % on average and by up to 56.06 % in certain test cases. A detailed comparison of AT3D and AT3D_PAQ is presented in a dedicated chapter.
Additionally, we investigated the performance of AT3D_PAQ in comparison to H.264. The results varied significantly for the different test sequences, yielding mixed results. For certain sequence types, H.264 outperforms AT3D_PAQ by far, while for others it depends on the quality settings, and for one class of test sequences we obtained (partially significantly) better results by AT3D_PAQ. For this class of artificially generated geometric sequences, AT3D_PAQ is drastically improved over AT3D, which was not capable of outperforming H.264 for all quality settings.

In the second major part of this thesis we describe our work to improve the runtime of AT3D. Despite its theoretical runtime of $\mathcal{O}(N \log N)$ it suffers from the implicitly given large constant. We applied numerous changes to the implementation, which reduced the number of memory allocations and deallocations significantly. Additionally we reduced the number of geometry tests, which have to be performed to assign a pixel to a tetrahedron,

which is necessary in the significance update of a pixel. Furthermore many minor changes were applied, yielding significant runtime improvements. Later we will provide theoretical evidence for the poor performance of the original assignment algorithm and describe our new implementation in detail.
Finally, detailed numerical results are presented, showing best- and worst-case improvements in different environments and on different microarchitectures. Our improvements result in a runtime reduction of 38.46 % in the worst case and of 48.88 % in the best case.
Additionally we want to note that for the first time a 64-bit implementation of AT3D was considered, which allowed the investigation of significantly larger video sequences than before. The largest considered sequences contain up to 2.4 million pixels, while in a 32-bit implementation sequences with at most 0.8 million pixels could be processed due to memory limitations.

1.1 Thesis organization and contributions

In the first chapter we introduce all concepts mentioned above in detail and sum up all conclusions needed. After fixing notational conventions in section 2.1 we start with setting up the framework we will operate in in section 2.2. There we define video data, video compression scheme notions, and quality measures needed to evaluate the output of a lossy compression scheme. In the following section 2.3 today's state-of-the-art video compression scheme, H.264, is described in detail. We pay particular attention to the features that are new compared to the well-known scheme H.261.
In chapter 2.4 we focus on information theory and provide a detailed summary of available literature and define entropy, entropy rates of sources and present results connecting these notions to data compression. We then introduce the concept of codes, illustrate it by Huffman codes and move on to arithmetic coding, which is one key ingredient in the entropy coding stage of AT3D as well as the one of AT3D_PAQ. We relate the notions from 2.4 to arithmetic coding and present results from the literature which show that arithmetic coding is close to optimal for our purposes. Finally we come to the conclusion, that it is a tool, whose compression performance is entirely dependent on our model development.
After all these literature summaries on topics, that played an important role in our work, we introduce AT3D in detail and for the first time present detailed summaries of the actual implementations of adaptive thinning, pixel exchange and the entropy coding stage in particular. All of these were extracted from the source code by performing a close analysis, which also led to the discovery of implementation errors, which were partly corrected in AT3D_PAQ, but without having a measurable impact on its performance.

In chapter 3 we start focusing on results of this thesis. We introduce PAQ, and the detailed documentation in this thesis extends the rare existing literature based on source code analysis and links mathematical investigations of techniques used in PAQ to the available literature on it in section 3.1. Our main focus is a detailed description of the architecture and the employed mathematical methods, but we also quote some recently published investigations, which for the first time proved theoretical results underlining the

quality of the architecture of PAQ. The following sections 3.2 and 3.3 are dedicated to a summary of our integration of PAQ into AT3D and to describing the architecture of AT3D_PAQ. We then continue with a presentation of our modifications and modeling approaches taken to improve compression.
Finally, we give numerical results in chapter 3.4, comparing AT3D to AT3D_PAQ in a wide variety of test cases, including different sequence types, lengths, and resolutions. We continue with a comparison of those two schemes and H.264 in order to allow a classification of the compression performance of AT3D_PAQ in comparison to a state-of-the-art compression scheme. We summarize our results in section 3.6 and give an outlook on possible further research projects in section 3.7.

Runtime optimization of AT3D is described in chapter 4. In section 4.1 we summarize all modifications applied to AT3D in order to improve its runtime. We begin with our new queue implementation, which reduces the number of memory allocations and deallocations significantly in section 4.1.2. Then we continue with a theoretical investigation of the original algorithm, which assigns pixels to tetrahedra in section 4.1.1. These theoretical investigations, which are a result of this thesis, suggest several ideas for reducing the number of necessary geometry-tests to assign pixels successfully, most importantly by using a technique called tetrahedron-walking. The implementation of these ideas is then described and the properties of walking are summarized from the literature. In section 4.1.3 a number of additional small improvements is listed, which also improved the runtime measurably.
These measurements are then presented in section 4.2, where we first select two benchmark environments based on a small selection of numerical results and then continue with the presentation of the average improvements in a best-case scenario and in a worst-case scenario. Results are then summarized in section 4.3 and some ideas for further improvements, which were out of the scope of this thesis, are presented in section 4.4. In appendix A the software used in this thesis is listed, appendix B contains a listing of all test sequences with a short summary of the contents of each scene. In the following appendix C detailed compression results comparing AT3D and AT3D_PAQ are listed. Additionally tables containing the numerical results of the comparison to H.264 are presented in appendix D. The results regarding the runtime investigation are listed in appendix E. All figures and illustrations are drawn and developed by the author of the thesis, unless indicated otherwise.

2 Fundamentals

2.1 Notational conventions

In this section we introduce some notational conventions that will be used in the remainder of this thesis.

Remark 2.1.1.

- Let $\mathbb{N}$ be the set of natural numbers starting at 1. Let $\mathbb{N}_0 = \mathbb{N} \cup \{0\}$.
- A sequence is denoted by $(c)_{k\in\mathbb{Z}}$, or $(c)_k$. A specific element of the sequence at index j is referred to by c_j. Note that the index may be negative.
- An equivalence, which is expressed by the term "if and only if" is abbreviated by "iff".
- For any vector $p \in \mathbb{R}^3$ its x-, y- and z-components are denoted by p_x, p_y and p_z.
- A set $\{m \in \mathbb{N}_0 \mid m \leq n \in \mathbb{N}_0\}$ will be denoted by $\underline{n}$.
- In listings of algorithms we will use a notation close to program code, allowing statements of the type $m := m + 1$, where convenient.

2.2 A brief introduction to video compression

Video compression has been subjected to research for several decades now. In this section, we will sum up the emerged standard notions necessary for the description of a video compression scheme, and fix, where this thesis fits into this large framework.
We will start with the formalization of the structure of video data, continue with the structure of a video compression scheme, and introduce the most common quality measure for compressed video sequences, called PSNR, which is also of great importance in AT3D.

The notion of a signal is a very general one, therefore we will start with a more specific definition that suits our purposes:

Definition 2.2.1 (Signal)**.** A ***signal*** f is a mapping

$$f : T \to R,$$

where $T \subset \mathbb{R}$ is a set of points in time and R is an arbitrary set of values.

- The signal f is called ***continuous***, if $T = [a, b]$ for $a, b \in \mathbb{R}$.
- It is called ***discrete-time signal***, if T is a countable set. Then the numbers $(f)_n := f(nT_s), n \in \mathbb{Z}$ are called ***samples*** of f with ***sampling interval*** T_s. Its inverse $f_s := \frac{1}{T_s}$ is called ***sampling frequency***. Signals sampled at a varying frequencies are not considered.

Remark 2.2.2. Note that the notion of sampling only refers to the discretization of a continuous signal. When trying to reconstruct a continuous signal from a discrete one, the ***Nyquist-Shannon sampling theorem*** formulates necessary conditions on the sampling frequency to be able to fully reconstruct the sampled continuous signal. Violating this necessary condition can lead to sampling artifacts.
In the scope of this thesis we will not consider these reconstruction problems and therefore refer to [Bla02] for further information on this subject.

Also note that the range of a discrete-time signal can still be an uncountable set. In this case signal processing on a computer would not be possible, because an infinite number of bits may be needed to save the signal. Therefore we need the following definition:

Definition 2.2.3 (Quantizer)**.** Let R be a range of values. Then a ***quantizer*** Q is a mapping

$$Q : R \to S, \qquad \text{with } |R| \geq |S| = s \in \mathbb{N}.$$

In particular S is a finite set.

Definition 2.2.4. Let f be a continuous signal and Q be a quantizer. Then a ***digital signal*** $(f^Q)_n$ is the sequence related to the discrete-time signal $(f)_n$ by

$$f_n^Q := Q(f_n), \qquad \text{with } n \in \mathbb{Z}.$$

In the following we will consider digital signals only.

Example 2.2.5. An ***audio signal*** f_A is a signal with $R = \mathbb{R}$, where the absolute values $|f_A(t)|$ with $t \in T$ represent the amplitude of an audible tone. This amplitude results from the superposition of several frequencies in the signal.

Video sequences can be defined in several different ways depending on the compression scheme to be used. The most common definition of a video sequence is the following (note that we will not use this approach in AT3D):

Definition 2.2.6 (Grayscale video sequence)**.** Let $W, H \in \mathbb{N}$ be a width and a height and let $f : T \to \mathbb{R}_+^{W \times H}$ be a continuous signal. Let $Q : \mathbb{R}_+^{W \times H} \to \underline{2^\rho - 1}^{W \times H}$, be a quantizer, where $\rho \in \mathbb{N}$. Then the grayscale values in one frame of the video sequence $(f^Q)_{n\in\mathbb{Z}}$ are defined by

$$f_n^Q := Q(f(nT_s)) \text{ for } n \in \mathbb{Z}.$$

Then the k-th ***frame*** of a sequence may be defined as

$$F_k := \left\{ \left(i, j, (f_k^Q)_{ij}\right) \in \underline{W-1} \times \underline{H-1} \times \underline{2^\rho - 1} \right\}.$$

A ***pixel*** is a tuple (i, j, k). Both W and H are fixed throughout the sequence. The size of a frame is called ***resolution of the video sequence***. The values from $\underline{2^\rho - 1}$ represent grayscale values, with $\rho = 8$ in general. $\frac{1}{T_s}$ is called ***frame-rate*** of the video sequence.

Remark 2.2.7. Note that following this definition a sample of a natural image sequence is a frame, not a pixel.

Another common approach is the natural extension of the mathematical approach of defining audio sequences as uni-variate mappings, images as bi-variate mappings and video sequences as tri-variate mappings. We will use it for AT3D:

Definition 2.2.8 (Grayscale video sequence as trivariate mappings)**.** Let f be a mapping satisfying

$$f : \mathbb{R}_+^2 \times \mathbb{R} \to R \subset \mathbb{R}_+.$$

Let Q be a quantizer satisfying

$$Q : R \to \underline{2^\rho - 1}.$$

The corresponding digital signal is defined by

$$f_{ijn}^Q := Q(f(i, j, nT_s)), \quad \text{with } 0 \le i < W \text{ and } 0 \le j < H, n \in \mathbb{Z}.$$

The ***frame*** F_k is defined by

$$F_k := \left\{ (i, j, f_{i,j,k}^Q) \in \underline{W-1} \times \underline{H-1} \times \underline{2^\rho - 1} \right\}.$$

We will refer to tuples (i, j, k) as ***pixels*** and to the corresponding grayscale value f_{ijk}^Q as the ***(grayscale) value at a pixel (position)***.

Remark 2.2.9. Note that opposed to the general definition of a video sequence, two spatial dimensions are part of the domain of the mapping and not of the range. When considering an analogous definition for images, there is no temporal component in the domain, thus an image is not time-dependent and can therefore not be defined as a signal in the classical sense mentioned before.

Two major organizations are responsible for issuing standards and so-called recommendations in video compression: The ***International Telecommunication union (ITU)***, founded in 1865, with its subsection ***ITU Telecommunication Standardization Sector (ITU-T)***, and the ***Moving Picture Expert Group (MPEG)***, which was founded

by the ISO in 1988 to define standards for audio and video compression and transmission schemes. Video compression is subject to this standardization process to ensure widespread availability of implementations of efficient compression schemes for video data transmission over networks like the internet.
One family of standards regard the format of color information: These values are coded by a ***RGB-color model***, which models a color space created by the additive combination of three basic colors: red, green, and blue. In order to save bandwidth in the transmission of color data, intermediate formats that represent RGB data by color differences were developed. Examples include ***YUV,*** $\boldsymbol{YP_bP_r}$***,*** $\boldsymbol{YC_bC_r}$ and several others. Their commonality lies in the separation of RGB-colors into ***luma-*** and ***chrominance-***components.
This separation reflects the important property of the human visual system to be more sensitive to differences in brightness than to differences in color.
RGB-colors are first non-linearly transformed into gamma-corrected R'G'B'-colors before calculating the corresponding luma-value by computing the weighted sum of those three R'- G'- and B'-components. The weights may slightly vary depending on the used ITU recommendation (most commonly [ITU11] or [ITU02]). The luma-component contains the brightness information of a color. The remaining two chrominance-components can then be derived from the R'G'B'-color and its luma-component. A common first compression step reduces the amount of color information: ***chroma-sub-sampling***, which is essentially a down-sampling of chroma information.
The employed color model, resolution and frame-rate depend on the used standard, e.g. NTSC, PAL or SECAM, which have also been developed by the organizations mentioned above.

Remark 2.2.10. Note that in the scope of this thesis we will only be interested in grayscale video sequences, which we defined above. The employed video formats only contain data following this definition (`.yuv` and `.pgmv`), thus all regulations imposed by the standards mentioned above are irrelevant for us. These formats do not contain information regarding the frame-rate and in our case no chrominance-, but only luma information.

After fixing the framework regarding input-data, we will now introduce the basic layout of a compression scheme.

Definition 2.2.11 (Video codec)**.** An ***encoder*** is a software program or hardwired circuit that converts an input bit-stream of video data into an encoded output bit-stream. To reconstruct a video a suiting ***decoder*** is needed, which decodes the bit-stream generated by the corresponding encoder, i.e. into the original format. An encoder together with a suitable decoder is called a ***video codec***.
If the input bit-stream is equal to the decoded encoder output bit-stream, the video codec is called ***lossless***. Otherwise it is called ***lossy***.

A codec is an implementation of a video compression scheme. A key property of most encoders is the reduced size of its output bit-stream compared to the input bit-stream. If

the encoding process requires significantly less computational effort than the decoding, a video codec is called ***asymmetric***. Otherwise it is called ***symmetric***. Generally, in an asymmetric codec the encoder is considerably harder to design and implement compared to the decoder.
Compression ratios achieved by lossless codecs are significantly worse than the ones achieved by lossy codecs. High-resolution video data is always encoded by lossy codecs in practical applications.

Definition 2.2.12 (Bit-rate)**.** The ***bit-rate*** of video data is the number of bits needed to save a sequence of frames that is played back within one second. It is measured in bits per second, abbreviated by ***bit/s***.
It is not necessarily constant throughout a video sequence. In this case the bit-rate refers to the average bit-rate of the sequence.

Remark 2.2.13. The bit-rate of a video depends on the used codec. It also depends on the frame-rate of the sequence. A higher bit-rate tends to result in better visual quality when a fixed compression scheme is used.

The goal of video compression is to make use of redundancies in video sequences and to define a scheme to remove those redundancies, and thereby transform the sequence into a shorter representation. We will give an exact definition of redundancy in chapter 2.4, for now we regard redundancies as pieces of information contained in the luma- or color-values at pixel positions, that are also implicitly contained in values of neighboring pixels.
While audio sequences only contain ***temporal*** redundancies, images contain ***spatial redundancies***. When considering video sequences to be composed of frames, note that they contain intra-frame spatial redundancies and inter-frame temporal redundancies. These can be related (e.g. consider a sequence showing a moving box, where spatial redundancies are caused by the structure of the box and temporal redundancies are caused by its motion).
A very simple example of a scheme to remove redundancies from an audio sequence would be the coding of differences of sample values instead of coding the sample values themselves.

When applying a lossy compression scheme, the reconstructed signal will differ from the original one. Since we are considering digital signals, this resulting error can be measured by mathematical means. The most common error measure is the following:

Definition 2.2.14 (Peak signal-to-noise ratio, PSNR)**.** Let $(f^Q)_n$ be a finite digital signal with k samples sampled from a bounded continuous signal f. Let f_{max} be the upper bound of the range of f. Let $\widetilde{(f^Q)}_n$ be the coded and decoded reconstruction of $(f^Q)_n$. Then the ***PSNR*** is defined as

$$\mathrm{PSNR} = 10 \log_{10} \frac{f_{max}^2}{\widetilde{\eta}^2((f^Q)_n, \widetilde{(f^Q)}_n)} \text{ dB},$$

with

$$\widetilde{\eta}^2((f^Q)_n, \widetilde{(f^Q)}_n) := \frac{1}{k}\sum_n \left| f_n^Q - \widetilde{f_n^Q} \right|^2.$$

The ***mean-squared error*** is denoted by $\widetilde{\eta}^2$.

While it is a mathematically simple measure and yields a fairly good estimation of the codec-induced error, it sometimes does not correspond to the perceived error. When the error created by the compression becomes perceptible, it is usually called an ***artifact***. In rare cases reconstructed signals with a large PSNR-value may be considered as signals of poor quality. Conversely, signals with low PSNR-value may be considered as having high quality. Perceived quality is strongly dependent on the perceiving individual. Generally, a value above 40 dB indicates very good visual quality and a value below 30 dB indicates poor visual quality. The type of audible or visible artifacts is usually a characteristic property of a compression scheme, e.g. MPEG compression schemes are known to suffer from block artifacts at low bit-rates.
The design of appropriate quality measures is an important task of today's research. For further information we refer to [Wan03].

The development of efficient (lossy) video compression schemes has been subject to research for the last decades. There were many successful developments, most notably the standards ***MPEG1, MPEG2/H.262, MPEG4*** and ***MPEG4-AVC/H.264***. They all have their usage of a discrete cosine transform for residual coding, their block-based motion estimation and compensation scheme for inter-frame coding, and a final entropy coding procedure in common. MPEG1 introduced those key ingredients, and throughout the years many details in the standard were improved. H.264 is today's benchmark in multi-purpose compression schemes. A detailed introduction of H.264 is presented in section 2.3.

2.2.1 Summary and outlook

In order to link the notions introduced above to the contents of this thesis we may formulate the following summary:

- We consider AT3D, an asymmetric (lossy) video codec which bases its compression on a 3d-interpretation of the given input data. It will be introduced in chapter 2.5.
- We will compare it to the second considered codec, H.264, which is lossy and asymmetric as well and assumes a sequence of 2d images. It is today's benchmark in video compression. We will introduce H.264 in detail in chapter 2.3.
- Input- and output data are grayscale video sequences, which are saved in the `.yuv`- or `.pgmv`-formats and do not contain standardized information regarding frame-rate, resolution or color-information, therefore all playback-related data and formatting is irrelevant for us.
- The comparison between both codecs will be based on the PSNR-value of the encoded and decoded test sequences.

2.3 MPEG4-AVC/H.264

In this section we will introduce MPEG4-AVC/H.264 (abbreviated by H.264 in the remainder of this thesis) in more detail and summarize its properties. Our main focus is the description of the core mechanisms, which contribute to the compression performance most. Since only grayscale video sequences are in the scope of this thesis, we will restrict ourselves to describing the luminance value compression of H.264. Note that for H.264 only the ***decoding*** process is specified. Hence its practical performance is always dependent on the encoder implementation to some extent.

2.3.1 Overview

As a successor of earlier MPEG-standards, H.264 is a transform-based coding scheme, which relies on translational motion compensation and estimation between frames to take advantage of inter-frame redundancies. In general, a frame is predicted first by several mechanisms. Then a residual of the prediction to the original frame is calculated, which is transform-encoded afterwards. Therefore H.264 features very sophisticated prediction mechanisms in order to generate a low-entropy residual. A visualization of the H.264-encoder is provided in figure 1.
In this section, we follow [Pur04] and [Sul05] and also recommend these to the reader for further details.

2.3.2 Coding units

We will now describe the units into which frames are divided for encoding purposes. Note that H.264 uses the video sequence definition given in 2.2.6 and assumes a sequence of still images.

Definition 2.3.1 (Macro-block)**.** A ***macro-block*** is a 16x16 pixel area within a frame which forms the basic building block for compression. It may be partitioned into smaller units in some of the coding steps. The smallest possible sub-block covers a 4x4 pixel area.

Definition 2.3.2 (Slice)**.** ***Slices*** partition frames into sets of macro-blocks and remain fixed across frame boundaries. Thus, a slice can be interpreted as a mask applied to all frames. Each slice is encoded separately. The maximum size of a slice is the full frame size.

Definition 2.3.3 (Slice types)**.** There are five different ***slice types***, which determine the encoding techniques applied to an individual slice. We will only present the three types relevant for our application:

1. ***I-slice***: only intra-slice encoding techniques are applied.

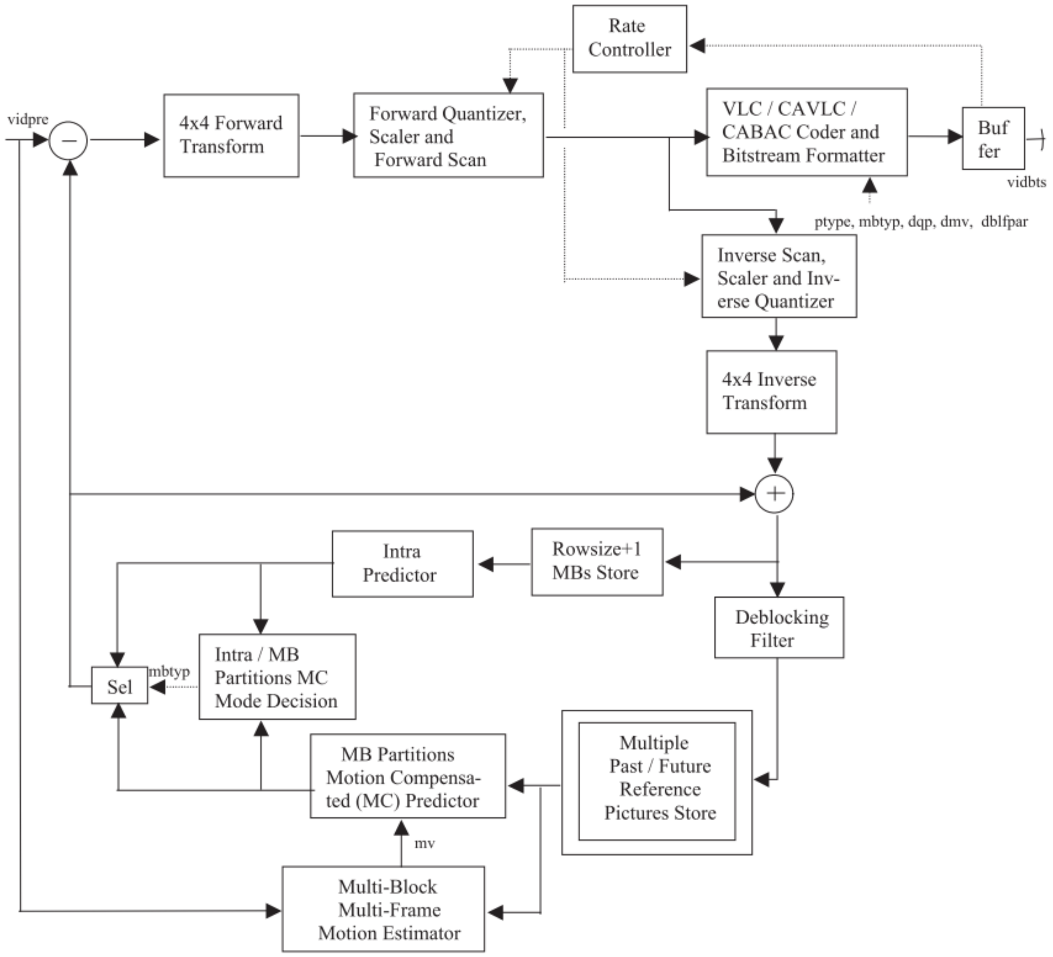
Rate Controller
vidpre
−
4x4 Forward Transform
Forward Quantizer, Scaler and Forward Scan
VLC / CAVLC / CABAC Coder and Bitstream Formatter
Buf fer
vidbts
ptype, mbtyp, dqp, dmv, dblfpar
Inverse Scan, Scaler and Inverse Quantizer
4x4 Inverse Transform
+
Intra Predictor
Rowsize+1 MBs Store
Deblocking Filter
Sel
mbtyp
Intra / MB Partitions MC Mode Decision
MB Partitions Motion Compensated (MC) Predictor
Multiple Past / Future Reference Pictures Store
mv
Multi-Block Multi-Frame Motion Estimator

Figure 1: Overview of the H264 encoding process ([Pur04], p.7).

2. ***P-slice***: in addition to the techniques for I-slices, at most one piece of motion compensated prediction information may be applied per macro-block.
3. ***B-slice***: in addition to the coding types for P-slices, two pieces of motion predicted information may be applied per macro-block by calculating a weighted average.

In H.264, a large number of different macro-block coding modes is supported, whose selection is performed by the encoder. These decisions are coded in a header to make them available to the decoder. In our further summary, we will not describe all options in full detail and instead refer the reader to [Pur04], which provides deeper insight into the design and inner workings of H.264.

2.3.3 Intra- and inter-frame prediction

A variety of intra-frame prediction mechanisms is provided in H.264. Such predictions may either be performed on 4x4-blocks for textured areas or on 16x16-blocks for smooth areas.

Definition 2.3.4 (Prediction)**.** In the following, we will refer to ***prediction*** or ***estimation*** as the process of estimating a coding unit by already processed or encoded data.

The predicted unit is then subtracted from the original unit and the residual is transform-encoded. Conversely, decoding reverses these steps. H.264 predicts both sizes of intra-frame blocks from pixels in previously encoded blocks in the same slice in the spatial domain, which means that the luma information is used. Earlier MPEG-standards conducted intra-frame prediction in the frequency domain.
4x4-blocks are predicted by either a fixed linear combination of surrounding pixels encoded before or by a directional prediction from those pixels to model edges with various angles. 16x16-blocks are predicted from a larger number of surrounding pixels or may be estimated in a plane-mode, where each luma-prediction at (x, y) is generated by a function of the pixels $(x, \widetilde{y})$, $(\widetilde{x}, y)$ and $(\widetilde{x}, \widetilde{y})$, where $\widetilde{x}$ and $\widetilde{y}$ are the coordinates of the encoded pixels right outside the current block.

Inter-frame motion estimation plays a key role in the compression performance of H.264 and is applied to P- and B-slices. It may be performed on sub-blocks of macro-blocks with width and height combinations of 16 and 8 pixels, or of 8 and 4 pixels.

Definition 2.3.5 (Motion vector)**.** A ***motion vector*** is a vector

$$\mathfrak{m} \in \left\{ v \in \mathbb{Q}^2 \,\middle|\, v = \left(\frac{k_W L_W}{4}, \frac{k_H L_H}{4} \right), k_W, k_H \in \mathbb{N} \right\},$$

where $L_W, L_H \in \mathbb{Q}$ are horizontal and vertical distances between neighboring luma samples, which is a pixel in general. In other words, the resolution of motion vectors is a quarter pixel in both dimensions in general, sometimes abbreviated by ***qpel-accuracy***. This motion vector defines the translation applied to a ***source block*** in the same slice of a previously encoded source frame to a predicted ***target block***.

If a motion vector points to a non-integer location in a pixel grid, the values at integer positions are interpolated. Additionally, note that the source block may be located in any previously encoded frame (which may even be a future frame in the sequence order), not only in the previous one as in earlier MPEG-standards. While motion-estimation in P-slices only relies on a single motion vector, H.264 allows the utilization of two motion vectors with source blocks being located in different frames or a linear combination of two such motion vectors for B-slices. H.264 also provides a block estimation mode, which omits the transform-coding of the residual, yielding high efficiency encoding of blocks with little or no changes compared to the source block.
In the final encoding process, only the difference to the median of the motion vectors of previously encoded surrounding blocks is encoded instead of the motion vector itself.

One key question left open by the H.264-standard is how these motion vectors are to be calculated. Extensive research has been conducted on this topic, and we refer to [Pur04] for a summary of currently used algorithms.

2.3.4 Transform coding

In earlier versions of the MPEG-standards, the key ingredient in transform coding was a discrete cosine transform (DCT) on 8x8-pixel-blocks. It is based on the real-valued cosine transform, which closely resembles the Fourier transformation, in which the sin-terms are omitted. It was introduced in [Ahm74], with a proof of the decorrelation properties of the DCT being very close to the optimal Karhunen-Loéve-transform.
In H.264 the DCT was replaced by a 4x4 pixel integer transformation, which is also referred to as ***high correlation transform (HCT)***, an approximation of the DCT. It may be calculated using 16-bit integer arithmetic only, thus reducing the computational complexity of the transform. Additionally, the 4x4 transform matrix to be used is specified by H.264, which eliminates differing accuracies of encoder and decoder, which introduced additional errors in previous standards.

Remark 2.3.6. Note that the reduction of the transform size from 8x8 pixels to 4x4 pixels reflects the increase in prediction accuracy achieved by H.264. Earlier standards relied more on the decorrelation properties of the DCT.

If a 16x16 block is intra coded and transformed, the first four entries of the resulting 4x4 matrices (in the literature referred to as ***DC-coefficients***) are transformed again by a 4x4 Hadamard transform, which is also related to the Fourier transform.

2.3.5 Scan, quantization and entropy coding

The final stage of encoding a macro-block consists of entropy coding the output values of the above transform coding. In order to prepare the values, the ***quantization*** of the resulting coefficients is already incorporated into the transform, but it may vary between blocks. The calculation of the quantization parameters is processed by the encoder and its rate control unit and is then saved in the header of the compressed output bit-stream. Quantization step sizes depend on the corresponding parameters logarithmically, opposed to the linear dependency in earlier standards.

Scanning refers to the order, in which a set of values is processed. In order to encode the transformed values more efficiently, they are scanned following a zig-zag-scheme, which tends to order them by their magnitude due to the properties of the 4x4 integer transform. Therefore correlations between the coefficients are better localized, resulting in more efficient entropy coding.

The final entropy coding stage is divided into two major steps:

In the first one, the value to be encoded is transformed into a binary representation. Since the variety of encoded integer values may be very large, using the ordinary binary representation of it may be inefficient, if the probability distribution of possible values is highly skewed. This is exploited by transforming each integer into a binary representation following a scheme called ***exponential golomb code (exp-golomb code)***. Similar to Huffman codes shorter sequences are assigned to more likely values, yielding a compressed binary representation. These assignments are based on hard-coded tables, which are additionally based on some context, depending on the data previously encoded. This scheme is called ***CAVLC (context adaptive variable length coding)***.

A more efficient coding scheme is called ***CABAC (context adaptive binary arithmetic coding)***. It employs exp-golomb coding or another variable length coding scheme with context modeling based on previously encoded data and an efficient arithmetic coder, based on an integer implementation. CABAC yields results improved by 10-15% compared to CAVLC at the cost of increased compression times.

2.3.6 In-loop deblocking filter

One substantial improvement in H.264 has been achieved by introducing the in-loop deblocking filter. Due to the transform block size of 8x8 pixels in earlier versions of MPEG standards, encoded video sequences showed massive block artifacts at low bit-rates.

H.264 still contains two elements possibly inducing block artifacts: The 4x4 integer transform with its included quantization, and errors in the motion compensation procedure caused by error-propagation of errors in the source blocks of motion vectors.

The in-loop deblocking filter in H.264 applies countermeasures to these artifacts during the encoding process (cf. figure 1). Since the deblocking filter is part of the encoding and decoding process, it is specified in detail by H.264. We refer to [Pur04] for further details regarding this specification.

Note that the described deblocking filter is a substantially different concept compared to an ordinary deblocking filter applied during playback of a sequence. The encoder (and hence also the decoder) use output of the in-loop filter directly, which results in significantly increased visual quality of the compressed video sequence.

2.3.7 Rate control

Another important element of the encoding process is rate control, which refers to the process of adjusting the quantization and other quality-impairing features during the encoding process to ensure reaching a target bit-rate specified in advance. There are several efficient algorithms available, and we refer to [Pur04] for further reference on this topic.

2.3.8 Summary

Finally, we may summarize the design of MPEG4-AVC/H.264 and describe some of its characteristics:

- H.264 relies on intra- and inter-frame prediction of variable sized pixel blocks and encoding of the residual values by an integer transform, which approximates the DCT.
- In contrast to earlier MPEG standards, H.264 is less dependent on the decorrelation performance of the integer transform.
- Prediction quality has increased significantly due to the introduction of variable sized blocks, quarter-pixel accuracy of motion vectors in combination with a more efficient interpolation, and a large set of macro-block encoding modes.
- All inter-frame predictions rely on translation-based predictions, suggesting performance degradation for other types of motion.
- The in-loop deblocking filter reduces the tendency to introduce block artifacts significantly, thus improving the perceived and measured visual quality at low bit-rates.
- The final entropy coding stage is significantly improved compared to the ones of earlier MPEG-standards.
- Numerical results indicate, that H.264 produces the same measured and perceived visual quality of MPEG-2/H.262 at about half the bit-rate.

2.4 Introduction to information theory

In this section we will introduce the fundamentals of information theory, a field that arose from major contributions of Claude E. Shannon in the 1940's and 50's. Its goal is the definition of ***information***, which is contained in e.g. a message bit-stream. In the following section we will answer the following important questions:

1. Given a bit-stream of data, what is the size of the shortest possible representation of this bit-stream?
2. If such a representation exists, how can it be constructed?
3. Are there compression schemes available that achieve optimal compression ratios under certain conditions?
4. How can we improve the compression performance of the AT3D-implementation?

Some of these questions require notions from the connected field of statistical modeling, which we will introduce throughout this section. Another goal of this section is an exact classification of the improvement problem.

2.4.1 Basic definitions

Definition 2.4.1 (Alphabet)**.** Let $\mathcal{A}$ be a nonempty set of ***symbols*** $\{a_i \mid i \in \underline{n-1}\}$. Then $\mathcal{A}$ is called an ***alphabet*** with size n.

Definition 2.4.2 (Sequence)**.** Let $\mathcal{A}$ be a given alphabet. Then $S = (s_i)_{i \in \mathbb{N}_0}$ is called a ***sequence (of symbols)***, if $s_i \in \mathcal{A}$ is satisfied for all i. The length of S is denoted by $|S|$.

Definition 2.4.3 (Random variable)**.** Let Ω be a sample space, $\mathcal{X}$ an observation space, $\mathcal{E}$ and $\mathcal{F}$ be σ-algebras, $(\Omega, \mathcal{E})$ and $(\mathcal{X}, \mathcal{F})$ be measurable spaces, all defined as usual. Let $(\Omega, \mathcal{E}, P)$ be a probability space, also defined as usual. Then a mapping $\mathbb{X} : \Omega \to \mathcal{X}$ is called a ***random variable***, if $\mathbb{X}$ is $\mathcal{E} - \mathcal{F}$-measurable, which means that the preimage of all subsets of $\mathcal{F}$ is contained in $\mathcal{E}$.

Remark 2.4.4. It can be shown that $\mathbb{X}$ together with P induces a probability measure P^X over $(\mathcal{X}, \mathcal{F})$. The distribution of $\mathbb{X}$ under P is denoted by $P^{\mathbb{X}}$. The main purpose of the definition of random variables is the focus on their distribution and range of values.

Definition 2.4.5 (Source)**.** A ***source*** $\mathcal{S}$ is a random process generating sequences of symbols from an alphabet $\mathcal{A}$. A ***random process*** $\mathcal{S}$ is defined as a sequence of random variables indexed by a set $T \subset \mathbb{N}_0$:

$$\mathcal{S} := (\mathcal{S}_t)_{t \in T} \,.$$

In general, T is a set of discrete units of time, chosen as the set of consecutive natural numbers starting at 0: $T = \{0, 1, 2, \ldots\}$.
A source is called ***(strictly) stationary***, iff for all subsets $\{t_0, \ldots, t_k\} =: \widetilde{T} \subset T$ and with a $\tau \in \mathbb{Z}$ all $\widetilde{T}_\tau := \left\{ t_{k+\tau} \in T \,\middle|\, t_k \in \widetilde{T} \right\} \subset T$ with $\left|\widetilde{T}\right| = \left|\widetilde{T}_\tau\right|$ the corresponding joint probability distributions satisfy

$$P(\mathcal{S}_{t_0}, \ldots, \mathcal{S}_{t_k}) = P(\mathcal{S}_{t_0+\tau}, \ldots, \mathcal{S}_{t_k+\tau}).$$

In other words, a source is stationary, iff all joint probability distributions of sets of random variables from the source are independent of the time index t.

Remark 2.4.6. Note that no statement regarding the distribution of the random variables is made, however they are required to assume values in the same alphabet $\mathcal{A}$.

Example 2.4.7. The following examples illustrate the concept of sources.

- A very simple source $\mathcal{S}^1$ with alphabet $\mathcal{A}_1 := \{\text{Heads}, \text{Tails}\}$ is a coin that is flipped. Each flip generates a sequence with length 1.
- The sources of interest in this thesis are encoders, which typically generate sequences with symbols from the alphabet $\mathcal{A}_2 := \{0, 1\}$. We will describe the AT3D encoder in chapter 2.5.8. The size of the generated sequences varies and depends on the input.

2.4.2 Entropy

We will now introduce the important concept of entropy. To the most part, we will follow [Cov91]. In particular we turn our attention to clarifying the conditions in the presented theorems and to clarifying the fields of application of the introduced notions, which are quite confusing in some parts of the literature on this topic.

Definition 2.4.8 (Entropy of a sequence, self-information)**.** Let S be a sequence with symbols from an alphabet $\mathcal{A}$. Then the distribution of the s_i in S induces a probability mass function P_S defined as follows:

$$P_S : \{s_i \in S\} \to [0, 1], \qquad P_S(s_i) = \frac{|\{s_j \in S \mid s_j = s_i\}|}{|S|}.$$

Then the ***entropy of the sequence*** S is defined as

$$H(S) = -\sum_{i=0}^{|S|-1} P_S(s_i) \log_2 P_S(s_i) \frac{\text{bits}}{\text{symbol}}.$$

The ***self-information*** $I(s_i)$ ***of a symbol*** s_i is defined as

$$I(s_i) := -\log_2 P_S(s_i) \text{ bits},$$

and the ***self-information of a sequence*** S is defined as

$$I(S) := -\sum_{i=0}^{|S|-1} \log_2 P_S(s_i) \text{ bits.}$$

Analogously we may define the above notions for random variables $\mathbb{X}$ by simply replacing P_S with $P^{\mathbb{X}}$ and summing over all possible outcomes.

While the entropy of the sequence can be interpreted as the expected information content per symbol, self-information can be viewed as the true information content of a symbol or a sequence.
We may define joint entropy of two random variables and conditional entropy of a random variable under another random variable as follows:

Definition 2.4.9 (Joint entropy, conditional entropy)**.** Let $\mathbb{X}_1$ be a random variable assuming values in $\mathcal{X}_1$ and $\mathbb{X}_2$ a random variable assuming values in $\mathcal{X}_2$ with joint probability distribution $P \equiv P^{\mathbb{X}_1,\mathbb{X}_2}$.
Then the ***joint entropy of*** $\mathbb{X}_1$ ***and*** $\mathbb{X}_2$ is defined as

$$H(\mathbb{X}_1, \mathbb{X}_2) := -\sum_{x_1 \in \mathcal{X}_1} \sum_{x_2 \in \mathcal{X}_2} P(\mathbb{X}_1 = x_1, \mathbb{X}_2 = x_2) \log_2 P(\mathbb{X}_1 = x_1, \mathbb{X}_2 = x_2),$$

and the ***conditional entropy*** of $\mathbb{X}_2$ under $\mathbb{X}_1$ is defined as

$$\begin{aligned} H(\mathbb{X}_2 \mid \mathbb{X}_1) &:= -\sum_{x_1 \in \mathcal{X}_1, x_2 \in \mathcal{X}_2} P(\mathbb{X}_1 = x_1, \mathbb{X}_2 = x_2) \log_2 P(\mathbb{X}_2 = x_2 \mid \mathbb{X}_1 = x_1) \\ &= -\sum_{x_1 \in \mathcal{X}_1} P(\mathbb{X}_1 = x_1) \sum_{x_2 \in \mathcal{X}_2} P(\mathbb{X}_2 = x_2 \mid \mathbb{X}_1 = x_1) \log_2 P(\mathbb{X}_2 = x_2 \mid \mathbb{X}_1 = x_1) \\ &= -\sum_{x_1 \in \mathcal{X}_1} P(\mathbb{X}_1 = x_1) H(\mathbb{X}_2 \mid \mathbb{X}_1 = x_1). \end{aligned}$$

Remark 2.4.10. Note that the only difference in the definitions of entropy of a sequence and of joint entropy is the probability distribution they are based on.

Definition 2.4.11 (Entropy rate of a source)**.** Let $\mathcal{S}$ be a source generating symbols from an alphabet $\mathcal{A}$. The ***first-order entropy of*** $\mathcal{S}$ is defined as

$$H_1(\mathcal{S}) := H(\mathcal{S}_0) := -\sum_{i=0}^{n-1} P(\mathcal{S}_0 = a_i) \log_2 P(\mathcal{S}_0 = a_i),$$

and the ***n-th-order entropy of*** $\mathcal{S}$ is defined as

$$\begin{aligned} H_m(\mathcal{S}) &:= H(\mathcal{S}_0, \dots, \mathcal{S}_{m-1}) \\ &:= -\sum_{i_1=0}^{n-1} \dots \sum_{i_m=0}^{n-1} P(\mathcal{S}_0 = a_{i_1}, \dots, \mathcal{S}_{m-1} = a_{i_m}) \log_2 \left(P(\mathcal{S}_0 = a_{i_1}, \dots, \mathcal{S}_{m-1} = a_{i_m})\right). \end{aligned}$$

The ***entropy rate of*** $\mathcal{S}$ is defined as

$$H(\mathcal{S}) := \lim_{m \to \infty} \frac{1}{m} H_m(\mathcal{S}),$$

if the limit exists.

Definition 2.4.12 (Alternative entropy rate)**.** Let $\mathcal{S}$ be a source. Then its entropy rate may alternatively defined as

$$H'(\mathcal{S}) = \lim_{n\to\infty} H(\mathcal{S}_n \mid \mathcal{S}_{n-1}, \dots, \mathcal{S}_0),$$

if the limit exists.

Theorem 2.4.13. *Let $\mathcal{S}$ be a stationary source. Then its entropy rate is well-defined, i.e. both of the above limits exist and are equal:*

$$H(\mathcal{S}) = H'(\mathcal{S}).$$

Proof:
Cf. [Cov91], theorem 4.2.1. □

Remark 2.4.14. The definition of $H'(\mathcal{S})$ yields an interesting interpretation, if $\mathcal{S}$ is non-stationary: The proof of the above theorem in [Cov91] relies on the monotonicity of $H(\mathcal{S}_n \mid \mathcal{S}_{n-1}, \dots, \mathcal{S}_0)$ in the stationary case, which is violated in the non-stationary case. This means that on average more knowledge of past observations does not yield more information about the probability distribution of the underlying source.

Remark 2.4.15. Entropy may be measured in different units, depending on the base of the used logarithm. If the natural logarithm is used, entropy is measured in ***nats***, if the base is 10, the corresponding unit is called ***bans***.
In the remainder of this thesis we fix the logarithmic base to 2 and thus the corresponding unit to ***bits***, since we will work with output bit-streams of an encoder. Hence

$$\log x := \log_2 x$$

for any positive real number x from now on.
Additionally, we define

$$0 \log_2(0) := 0,$$

which is mathematically justified by $\lim_{p\downarrow 0} p \log p = 0$ and from an information theoretic point of view by the focus on occurring symbols only. Otherwise $H(\mathcal{S}) = a \in \mathbb{R}_+ \cup \{\infty\}$ for sources $\mathcal{S}$, which do not emit all possible symbols or sequences from its alphabet with strictly positive probability. This strong constraint would render entropy useless in many applications.

Note the difference between the entropy of a sequence and the entropy rate of a source: When considering a source $\mathcal{S}$, sequences generated by $\mathcal{S}$ can have lower entropy than the source itself, but in this case the probability of the generation of this sequence is accordingly smaller. The entropy rate of $\mathcal{S}$ is the expected information content per symbol of all sequences generated by $\mathcal{S}$, while the entropy of a sequence S emitted by $\mathcal{S}$ is the expected information content per symbol of this specific sequence.

Theorem 2.4.16 (Properties of entropy)**.** *Let $\mathcal{S}$ be a source and let $\mathbb{X}, \mathbb{Y}$ be random variables with joint probability distribution P. Then*

1. $H(\mathbb{X} \mid \mathbb{X}) = 0$,
2. $H(\mathbb{Y} \mid \mathbb{X}) \leq H(\mathbb{Y})$, *equality is assumed iff $\mathbb{X}$ and $\mathbb{Y}$ are stochastically independent.*
3. $H(\mathbb{X}, \mathbb{Y}) = H(\mathbb{X}) + H(\mathbb{Y} \mid \mathbb{X})$,
4. $H(\mathcal{S}) \geq 0$,
5. $H(\mathcal{S}) \leq H_1(\mathcal{S})$, *equality is assumed, iff all $\mathcal{S}_i$ are stochastically independent.*
6. $H(\mathcal{S}) \leq H_n(\mathcal{S})$ *for all $n \in \mathbb{N}$.*

Proof:
Cf. [Cov91], chapters 2 and 4.

□

When investigating conditional entropy in the context of sources, it is immediately seen that dependencies between the emitted symbols lower the entropy rate of the source. We will make use of this fact later in the modeling process.

Example 2.4.17. Consider the source $\mathcal{S}^1$, a (fair) coin flip. All $\mathcal{S}^1_t$ are obviously independent and identically distributed, with probabilities $P(\mathcal{S}^1_t = 0) = 0.5$ and $P(\mathcal{S}^1_t = 1) = 0.5$. Hence

$$H(\mathcal{S}) = H_1(\mathcal{S}) = 0.5 \log 2 + 0.5 \log 2 = 1 \frac{\text{bit}}{\text{symbol}}.$$

Example 2.4.18. Consider a source $\mathcal{S}^2$, representing an unfair coin flip with probabilities $P(\mathcal{S}^2_t = 0) = 0.1$ and $P(\mathcal{S}^2_t = 1) = 0.9$. Then

$$H(\mathcal{S}) = 0.1 \log 10 + 0.9 \log \frac{10}{9} \approx 0.467 \frac{\text{bits}}{\text{symbol}}.$$

Note that $H(\mathcal{S}^2)$ is significantly smaller than $H(\mathcal{S}^1)$. This reflects the fact, that $\mathcal{S}^2$ is very likely to generate a 1, but in this case a 1 has a self-information of $\log \frac{10}{9} \approx 0.152$ bits - opposed to the rare zeros, which are more significant and have an information content of $\log 10 \approx 3.322$ bits.

Remark 2.4.19. It is shown in [Cov91] that entropy is maximized, if the underlying probability distribution is the uniform distribution. Entropy assumes the minimal value 0, if there is a symbol s_i with $P(s_i) = 1$. This can be generalized to the entropy rate with suitable requirements on the individual random variables $\mathcal{S}_i$.

Remark 2.4.20. While it is possible to calculate the entropy rate of simple sources, e.g. the source entropy rates of $\mathcal{S}^1$ and $\mathcal{S}^2$, it is not computable for complex sources like the encoder we will investigate in later sections of this thesis. This fact introduces the need for a good ***model*** of the source and raises the questions, how to determine such a model and the quality of it. We will return to these questions later.

Another question is immediately raised: If an incorrect probability distribution is assumed for a source, what is the error in the calculation of the entropy rate induced by this incorrect choice?
It is given by the so-called ***Kullback-Leibler divergence***, also called relative entropy:

Definition 2.4.21 (Relative entropy)**.** Let $\mathbb{X}, \mathbb{Y}$ be random variables assuming values in $\mathcal{X}$ with probability mass functions P and Q satisfying

$$P(\mathbb{X} = x) = 0 \quad \Rightarrow \quad Q(\mathbb{Y} = x) = 0.$$

Then the ***relative entropy*** or ***KL-divergence*** of P and Q is defined as

$$D(P \,||\, Q) := \sum_{x \in \mathcal{X}} P(\mathbb{X} = x) \log \frac{P(\mathbb{X} = x)}{Q(\mathbb{Y} = x)}.$$

Remark 2.4.22.

- Relative entropy is not a metric, since it violates the triangle inequality and since it is not symmetric. Yet it can be viewed as a distance between two probability distributions.
- In practical applications there is a problem with the calculation of relative entropy: the true distribution of a source, P, is not known in general. Thus the relative entropy is not computable in these cases.

2.4.3 Linking entropy to data compression

The notions of entropy and self-information are closely related and suggest the possibility of encoding a symbol s_i with a number of bits that corresponds to its self-information $I(s_i)$. In the following subsection we will connect the concepts of entropy and compression.

Definition 2.4.23 (Code)**.** Let $\mathbb{X}$ be a random variable which assumes values in an alphabet $\mathcal{A}$. Let $\mathcal{A}_\Gamma^*$ be the set of finite sequences over a ***codeword alphabet*** $\mathcal{A}_\Gamma$ with $|\mathcal{A}_\Gamma| = \gamma$. Then a ***code*** is a mapping Γ satisfying

$$\Gamma : \mathcal{A} \to \mathcal{A}_\Gamma^*, \quad x \mapsto \Gamma(x).$$

The range of Γ is called the set of ***codewords***. The number of symbols from $\mathcal{A}_\Gamma$ of which a codeword $\Gamma(x)$ is composed, is denoted by $l(x)$.
The ***expected length*** of a code is

$$L_\Gamma = \sum_{a_i \in \mathcal{A}} P(\mathbb{X} = a_i) l(a_i).$$

A code is called ***prefix code***, if no codeword $\Gamma(x)$ is a prefix of a codeword $\Gamma(y)$ with $x \neq y$.

Remark 2.4.24. It is useful to require a code to be a prefix code: it ensures unique decodability and the capability of instantaneous decoding without the need to read symbols in a sequence beyond the ones to be decoded.

A fundamental result, which is known as the ***source coding theorem***, was proved by Shannon (1948). It provides us with the link between entropy and ideal code length. Here we give a slightly modified formulation:

Theorem 2.4.25 (Source Coding Theorem)**.** *Let $\mathbb{X}$ be a random variable. Then the expected code-length L_Γ for any prefix codes Γ satisfies*

$$\log_\gamma(2) H(\mathbb{X}) \leq L_\Gamma.$$

Equality is assumed, if $P(a_i) = \gamma^{-l(a_i)}$ for all $a_i \in \mathcal{A}$.
In other words, the entropy of a random variable is the tight lower bound for the expected code length per symbol of any prefix code.

Proof:
Cf. [Cov91], theorem 5.3.1. ☐

This result can only be applied to single random variables or to independent and identically distributed (i.i.d.) sources. In practical applications, both the identical distribution and the independence are violated in general. Yet, a corresponding result for arbitrary sources is formulated in the following theorem:

Theorem 2.4.26. *Let $\mathcal{S}$ be a source. Then the minimal expected codeword length per symbol, denoted by L_n^*, is defined as*

$$L_n^* := \min_\Gamma L_{\Gamma_n} := \frac{1}{n} \sum_{(s_0,\ldots,s_{n-1}) \in \mathcal{A}^n} P(\mathcal{S}_0 = s_0, \ldots, \mathcal{S}_{n-1} = s_{n-1}) l(s_0, \ldots, s_{n-1}).$$

It satisfies

$$H(\mathcal{S}_0, \ldots, \mathcal{S}_{n-1}) \leq n L_n^* \leq H(\mathcal{S}_0, \ldots, \mathcal{S}_{n-1}) + 1.$$

If $\mathcal{S}$ is stationary, we additionally get

$$\lim_{n\to\infty} L_n^* = H(\mathcal{S}).$$

Proof:
Cf. [Cov91], theorem 5.4.2. ☐

These results yield a lower bound for the expected length of a codeword for a sequence S generated by a source $\mathcal{S}$. Yet in practical applications with complex sources involved, it is not computable.

One of the goals of this thesis is the development of a model or a class of models of probability distributions which describe the encoder output as good as possible. From the above theorems it follows, that models, which are closer (in the sense of KL-divergence) to the true underlying probability distribution of the source (if existent) will result in better compression performance.

Note that the entropy rate is only well-defined if the encoder is a stationary source, which we will investigate in section 2.5.11 for our application. But even in the case of a non-stationary source the above theorem suggests that finding a model probability distribution, whose joint entropy is as close as possible to the joint entropy of the source, will help the overall compression performance. The above result also suggests that the quality of a model can be measured by its compression performance across a variety of typical outputs of the encoder of AT3D.

Remark 2.4.27. Note that these lower bounds are only valid, if the original sequence S has to be identically reconstructed from the generated codeword. If the expected codeword length is below the entropy (rate), information is lost with a probability close to 1 and the original sequence may not be reconstructed, making the compression scheme a lossy one.

We now turn our attention to the second introductory question of this chapter, and investigate how to construct a coding scheme which is capable of compressing sequences to an optimal length. These schemes exist and a first approach is the construction of ***Huffman codes***. These are prefix codes which assign the shortest codewords to symbols s_i with the highest probabilities. A more detailed description of Huffman coding can be found in [Cov91], chapter 5.6 and [Say00], chapter 3.

Remark 2.4.28 (Optimality of Huffman code)**.** In the literature, Huffman codes are often considered to be "optimal", e.g. in [Cov91] and [Say00]. This can be misleading, because only i.i.d. sources with known probability distributions are considered. Additionally, this notion refers to ***asymptotic optimality***, which is achieved if the expected codeword length converges to the entropy rate of the i.i.d. source $\mathcal{S}$ as the sequence length grows. True non-asymptotic optimality is only achieved, if the symbol probabilities satisfy

$$\forall s_i \in \mathcal{A} : \exists c \in \mathbb{N}_0 : P(s_i) = 2^{-c},$$

i.e. if all symbol probabilities are powers of two.

Remark 2.4.29 (Practical drawbacks of Huffman code)**.** In practice, we have to consider the following issues:

- As stated in [Cov91], Huffman codes only converge to the source entropy as the block size of blocks of symbols assigned to codewords grows. Thus the coding of a binary alphabet would only be optimal, if both 0 and 1 are equiprobable, or asymptotically optimal if long symbol blocks are coded. Due to the construction technique used to generate a Huffman code, large block sizes m require exponentially more memory as m grows, which leads to problems in practical applications.

- A Huffman code cannot be changed during the compression of a sequence - therefore locally changing probability distributions cannot be taken into account, resulting in suboptimal compression performance.
- In summary, static Huffman codes will not achieve satisfactory results in our application. As stated in [Cov91], there are adaptive Huffman codes available, but they also suffer from several drawbacks.

When turning to practical aspects of entropy coding, we have to introduce another concept of great importance. It enables us to compute the true code length of a symbol.

Definition 2.4.30 (Bit-wise coding cost)**.** The ***coding cost*** of encoding a bit $x \in \{0,1\}$ with probability estimates $\widehat{P}(1) = p \in [0,1]$ and $\widehat{P}(0) = 1 - p$ is defined as

$$u_p(x) := -\log(\widehat{P}(x)) + \delta, \qquad \delta > 0.$$

The ***bit-wise coding cost*** of a sequence S with n elements s_i with $0 \leq i < n$, whose probabilities are estimated to be $\widehat{P}(1) = p_i \in [0,1]$ and $\widehat{P}(0) = 1 - p_i$ is defined as

$$u(S) := -\sum_{i=0}^{n} \log(\widehat{P}(s_i)) + \delta, \qquad \delta > 0.$$

Remark 2.4.31. For Huffman codes, the above estimates $\widehat{P}(1)$ and $\widehat{P}(0)$ are necessarily given by the probability distribution, from which the code symbols are derived. As we have seen in the above remark, Huffman codes are not suitable for binary encoding, i.e. δ is large.

Note that coding cost is very closely related to self-information. Bit-wise coding cost refers to single symbols and is based on a probability estimate, which is not necessarily related to the probability distribution of the encoded sequence or its source, if it exists. But if the estimate and true probability are equal, then coding cost and self-information are equal as well. Additionally note that coding cost of a sequence is independent of the entropy of the sequence, since the estimates used for coding may be independent of the probability distribution of the source. Yet, we may relate coding cost to the joint entropy of emitted sequences of a source:

Lemma 2.4.32. *Let $\mathcal{S}$ be an arbitrary source emitting sequences to be encoded and let s_i with $0 \leq i < n$ be the realizations of the random variables $\mathcal{S}_i$, forming sequences S_n. We denote the joint probability distribution of $\mathcal{S}_0, \ldots, \mathcal{S}_{n-1}$ by P_n and the joint probability* *distribution induced by the corresponding estimates of $P(s_i)$ by Q_n. Then the expected value of coding cost $E(u(S_n))$ satisfies*

$$H_n(\mathcal{S}) + D(P_n \,||\, Q_n) \quad \leq \quad E(u(S_n)) \quad < \quad H_n(\mathcal{S}) + D(P_n \,||\, Q_n) + 1.$$

Proof:
Follows from [Cov91], theorem 5.4.3 by replacing the probabilities p and q by the joint probabilities of the sequence.

□

2.4.4 Arithmetic coding

In the previous section an example of a coding scheme was introduced, which essentially achieves compression by an efficient replacement of the symbols in a sequence according to their individual probabilities. Arithmetic coding uses a different approach:
A sequence S with $|S| = n$ from an alphabet $\mathcal{A}$ is coded as a real number $x \in [0, 1)$. This is achieved by starting with $I_0 = [0, 1)$ and then iteratively choosing sub-intervals before selecting a number from the final sub-interval. In the k-th step the sub-interval $I_k \subset [0, 1)$ is partitioned into intervals I_{k_i} whose sizes correspond to the probabilities of the symbols from $\mathcal{A}$. Then $I_{k+1} = I_{k_i}$ is chosen to encode the symbol $s_k = a_i \in \mathcal{A}$. Finally in the n-th step, the real number $x \in I_n$ with the shortest representation is chosen, which is the codeword for S.
In this section the concept of arithmetic coding will be introduced in detail and it will be connected to the concepts from the previous section. Finally, we will return to the problem of modeling a suitable probability distribution. Algorithm 1 describes the arithmetic coding of a given sequence S and a known probability distribution.

Remark 2.4.33. Given a source $\mathcal{S}$ with alphabet $\mathcal{A}$, Huffman codes are mappings $\Gamma_H : \mathcal{A}^m \to \mathcal{A}_\Gamma^*$, with $\mathcal{A}_\Gamma^*$ being the set of finite sequences over the codeword alphabet $\mathcal{A}_\Gamma$ and $m \in \mathbb{N}$ fixed. It maps symbols or blocks of symbols to codewords.
An arithmetic code is a mapping $\Gamma_A : \mathcal{A}^* \to \mathcal{A}_\Gamma^*$, where $\mathcal{A}^*$ denotes the set of finite sequences of symbols from $\mathcal{A}$. It maps whole sequences of symbols to codewords.

While the algorithm is straightforward, the last step raises the question, which representation of x is the most space-efficient and how it is determined. For that purpose we introduce the following notion:

Definition 2.4.34 (Binary fraction in the unit interval)**.** A finite ***binary fraction*** $x_b = 0.b_1 b_2 \ldots b_n \in [0, 1)$, with $b_i \in \{0, 1\}$ and $n \in \mathbb{N}$ can be converted into a real number x by applying the following mapping:

$$Dec_n : \{0,1\}^n \to [0,1)\,, \quad (b_1, \ldots, b_n) \mapsto Dec_n(b_1, \ldots, b_n) := \frac{\sum_{i=1}^{n} b_i 2^{n-i}}{2^n}.$$

Algorithm 1 Arithmetic coding: Encoder

```
Input: $\mathcal{A} = \{a_0, \dots a_{n-1}\}, P(a_0), \dots, P(a_{n-1}), S = (s_0, \dots, s_{m-1})$
Let $\mathfrak{a}_0 := 0,\ \mathfrak{b}_0 := 1,\ I_0 := [\mathfrak{a}_0, \mathfrak{b}_0)$
for $k = 1$ to $m$ do
    Determine $j$, s.t. $s_{k-1} = a_j$
    $\mathfrak{a}_k := \mathfrak{a}_{k-1} + (\mathfrak{b}_{k-1} - \mathfrak{a}_{k-1}) \sum_{i=0}^{j-1} P(a_i)$
    $\mathfrak{b}_k := \mathfrak{a}_{k-1} + (\mathfrak{b}_{k-1} - \mathfrak{a}_{k-1}) \sum_{i=0}^{j} P(a_i)$
    $I_k := [\mathfrak{a}_k, \mathfrak{b}_k)$
end for
return $x \in I_m$ with shortest representation
```

The inverse of Dec_n is only defined on $R_{Dec_n} := \left\{ x \in [0,1) \,\middle|\, x = \frac{i}{2^n}, i \in \mathbb{N}_0, i < 2^n \right\}$. Thus finding the number $x \in I_n$ with the shortest possible binary fractional representation is equivalent to finding the smallest n, such that $R_{Dec_n} \cap I_n \neq \emptyset$.

Example 2.4.35. The following examples illustrate the conversion of binary fractions to decimal numbers:

1. $x_b = 0.01_b = \frac{3}{4}$,
2. $x_b = 0.110_b = \frac{6}{8} = \frac{3}{4}$,
3. $x_b = 0.111_b = \frac{7}{8}$.

In practical applications only the fractional part is saved as a bit-stream. Note that the calculation of x depends on the employed implementation. Therefore, we will not go into further detail and refer to [Say00] for more information on this subject.

Example 2.4.36 (Encoding)**.** Let $\mathcal{S}$ be an i.i.d. source with alphabet $\mathcal{A} = \{a, b, c\}$ and the following probability distribution: $P(\mathcal{S}_t = a) = P(\mathcal{S}_t = b) = \frac{1}{5}, P(\mathcal{S}_t = c) = \frac{3}{5}$ for all $t \in \mathbb{Z}$. Let $S_1 = (a, c, c, c, b)$ be a sequence emitted by $\mathcal{S}$.
Note that here the symbol probability distributions of $\mathcal{S}$ and S_1 are identical by coincidence. The encoding procedure is visualized in figure 2 and starts with initializing $I_0 = [0, 1)$.

- For $k = 1$ the partition of I_0 results in

  $$I_{0_0} = [0, 0.2)\,, \qquad I_{0_1} = [0.2, 0.4)\,, \qquad I_{0_2} = [0.4, 1)\,.$$

  Since $s_0 = a$, the sub-interval I_1 is defined as

  $$I_1 := I_{0_0} = [0, 0.2)\,.$$

- For $k = 2$ the partition of I_1 results in

  $$I_{1_0} = [0, 0.04)\,, \qquad I_{1_1} = [0.04, 0.08)\,, \qquad I_{1_2} = [0.08, 0.2)\,.$$

  Since $s_1 = c$, the sub-interval I_2 is defined as

  $$I_2 := I_{1_2} = [0.08, 0.2)\,.$$

- The following steps are performed analogously. The final interval is given by $I_5 = [0.16544, 0.17408)$, and the number $x \in I_5$ with the shortest binary representation is

$$x = 0.001011_b = 0.171875.$$

Hence S_1 is represented in 6 bits, which is shorter than the entropy rate of the source and the (equal) entropy of S_1 suggest.

$$\begin{aligned} |S_1| \cdot H(S_1) &= 5\left(-\frac{1}{5}\log_2\left(\frac{1}{5}\right) - \frac{1}{5}\log_2\left(\frac{1}{5}\right) - \frac{3}{5}\log_2\left(\frac{3}{5}\right)\right) \\ &\approx 5 \cdot 1.37 \text{ bits} \\ &= 6.85 \text{ bits.} \end{aligned}$$

The reason for that is that entropy is a lower bound for the expected code length: When considering the entropy of the sequence S_1, there are 20 sequences which have a length of five and the given probability distribution. A small share of these may be coded in 6 bits and the remainder has to be coded in 7 bits. The expected code length for these sequences is 6.85 bits. Since $\mathcal{S}$ is i.i.d., additionally $H(S_1) = H(\mathcal{S})$ holds, thus the expected code length for all sequences with length five emitted by $\mathcal{S}$ is also 6.85 bits.

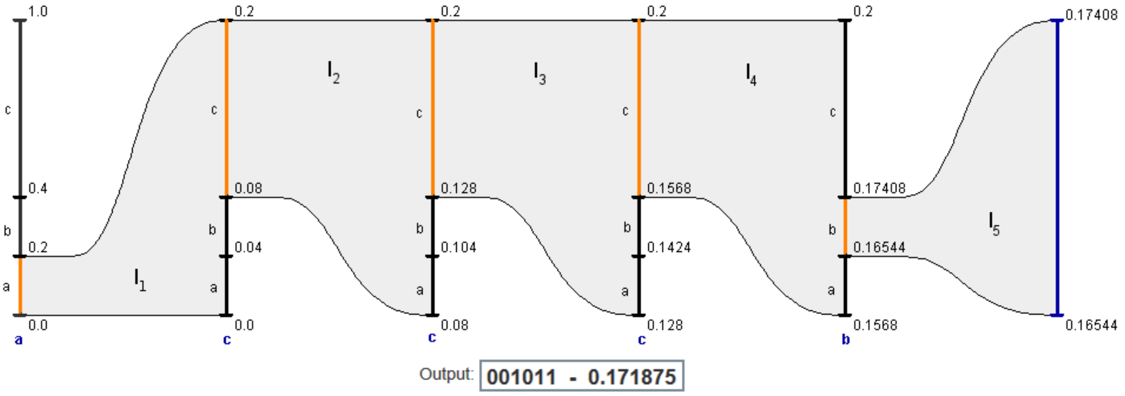

Figure 2: Arithmetic coding example of sequence $S_1 = (a, c, c, c, b)$. Blue letters indicate current symbols in each iteration, orange intervals the sub-intervals selected in the corresponding step.

Example 2.4.37. Let all prerequisites be the same as in the previous example, but consider the extension of S_1 to $S_2 = (S_1, c, b, b)$. Note that in this case the symbol probability distributions of S_2 and $\mathcal{S}$ are not identical. The resulting intervals are

$$I_6 = [0.168896, 0.17498)\,, \quad I_7 = [0.1701128, 0.1761968)\,, \quad I_8 = [0.1713296, 0.1725464)\,,$$

but since $x \in I_8$, the result

$$x = 0.171875$$

remains unchanged. The expected code-length can be calculated as follows:

$$|S_2| \cdot H(\mathcal{S}) \approx 10.97 \text{ bits.}$$

Here the codeword is significantly shorter than suggested by the source entropy.

The previous encoding examples raise a question regarding the decoding procedure: Both examples only differ in the the size of the input sequence, yet the resulting codeword is the same. With a suitable extension, the sequence S_2 may even be extended to any arbitrary length and still result in the same codeword.

- How is the decoder capable of calculating the correct output sequence?
  There are two possible ways to resolve this issue:
  - $|S|$ is sent to the decoder before the encoded bit-stream, which is preferred in general.
  - A special end-of-sequence symbol is encoded in the bit-stream. Note that this approach worsens the compression performance.

Algorithm 2 describes the decoding algorithm for a given real number $x \in [0,1)$, a sequence length $|S|$, and a symbol probability distribution P.

Algorithm 2 Arithmetic coding: Decoder

```
Input: $\mathcal{A} = \{a_0, \dots a_{n-1}\}, P(a_0), \dots, P(a_{n-1}), |S| =: m \in \mathbb{N}, x \in [0,1)$
Let $\mathfrak{a}_0 := 0,\ \mathfrak{b}_0 := 1,\ I_0 := [\mathfrak{a}_0, \mathfrak{b}_0),\ S_0 := \emptyset$
for $k = 1$ to $m$ do
    Partition $I_{k-1}$ into $n$ disjoint intervals $I_{(k-1)_0}, \dots, I_{(k-1)_{n-1}}$, s.t. $I_{(k-1)_i} \cap I_{(k-1)_j} = \emptyset$
      for $i \neq j$ and $\bigcup_{i=0}^{n-1} I_{(k-1)_i} = I_{k-1}$ according to given probabilities
    Determine $i \in \{0, \dots, n-1\}$, s.t. $x \in I_{(k-1)_i}$
    $s_{k-1} := a_i$
    $S_k := (S_{k-1}, s_{k-1})$
    $I_k := I_{(k-1)_i}$
end for
return $S_k$
```

The following example shows the decoding steps corresponding to the previously presented encoding example 2.4.36:

Example 2.4.38. Let $\mathcal{A}$ and P be defined as in example 2.4.36. Let $m = 5$ and $x = 0.171875$.

- In step $k = 1$ the initial unit interval $I_0 := [0,1)$ is split into the following partitions:

$$I_{0_0} = [0, 0.2), \qquad I_{0_1} = [0.2, 0.4), \qquad I_{0_2} = [0.4, 1).$$

  Now the index i satisfying $x \in I_{0_i}$ is identified as $i = 0$, resulting in

$$S_1 := (a),$$
$$I_1 := I_{0_0}.$$

- For $k = 2$ the interval I_1 is partitioned into the following sub-intervals:

$$I_{1_0} = [0, 0.04), \qquad I_{1_1} = [0.04, 0.08), \qquad I_{1_2} = [0.08, 0.2).$$

Now the index i satisfying $x \in I_{1_i}$ is identified as $i = 2$, resulting in

$$\begin{aligned} S_2 &= (a, c), \\ I_2 &= I_{0_2}. \end{aligned}$$

- The following steps are performed analogously, yielding

  $$S_5 = (a, c, c, c, b)$$

  after $k = 5$ steps, where the decoding process is terminated with output sequence S_5.

Again, figure 2 can be used as a visualization, if the blue letters are regarded as output symbols determined by the assignment of x to the interval I_{k_i}.

After this introduction of the encoding and decoding mechanisms of arithmetic coding we will now describe some of its characteristics and compare it to Huffman coding, which was introduced in the last section.

Theorem 2.4.39. *If the intervals I_{k_i} are disjoint for any arbitrary, but fixed k, an arithmetic code is a prefix code.*

Proof:
Cf. [Say00], chapter 4.4. □

Remark 2.4.40 (Optimality of arithmetic coding for i.i.d. sources)**.** Assume $\mathcal{S}$ is an i.i.d. source with entropy rate $H(\mathcal{S})$ and known probability distribution. It can be shown, that the code-length of arithmetic codes asymptotically converges to $H(\mathcal{S})$. If the source alphabet $\mathcal{A}$ is small, arithmetic codes tend to produce shorter codes compared to Huffman codes for fixed $|S|$. This is due to the capability of assigning fractional bit-lengths to symbols.

Unfortunately the above results are not of practical use, because they also only apply to i.i.d. sources. Analogous to the previous section, we will investigate practical properties of arithmetic codes:

Remark 2.4.41 (Practical aspects of arithmetic coding)**.**

- Up to now we have not considered implementational details of arithmetic coding. But algorithms 1 and 2 cannot be implemented on a computer with limited memory for sequences of arbitrary length, because it would have to handle real numbers with infinite precision. In [Say00], chapter 4.4.3, a sample implementation using integer arithmetic only is described. Essentially it uses the fact that if an interval I_k is sufficiently small, all binary fractions $y_b \in I_k$ contain the same leading bits. Thus these bits can be written to disk and are not kept in memory. This scheme leads to another useful property: the sequence can be decoded partially.

- Depending on the implementation a small loss of coding efficiency has to be accepted to ensure unique decodability. Essentially, not the full interval range can be used due to integer rounding. [Mah02] states, that the arithmetic coder of PAQ (which is introduced in chapter 3.1), incurs $\delta < 0.0002$ additional bits per symbol.

- As shown in [Say00], chapter 4.4.1, the length of an arithmetic code is always within two bits of the source entropy, iff its probability distribution is known and used for the coder.

2.4.5 Separation of modeling and encoding

The last remark yields a crucial result in particular: If we try to compress output from an arbitrary source using an arithmetic coder, the most important component is the used statistical model, which feeds probability estimates to the arithmetic coder. It is capable of encoding any sequence with a code-length almost equal to the entropy of the model plus a penalty, that can be quantified theoretically using the KL-divergence.

Remark 2.4.42 (Separation of modeling and coding)**.** Note that opposed to many other prefix codes the statistical modeling is independent of the coding in arithmetic coding. Encoder and decoder have to use the same model, either by hard-coding it into both or by a selection mechanism in the encoder and an incorporation of this choice into the resulting bit-stream. Therefore, the disadvantages of Huffman coding are eliminated: An arithmetic coder can react arbitrarily fast to changed probability distributions and therefore gives us maximal flexibility in the model design while maintaining close-to-optimal compression performance.

Definition 2.4.43 (Adaptive arithmetic coder)**.** An arithmetic coder, which uses a variable probability distribution, is called ***adaptive arithmetic coder***.

Definition 2.4.44 (Context-based adaptive arithmetic coder)**.** An arithmetic coder, which uses conditional probabilities $P(\mathcal{S}_n \mid \mathcal{S}_1, \ldots, \mathcal{S}_{n-1})$ to model the source probability distribution is called ***context-based adaptive arithmetic coder***.

Recall that modeling a complex source may be a very difficult task. The description of a source like the AT3D encoder with a single model is very likely to be impossible. Thus, a technique called ***context mixing*** may yield better results. It generates a probability estimate $\widehat{P}(\mathbb{X} = x)$ for the symbol to be coded next by combining the estimates of multiple models. Under certain conditions this combined estimate is better than each of the individual estimates of $P(\mathbb{X} = x)$, which we will show in section 3.1.

Up to this point we were able to understand, that a lower bound for code lengths exists, but we are not able to compute it in practical applications. We also know, that the problem of achieving the best possible code lengths is solely connected to the quality of the statistical models employed to describe the output of the AT3D encoder, since tools like arithmetic coding are readily available.

Remark 2.4.45 (Measuring model quality by compression performance)**.** With the above results we can draw a very important conclusion based on a theoretical foundation, that is compliant with intuition:
We may measure the quality of an implemented model directly by its resulting compression performance. The converse holds as well, if a model compresses input data well, it also describes the data well. A theoretical approach to model design for various sources is far beyond the scope of this thesis, therefore we refer the reader to e.g. [Fed92], which gives a summary on universal source coding.

A theoretical classification of the characteristics of the AT3D encoder with regard to information theory is a very delicate task due to its complexity. In the following chapter we will describe this algorithm, which produces the data we later want to entropy-code.

2.4.6 Summary

In this chapter we provided an overview of the literature on information theory basics. The most important aspects are:

- Theoretical expected lower bounds for any kind of data compression exist. For stationary sources $\mathcal{S}$ it is the source entropy rate $H(\mathcal{S})$, and for non-stationary sources (as the AT3D encoder, cf. section 2.5.11) it is $H(\mathcal{S}_0, \ldots, \mathcal{S}_{n-1})$ when considering a sequence of length n.
- These bounds are not computable in practical applications in general.
- Compression is improved by finding a model probability distribution, that is close (in the sense of KL-divergence) to the true distribution induced by $\mathcal{S}$.
- If such a model is found, it may be efficiently implemented by using a context-based arithmetic coder.

2.5 AT3D

In this section we will introduce the compression scheme AT3D in detail. We follow [Dem11] to the most part, but extend it with regard to the implementational details of the described algorithms. Therefore, this section contains the most extensive investigation of the current state of AT3D available so far.
The method AT3D is the generalization of the 2d image compression algorithm proposed in [Dem06] to 3d video data compression. "It combines a recursive pixel removal scheme, adaptive thinning, with least squares approximation by tri-variate linear splines." (Cf. [Dem11]) After a pre-processing edge-detection procedure, which reduces the size of the initial pixel grid, adaptive thinning is applied to obtain a sparse set of significant pixels.

> The set of significant pixels yields a (unique) anisotropic Delaunay tetrahedralization, where the vertices of the tetrahedralization are given by the significant pixels. This in turn defines a linear approximation space of trivariate linear spline functions over the so obtained anisotropic tetrahedralization. The approximation to the video data is then given by the best approximating linear spline, in the sense of least squares approximation. The overall aim of the resulting adaptive approximation algorithm is to construct a suitable linear spline space of small dimension (being represented by a sparse set of significant pixels), such that the distance between the best approximating linear spline and the given video data is small, thus yielding a small reconstruction error. The construction of the spline space can be viewed as a highly nonlinear approximation process. (cf. [Dem11])

In each iteration a least significant pixel is determined by measuring the mean squared interpolation error incurred by the removal of a pixel from the tetrahedralization. After obtaining the sparse pixel set, a post-processing step called ***pixel exchange*** further optimizes this pixel set according to local properties of the original video data. Finally, the global best approximation on the exchanged pixel grid is calculated. In order to obtain a compressed bit-stream, first the pixel-positions are entropy-coded and afterwards the approximated grayscale values at the significant pixels are quantized and entropy-coded. The used entropy coding method is an efficient ***context-based adaptive arithmetic coder***. One of the main goals of this thesis is the improvement of the entropy coding compression performance.
AT3D is a lossy compression scheme in general, yet if a sufficiently low number of pixels is removed, it remains lossless. The stages of AT3D are listed in table 1.

Geometry	Pre-Processing	Edge detection
	Initial pixel grid	Interpolation with trivariate linear splines over a Delaunay tetrahedralization
	Thinning of pixel grid	Adaptive thinning 6
	Post-Processing	Pixel exchange Best approximation
Entropy-coding	Preparation of data	Quantization
	Bit-stream generation	(Context-based) adaptive arithmetic coding

Table 1: Overview of AT3D stages.

The very first step is to choose a suitable definition of video data for our intended use. Recall that the chosen approach is mathematically different from the one used in the known MPEG-standards. Here we consider video data to be a three-dimensional scalar field with time being the third spatial dimension.
In the remainder of this thesis the sampling- and quantization process, which is essentially the conversion from an analog to a digital signal, is not of interest. This is reflected in the following simplified version of definition 2.2.8:

Definition 2.5.1 (Video domain)**.** Let

$$X = \underline{W-1} \times \underline{H-1} \times \underline{T-1} \subset \mathbb{R}^3, \quad W, H, T \in \mathbb{Z}_+^*$$

With $[X]$ denoting the convex hull of X, a video is a mapping

$$V : [X] \to \underline{2^\rho - 1}, \quad \text{with } |X| = N := WHT.$$

Usually we consider videos with $\rho = 8$, which results in a total of 256 colors.

Definition 2.5.2 (Samples)**.** Let $X \subset \mathbb{R}^3$ be a given video domain.
Then we call the set

$$V|_X := \{V(x) \mid x \in X\}$$

a set of ***video samples***. Analogous to definition 2.2.8 we will refer to a ***pixel*** as the tuple (x_x, x_y, x_z) for any $x \in [X]$ and to $V(x)$ as the grayscale value at the pixel (position) x.

Remark 2.5.3. Note that in the remainder of this thesis we consider the values at individual pixel positions as samples, opposed to the classical signal processing notion, which would consider frames as samples. This is due to the different domain in our definition of video data.

2.5.1 Delaunay tetrahedralizations

In this section we will introduce Delaunay tetrahedralizations and establish their most important properties and describe their influence on the design of AT3D.
Note that in this thesis we choose a slightly different definition of tetrahedra compared to [Dem11], due to the better suitability for chapter 4.

Definition 2.5.4 (Tetrahedron)**.** Let $T_p := \{p_0, p_1, p_2, p_3\} \subset \mathbb{R}^3$. Then a ***tetrahedron*** T is the convex hull $[T_p]$ of those four points. Those points are called ***vertices*** of T.
A tetrahedron T is called ***regular***, iff no three vertices of T are collinear. Otherwise, T is said to be ***degenerated***.

Definition 2.5.5 (Tetrahedralization)**.** Let $Y \subset \mathbb{R}^3$ with $|Y| = n \in \mathbb{N}$.
A ***tetrahedralization*** $\mathcal{T}$ of $[Y]$ is a nonempty set of tetrahedra satisfying

1. $\bigcup_{T\in\mathcal{T}} T = [Y]$,
2. For all $T_1, T_2 \in \mathcal{T}$ with $T_1 \neq T_2$: $\overset{\circ}{T_1} \cap \overset{\circ}{T_2} = \emptyset$.

A tetrahedralization $\mathcal{T}$ is called ***conformal***, if it does not contain hanging edges, meaning two tetrahedra $T_1, T_2 \in \mathcal{T}$ with $T_1 \neq T_2$ only intersect in at most one common face, edge or vertex. Then Y is called the vertex set of the tetrahedralization $\mathcal{T}$.

Definition 2.5.6 (Cell)**.** Let $\mathcal{T}$ be a tetrahedralization with vertex set Y. Then for any vertex $y \in Y$ the ***cell of*** y is denoted by $\mathcal{C}_y$ and defined as

$$\mathcal{C}_y := \bigcup_{y\in T} T.$$

Thus, the cell of a point y is the set of all those tetrahedra that y is a vertex of.

Definition 2.5.7 (Delaunay tetrahedralization)**.** A conformal tetrahedralization $\mathcal{D}_Y$ with vertex set Y is called a ***Delaunay tetrahedralization***, iff for each tetrahedron $T \in \mathcal{D}_Y$ the interior of its circumsphere only contains the vertices of T.

Theorem 2.5.8. *Let $Y \subset \mathbb{R}^3$ be the vertex set of a Delaunay tetrahedralization $\mathcal{D}_Y$. Then*

- *$\mathcal{D}_Y$ is determined uniquely, iff no five vertices from Y are co-spherical, which means there may not exist a sphere with at least five vertices from Y being located on the surface of it,*
- *the Delaunay tetrahedralization $\mathcal{D}_{Y\setminus\{y\}}$ can be calculated from $\mathcal{D}_Y$ only by retetrahedralizing the cell of y, which is equivalent to*

$$\mathcal{D}_Y \setminus \mathcal{C}_y = \mathcal{D}_{Y\setminus\{y\}} \setminus \mathcal{C}_y.$$

Proof:
The first part is proved in [deB08], chapter 9.
The second part follows immediately from the in-sphere-property of Delaunay tetrahedralizations, as the circumsphere of no tetrahedron outside $\mathcal{C}_y$ contains y, so that the removal of y does not affect any tetrahedron outside the $\mathcal{C}_y$.
□

For further information on Delaunay tetrahedralizations and their properties we refer to [deB08]. Details concerning the efficient construction of Delaunay tetrahedralizations in AT3D can be reviewed in [Kha11].

2.5.2 Linear splines over Delaunay tetrahedralizations

The main goal of AT3D is the space-efficient representation of given video data V with domain X. In this section we will describe how to use the vertex set of a given Delaunay

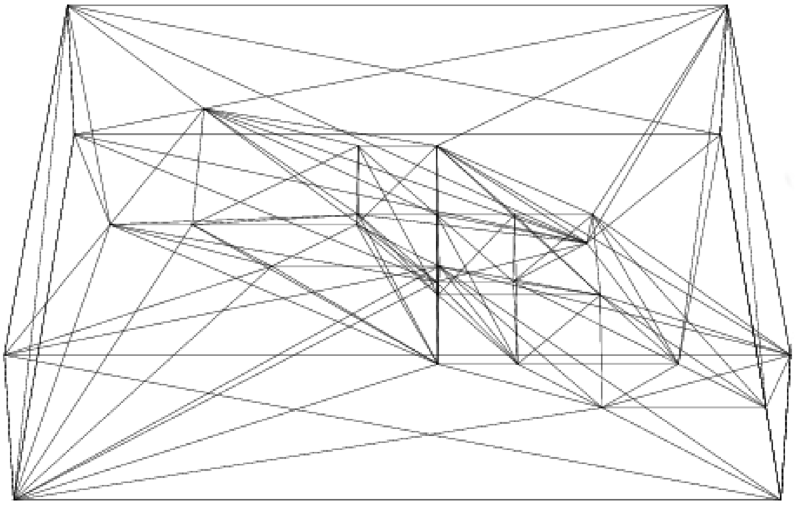

Figure 3: Exemplary Delaunay tetrahedralization of a video domain containing a moving box in front of a solid background.

tetrahedralization to create a suitable approximation space to be able to approximate the given video data $V|_X$ on a subset $Y \subset X$.
In addition to the unique Delaunay tetrahedralization of a given vertex set Y we will be able to determine a unique linear spline interpolant, which can be constructed from Y and a given set of grayscale values at each vertex of Y.

Definition 2.5.9 (Linear spline space)**.** Let Π_1 denote the set of trivariate polynomials with degree at most one. Let $\mathcal{D}_Y$ be a fixed Delaunay tetrahedralization with vertex set Y. When denoting the set of continuous functions on some set A by $\mathcal{C}^0(A)$, the set of linear splines over Y, S_Y, is defined as

$$S_Y := \left\{ f \equiv f(x_1, x_2, x_3) \in \mathcal{C}^0\left([Y]\right) \,\middle|\, \forall\, T \in \mathcal{D}_Y : f|_T \in \Pi_1 \right\}.$$

The set S_Y is the linear space of all trivariate continuous functions that are linear on any tetrahedron $T \in \mathcal{D}_Y$ if restricted to T. An element $s \in S_Y$ is called a ***linear spline over*** $\mathcal{D}_Y$.

Theorem 2.5.10. *Let $V|_Y := \{V(y) \mid y \in Y\}$ be the set of luminance values at the vertices in Y. Then there is a unique spline interpolant $L(Y, V) \in S_Y$ satisfying*

$$L(Y, V)(y) = V(y) \quad \forall y \in Y.$$

The spline interpolant can be represented in the following way:

$$L(Y, V) = \sum_{y \in Y} V(y)\varphi_y,$$

where φ_y are the unique Lagrangian basis functions in S_Y, called **courant elements**. *They satisfy*

$$\varphi_y(x) = \begin{cases} 0 & \text{for all } x \neq y \\ 1 & \text{for } x = y \end{cases} \quad \text{for all } x \in Y. \tag{2.1}$$

Proof:
Cf. [Che05], chapter 1.

□

For a given set Y and the corresponding Delaunay tetrahedralization $\mathcal{D}_Y$ we can now choose S_Y as an approximation space. Even though we are interested in finding the best approximation to $V|_X$ from S_Y, we will restrict ourselves to finding the unique interpolant instead, because the calculation of the best approximation is a global operation after the removal of each pixel from the set Y, therefore it is only calculated in a final post-processing step. The interpolant may be updated locally in the cell of the removed pixel.

Remark 2.5.11. Note that an interpolant from $\mathcal{S}_Y$ is not well defined in $[X] \setminus [Y]$, iff $[X] \neq [Y]$. Therefore we require the set of significant pixels Y to contain the eight corner pixels of X from now on, which ensures $[X] = [Y]$.

2.5.3 Pre-processing

The most integral part of the adaptive thinning algorithm is the removal of least significant pixels. Despite the theoretical complexity bound $\mathcal{O}(N \log N)$, where N denotes the number of pixels to be removed from X, the thinning procedure requires many calculation steps and therefore suffers from a relatively long running time.
Therefore, W. Khachabi introduced a pre-processing step to delete a large number of pixels that would be thinned anyways. Since adaptive thinning tends to adapt to the geometry of the underlying picture or video very well and thus produces large concentrations of significant pixels at edges, it was a natural choice to use an efficient edge detection algorithm for pre-processing.
A ***sobel edge detector*** was chosen for that task. Edges can be interpreted as discontinuities in the luminosity function l of a video. These can be detected by calculating an approximation of the gradient of l. The sobel edge detector approximates the partial derivatives of l in all directions by convolution of the original data with so-called convolution kernels, which in this case is a set of three 3x3-matrices. The sobel convolution kernel also incorporates smoothing to avoid false positive detections.
The application of the sobel edge detector leads to the removal of $50 - 80\%$ of the initially present pixels while introducing an additional error of 0.1dB, as stated in [Kha11].
These results have to be considered with care, because Khachabi used test sequences of a very specific type. Brief experiments yielded, that the number of removed pixels with a threshold setting of 32 may result in the removal of significantly less pixels than stated in

[Kha11]. Further investigations are beyond the scope of this thesis, since we will not use pre-processing algorithms in our numerical experiments.

2.5.4 Adaptive thinning

As noted before, the fundamental goal of AT3D is the approximation of given video data $V|_X$ on a set of pixels from a video domain X, minimizing the (ℓ^2-)error. Since the unique linear spline interpolant can be calculated from given X and $V|_X$, the main task in AT3D is to determine a sparse set $X_n \subset X$, which allows for this approximation by interpolation of the given video data. The employed algorithm is called ***adaptive thinning*** and will be introduced in this section.

Selecting a suitable sparse subset X_n is crucial to the quality of the approximation. Unfortunately finding the optimal subset $\widetilde{X_n}$ is a NP-hard problem. Hence there is no algorithm that is capable of finding $\widetilde{X_n}$ in a feasible amount of time for large n, if P $\neq$ NP holds. Therefore, adaptive thinning constructs a nested sequence of subsets of X,

$$X_n \subset X_{n+1} \subset \ldots \subset X_N = X,$$

using a greedy approach. In each step it determines the least significant pixel in a set X_i and removes it. The algorithm can be outlined as follows:

Algorithm 3 Adaptive thinning

```
$X_N := X$
for $k = 1, \ldots, N - n$ do
    Find least significant pixel $x \in X_{N-k+1}$
    $X_{N-k} := X_{N-k+1} \setminus \{x\}$
end for
```

Instantly, this raises the question of how to determine the least significant pixel. We can answer this with the tools introduced before: interpolation with linear splines over Delaunay tetrahedralizations.

Definition 2.5.12 (Pixel significance measure)**.** Let $Y \subset X$ be a subset of a video domain X and let $V|_X$ be a given set of video samples. A pixel $y^* \in Y$ is a ***least significant pixel in*** Y, if

$$e(y^*) = \min_{y \in Y} e(y),$$

where for $y \in Y$

$$e(y) := \eta(Y \setminus \{y\}, X),$$

and

$$\eta(Y, X) := \sum_{x \in X} |L(Y, V)(x) - V(x)|^2,$$

where $L(Y, V)$ is the unique linear spline-interpolant that satisfies

$$L(Y, V)(y) = V(y) \quad \forall y \in Y.$$

Remark 2.5.13. By choosing this measure AT3D minimizes the mean squared error of the interpolant and maximizes the PSNR, which was defined in a more general context in definition 2.2.14. The PSNR in the context of AT3D is defined as follows:

$$\mathrm{PSNR} = 10 \log_{10} \frac{2^{\rho} 2^{\rho}}{\widetilde{\eta}^2(Y, X)},$$

with

$$\widetilde{\eta}^2(Y, X) := \frac{1}{|X|} \sum_{x \in X} |L(Y, V)(x) - V(x)|^2.$$

Note that the pixel significance measure in the above form is not local. Thus after each pixel removal the significances of all vertices of the tetrahedralization would have to be recalculated and the computational complexity would increase significantly. Luckily a slightly modified significance measure can be introduced, which indeed is local:
Recall the locality property of vertex removal from a Delaunay tetrahedralization from theorem 2.5.8. It can be used to prove the following lemma, which defines an ***equivalent local error measure***:

Lemma 2.5.14. *Let $Y \subset X$ be a subset of significant pixels. A pixel y^* is a least significant pixel according to definition 2.5.12, iff*

$$e_\delta(y^*) = \min_{y \in Y} e_\delta(y),$$

where

$$e_\delta(y) := e(y) - \eta(Y, X) = \eta(Y \backslash \{y\}, X) - \eta(Y, X).$$

Then e_δ is a **local significance measure** *that is only affected by pixels from the cell $\mathcal{C}_y$.*

Proof:
Let $y^* \in Y$ be a least significant pixel. Then obviously $e_\delta(y^*) = \min_{y \in Y} e_\delta(y)$, since $\eta(Y, X)$ does not depend on the choice of y. With the same argument the converse holds as well. In order to prove the locality of e_δ, we recall that $\mathcal{D}_{Y \backslash \mathcal{C}_y} = \mathcal{D}_{(Y \backslash \{y\}) \backslash \mathcal{C}_y}$, meaning that a Delaunay tetrahedralization remains unchanged outside the cell of a removed pixel y. Thus

$$e_\delta(y) = e(y) - \eta(Y, X) = \eta(Y \backslash \{y\}, X) - \eta(Y, X) = \eta(Y \backslash \{y\}, X \cap \mathcal{C}_y) - \eta(Y, X \cap \mathcal{C}_y).$$

Hence only pixels from $\mathcal{C}_y$ affect e_δ and the lemma is proven.

$\square$

In more recent revisions of adaptive thinning and in particular in its implementation in AT3D, it is not only possible to remove the least significant pixel from a set, but also a set of two connected significant pixels. This modification improves the compression performance of AT3D significantly.

Definition 2.5.15. Let $Y \subset X$ and Y^e denote the set of pairs of neighboring vertices in $\mathcal{D}_Y$. A pair $\{y_1^*, y_2^*\} \in Y^e$ of neighboring vertices in $\mathcal{D}_Y$ is called a ***least significant connected pixel pair*** in Y, if

$$\eta(y_1^*, y_2^*) = \min_{\{y_1, y_2\} \in Y^e} \eta(y_1, y_2),$$

where for any pixel pair $\{y_1, y_2\} \in Y^e$

$$\eta(y_1, y_2) := \eta(Y \setminus \{y_1, y_2\}, X).$$

Analogous to lemma 2.5.14 we can define a corresponding local error measure:

Lemma 2.5.16. *Let $Y \subset X$ be a subset of significant pixels and let Y^e be defined as above. A pixel pair of neighboring vertices in $\mathcal{D}_Y$, denoted by $\{y_1^*, y_2^*\}$ is a least significant connected pixel pair according to definition 2.5.15, iff*

$$e_\delta(y_1^*, y_2^*) = \min_{\{y_1, y_2\} \in Y^e} e_\delta(y_1, y_2),$$

where

$$e_\delta(y_1, y_2) := e(y_1, y_2) - \eta(Y, X) = \eta(Y \setminus \{y_1, y_2\}, X) - \eta(Y, X).$$

Then e_δ is a **local significance measure** *that is only affected by pixels from the union of the cells $\mathcal{C}_{y_1} \cup \mathcal{C}_{y_2}$.*

Proof:
The proof is completely analogous to the proof of lemma 2.5.14. We only have to note that the removal of $\{y_1^*, y_2^*\}$ can be interpreted as the consecutive single removals of y_1^* and y_2^*. Thus the affected cells are $\mathcal{C}_{y_1}$ and $\mathcal{C}_{y_2}$ and hence e_δ is only affected by pixels from their union.

□

This leads to the updated adaptive thinning algorithm presented in algorithm listing 4. The implementation of AT3D relies on priority queue implementations via ***heaps***. There is one heap for the individual pixel significances and another for storing the significances of connected pixel pairs. It was shown in [Dem11] that this leads to a theoretical computational complexity of adaptive thinning of order $\mathcal{O}(N \log N)$.

2.5.5 Post-processing: Pixel exchange

As we stated before, the output set X_n of adaptive thinning will most likely not be optimal, because it is a greedy scheme. Thus the approximation quality of the interpolant may be further improved by modifying X_n by appropriate measures, which will be introduced in this chapter.
The first post-processing step performs a local optimization by applying a pixel exchange algorithm, which we will introduce now.

Algorithm 4 Adaptive thinning 6

```
Let $X_N \subset X \subset \mathbb{R}^3$ be a subset of a video domain
for $k = 1, \dots, N - n$ do
    Determine
        $$y' = \arg \min_{y \in X_{N-k+1}} \eta(y) \quad \text{and} \quad y'' = \arg \min_{y \in X_{N-k+1} \setminus \{y'\}} \eta(y)$$
    Determine
        $$\{y_1^*, y_2^*\} = \arg \min_{\{y_1, y_2\} \in X_{N-k+1}} \eta(y_1, y_2)$$
    if $\eta(y') + \eta(y'') > \eta(y_1, y_2)$ then
        $X_{N-k} := X_{N-k+1} \setminus \{y_1^*, y_2^*\}$
    else
        $X_{N-k} := X_{N-k+1} \setminus \{y'\}$
    end if
end for
```

Definition 2.5.17 (Exchangeable pixels)**.** For any $Y \subset X$, let $Z = X \setminus Y$. A pixel pair $(y, z) \in Y \times Z$ satisfying $\eta((Y \cup \{z\}) \setminus \{y\}; X) < \eta(Y; X)$ is called ***exchangeable***. A subset $Y \subset X$ is called ***locally optimal*** in X, iff there is no exchangeable pixel pair $(y, z) \in Y \times Z$.

Therefore each exchange strictly reduces the approximation error. We may now define the algorithm itself, it is listed as algorithm 5.

Algorithm 5 Significant pixel exchange

```
Let $Y \subset X$
while $Y$ not locally optimal in $X$ do
    Locate an exchangeable pixel pair $(y, z) \in Y \times Z$
    $Y := (Y \setminus \{y\}) \cup \{z\}, Z := (Z \setminus \{z\}) \cup \{y\}$
end while
return $Y \subset X$ locally optimal in $X$
```

When using the exchange algorithm in practice, it is necessary to keep $|X|$ small for the algorithm to remain local and keep its computational complexity low. Therefore in the current AT3D implementation for each vertex in the tetrahedralization, X is defined as the set of pixels surrounding it:

Definition 2.5.18. The set of pixels surrounding a significant pixel y, $\widetilde{X}_y$, is defined as

$$\widetilde{X}_y := \{z \in X \mid \max(|y_x - z_x|, |y_y - z_y|, |y_z - z_z|) = \|y - z\|_\infty < r \in \mathbb{Z}\}.$$

The set of ***optimizable pixels*** $\widetilde{Y}_y \subset \widetilde{X}_y$ is defined as

$$\widetilde{Y}_y := \left\{ z \in \widetilde{X}_y \cap X_n \right\}.$$

It contains all significant pixels in a box with radius r around a target pixel y. The complement of $\widetilde{Y}_y$ in $\widetilde{X}_y$ is defined as:

$$\widetilde{Z}_y := \widetilde{X}_y \setminus \widetilde{Y}_y.$$

The currently implemented exchange algorithm is summarized in listing 6.

Algorithm 6 Implementation of significant pixel exchange

```
Let $\mathcal{Q}$ be a queue of all significant pixels from a given set $Y \subset X_n$
while $\mathcal{Q} \neq \emptyset$ do
    $y$ = Head of queue $\mathcal{Q}$ (removed from $\mathcal{Q}$)
    $e_y := e_\delta(y),\ \Delta_{max} := 0,\ b := y,\ f$ = false
    for all $z \in \widetilde{Z}_y$ do
        $e_z := \eta((Y \setminus \{y\}) \cup \{z\}, X) - \eta(Y, X)$
        if $(e_z - e_y) > \Delta_{max}$ then
            $\Delta_{max} := (e_z - e_y),\ b := z,\ f :=$ true
        end if
    end for
    $Y := (Y \setminus \{y\}) \cup \{b\}$
    if $f$ = true then
        Add all neighboring significant pixels of $y$ and $b$ to $\mathcal{Q}$
    end if
end while
return Set $Y$ satisfying $\forall y \in Y : \widetilde{Y}_y$ locally optimal in $\widetilde{X}_y$
```

Note that the current implementation does not perform a local exchange. An exchanged pixel is re-inserted into the queue without saving the position it was switched from. Therefore, the original pixel may get exchanged multiple times and finally by pixels, which are outside the local box around the original pixel. Obviously, this algorithm results in better compression, but at the cost of an increased of runtime.

The radius r may be chosen freely. A smaller radius decreases the computational time needed to perform the exchange algorithm, a larger value increases the achieved improvement in quality. As W. Khachabi showed in [Kha11], the current implementation yields an average PSNR-improvement of 0.85dB when choosing $r = 3$. However we have to note that this evaluation was performed on a test set with sequences of similar characteristics.

2.5.6 Post-processing: Best approximation

After fixing the set Y and its Delaunay tetrahedralization $\mathcal{D}_Y$, it is possible to further improve the approximation: Up to now we only interpolated the luminance values $V|_X$ of a target function on the pixels in Y, but now we are able to calculate the (global) best approximation $L^*(Y, V) \in S_Y$, which satisfies

$$\sum_{x \in X} |L^*(Y, V)(x) - V(x)|^2 = \min_{s \in S_Y} \sum_{x \in X} |s(x) - V(x)|^2.$$

The best approximation $L^*(Y, V)$ exists and is unique, since S_Y is a finite dimensional linear space and since $Y \in X$. The calculation of the best approximation is realized with the preconditioned conjugate gradient method (PCG), which may not be optimal in terms of runtime due to the size of the linear system to solve, but it can be shown that degenerate cases with large condition numbers are very unlikely. Also the runtime of this calculation is negligible compared to the runtime of the adaptive thinning algorithm. W. Khachabi has shown in [Kha11] that the calculation of the best approximation improves the PSNR by 0.48 dB on average.

2.5.7 Quantization

After calculating a thinned set of pixels Y and the corresponding Delaunay tetrahedralization $\mathcal{D}_Y$ and a final approximation $L^*(Y, X) \in S_Y$ the resulting data now has to be saved in an efficient binary data format. This final step will be realized with an arithmetic coder, but prior to that another lossy compression step is performed: quantization.

Remark 2.5.19. Quantization referred to in this section is different from the quantization introduced in definition 2.2.3. We work with digital signals whose range is discrete by definition. The quantization step to be introduced in this section is used to further reduce the number of distinct luminance values, which may be encoded more efficiently. This is due to the fact that naturally less bits are needed to encode smaller values and thus they are encoded more efficiently by lossless entropy coding schemes.

Therefore in AT3D we may apply a quantizer of the following form to the set of luminance values $L_Y^* := \{L^*(Y, V)(y) \mid y \in Y\}$ at the significant pixels $y \in Y$:

Definition 2.5.20 (Uniform quantizer for AT3D)**.** We define the following quantizer Q:

$$\begin{aligned} Q : \underline{2^\rho - 1} &\to \underline{2^\xi - 1}, \qquad \text{with } \xi < \rho, \\ x \mapsto Q(x) &= \left\lfloor \frac{x}{2^{\rho-\xi}} \right\rfloor . \end{aligned}$$

It assigns values from an interval $\left[a2^{\rho-\xi}, (a+1)2^{\rho-\xi} - 1\right]$ with $a \in \left\{0, 1, \dots, 2^\xi - 1\right\}$ to a. A quantizer of this type is called a ***uniform quantizer*** with ***step-size*** $2^{\rho-\xi}$. The maximal grayscale value is called $Q_{max} := 2^{\xi-1}$.
A uniform quantizer is defined as a quantizer, which divides its domain into equally sized intervals and assigns different values to each interval.

The result of all computations so far is the following set E, which is entropy-coded into an output bit-stream by an algorithm described in the following section:

$$\boxed{E := \{(y, Q(L^*(Y, V)(y))) \mid y \in Y\}}$$

2.5.8 Implementation of arithmetic coding

In chapter 2.4.4 we introduced an entropy coding algorithm, which is capable of compressing arbitrary data with little additional redundancy with a size close to the entropy rate of the used model of the data source plus a penalty term induced by the incorrectness of the model. In this section we will describe the current implementation of arithmetic coding in AT3D and the used models.
Note that AT3D is an essentially deterministic compression scheme, which produces a unique output bit-stream for a given input data-stream and a given set of parameters. Yet due to the internal complexity of the encoding algorithm, its output may be interpreted as a random experiment, which may be described by statistical models.
The goal is the efficient entropy coding of a given set

$$E = \{(y, Q\left(L^*(Y,V)(y)\right)) \mid y \in Y\},$$

where 'efficient' only refers to the compression performance, not to compression speed. The computational complexity of entropy coding is negligible compared to adaptive thinning. AT3D uses an integer implementation of arithmetic coding, which is identical to the implementation used in the image compression scheme based on adaptive thinning, which is described in [Dem06]. The entropy coding stage is divided into several steps:

- Encoding of global information in a fixed number of bits,
- encoding of significant pixel positions,
- encoding of quantized grayscale values.

This separation enables us to use geometric properties of the tetrahedralization to encode the grayscale values. This is only possible if the decoder has access to the same information as the encoder, therefore the complete set of significant pixels has to be encoded first, so that the decoder can reconstruct the tetrahedralization from it.

2.5.9 Entropy coding: Pixel positions

Let Y be a given set of significant pixels. Instead of coding the components p_x, p_y, p_z of a pixel $p \in Y$, a grid consisting of all pixels in $X \supseteq Y$ is created, such that all positions of significant pixels are marked with ones and all other positions are marked with zeros:

Definition 2.5.21 (Pixel grid)**.** Let X be a given video domain and $Y \subset X$ be a set of significant pixels. We first define the (bijective) ***index mapping***

$$I : X \to \underline{WHT-1}, \qquad p \mapsto I(p_x, p_y, p_z) := WHp_z + Wp_y + p_x,$$

with inverse mapping

$$I^{-1} : \underline{WHT-1} \to X, \qquad i \mapsto I^{-1}(i) := (\widetilde{p}_x, \widetilde{p}_y, \widetilde{p}_z),$$

with

$$\widetilde{p}_z := \left\lfloor \frac{i}{WH} \right\rfloor, \quad \widetilde{p}_y := \left\lfloor \frac{i \mod \widetilde{p}_z WH}{W} \right\rfloor, \quad \widetilde{p}_x := \left\lfloor \frac{i \mod (\widetilde{p}_z WH + \widetilde{p}_y W)}{H} \right\rfloor.$$

For notational convenience we set the (undefined) $a \bmod 0 := a$ for any $a \in \mathbb{Z}$. Then we define

$$G := (g_i)_{0 \le i \le WHT-1}, \qquad g_i = \begin{cases} 0, & \text{if } p = I^{-1}(i) \in X \backslash Y, \\ 1, & \text{if } p = I^{-1}(i) \in Y, \end{cases}$$

as the ***pixel grid*** of (X, Y).

In this set the number of nonzero entries is very small in general. Due to the properties of adaptive thinning significant pixels tend to cumulate where the video data varies with high frequency, which corresponds to edges within a frame or the inter-frame appearance of new objects. The current model design is aimed at capturing this behavior. Before it can be introduced, we need to define the necessary notions.

Definition 2.5.22 (2d context box)**.** Let X be a given video domain with corresponding pixel grid G and let $p \in X$ be a pixel. Then the sets

$$F_i := \{p = (p_x, p_y, p_z) \in X \mid p_z = i\}, \qquad \text{with } 0 \le i < T,$$

are the pixels from individual frames. A ***2d context box of*** p ***with radius*** r ***in frame*** ***i*** is defined as

$$C_{p,r,i} := \{q \in F_i \mid \|q - p\|_\infty < r\}.$$

The corresponding ***restricted 2d context box*** is defined as

$$\widehat{C}_{p,r,i} := \{q \in F_i \mid \|q - p\|_\infty < r \wedge I(q) < I(p)\}.$$

We define the cardinality of any 2d context box as $N_{C_{p,r,i}} := |C_{p,r,i}|$. The set of all 2d context boxes is denoted by C_{2d}. The number of significant pixels in a 2d context box is defined by the mapping

$$N_{2d} : C_{2d} \to \mathbb{N}_0, \qquad C_{p,r,i} \mapsto N_{2d}(C_{p,r,i}) := \left| \left\{ q \in C_{p,r,i} \,\middle|\, g_{I(q)} = 1 \right\} \right|.$$

A restricted 2d context box is visualized in figure 4.

Note that the restriction $I(q) < I(p)$ is necessary to ensure the symmetry between encoder and decoder. In the AT3D implementation, restricted 2d context boxes in the currently encoded frame are not used, but they will be for the new implementation. Video sequences may be divided into blocks of frames, which are processed separately. In general, the separation is applied between strongly differing frames or after a fixed number of frames. In both cases little motion in the block sequence may be assumed, which leads to the following idea to design the contextual model:
Under the above assumption it is likely that most significant pixels are located in the first and last frame of a sequence of frames, therefore the 2d context box of a pixel p in frame i

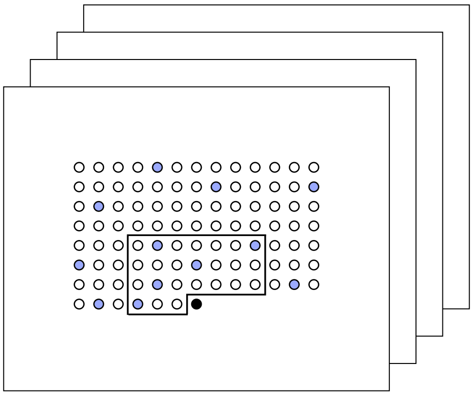

Figure 4: Schematic display of a restricted 2d context box. The black pixel is being encoded, white pixels are thinned and colored pixels are significant pixels.

with radius r is projected into the first and last frame, where the counts of significant pixels are calculated. Therefore we change the formal frame ordering and denote these bound frames by F_0 and F_1. They are encoded first by regular arithmetic coding, so that the unrestricted context boxes become available. The underlying assumption is only justified for certain sequence types, however. For the sake of a simplified notation, we will assume, that the video sequence is a single block. The projection can then be formalized as follows:

Definition 2.5.23 (Projected context box)**.** Let $p \in X \cap F_i$ be a pixel located in frame $i > 1$. Then there are corresponding pixels $p_1 \in F_0$ and $p_2 \in F_1$, which satisfy

$$(p_1)_x = (p_2)_x = p_x \quad \wedge \quad (p_1)_y = (p_2)_y = p_y.$$

Then the ***projected context box*** is defined as

$$C_{p,r} := \{q \in F_0 \cup F_1 \mid \|q - p_1\|_\infty < r \vee \|q - p_2\|_\infty < r\}.$$

Now the ***combined projected pixel count*** N_p is defined as

$$N_p := N_{2d}(C_{p,r}).$$

Now we can define the contexts that will be used in the probability estimation. These are equivalence classes of pixels with projected context boxes containing the same number of significant pixels N_p:

Definition 2.5.24 (Pixel position contexts)**.** Two pixels p and q are said to be in the same context C_i^*, iff $N_p = N_q$. Then $i = N_p$. Thus

$$C_i^* := \{p \in X \mid N_p = i\}.$$

The model now calculates a prediction as follows:

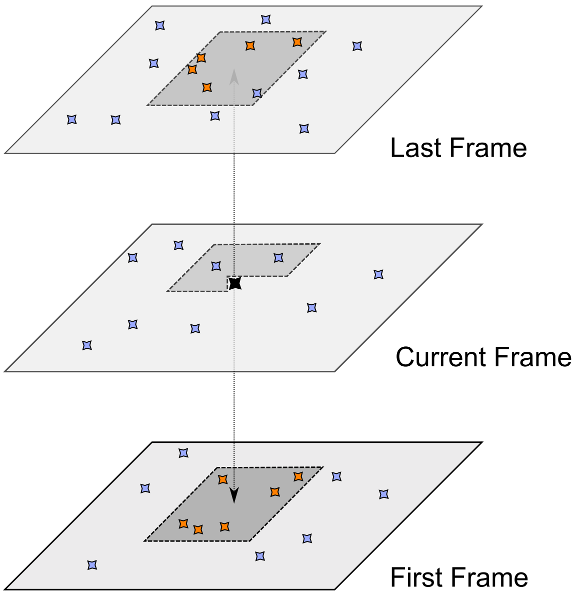

Figure 5: Schematic display of context box projection. Blue and orange pixels are significant pixels, black pixel is the pixel position to be coded, orange pixels are in projected area and thus counted.

Definition 2.5.25 (Probability estimation for pixel positions)**.** Let $\mathcal{G}$ be a source modeling the generation of the sequence G, where $g_i \in \{0,1\}$ is the outcome of the random variable $\mathcal{G}_i$. Now we can define a source $\mathcal{G}^+$, which emits random variables $\mathcal{G}_i^+$, whose outcomes are the numbers of significant pixels in the projected context boxes of $p_i = I^{-1}(i)$, denoted by $N_{I^{-1}(i)} \in \underline{N_{max}}$.

Let p be the next pixel to be coded. In order to define the probability estimates, the number of zeros and ones that were encountered in the same context $C^*_{N_p}$ as p up to the encoding of p are defined as follows:

$$N_{0,p} = \left|\left\{q \in X \;\middle|\; q \in C^*_{N_p} \wedge I(q) < I(p) \wedge g_{I(q)} = 0\right\}\right|,$$

and

$$N_{1,p} = \left|\left\{q \in X \;\middle|\; q \in C^*_{N_p} \wedge I(q) < I(p) \wedge g_{I(q)} = 1\right\}\right|.$$

The following conditional probability estimates are used for the arithmetic coder:

$$\widehat{P}(\mathcal{G}_k = 0 \mid \mathcal{G}_k^+ = N_{I^{-1}(k)}) = \frac{N_{0,I^{-1}(k)}}{N_{0,I^{-1}(k)} + N_{1,I^{-1}(k)}},$$
$$\widehat{P}(\mathcal{G}_k = 1 \mid \mathcal{G}_k^+ = N_{I^{-1}(k)}) = \frac{N_{1,I^{-1}(k)}}{N_{0,I^{-1}(k)} + N_{1,I^{-1}(k)}}.$$

Note that in this approach significant pixels in the restricted 2d context box of p are ignored. Now we can describe the AT3D implementation of the arithmetic coding of the pixel positions:

We can summarize the encoding of pixel positions as follows:

Algorithm 7 Arithmetic coding of pixel positions in AT3D

```
Let $G$ be the pixel grid defined in definition 2.5.21 for given sets $X$ and $Y$
Sort frames, bound frames first, then remaining frames in original order
Write number of significant pixels $n_i$ in frame $i$ to bit-stream for each frame
for $i = 0$ to 1 do
    for each pixel $p \in F_i$ do
        Encode $g_{I(p)}$ with probability estimates $\widehat{P}(1) = \frac{n_i}{WH}, \widehat{P}(0) = 1 - \widehat{P}(1)$.
        if $g_{I(p)} = 1$ then
            $n_i := n_i - 1$
        end if
    end for
end for
Calculate $N_p$ for all pixels $p \in X$
for each pixel $p$ in remaining frames do
    if $(N_{0,p} = 0 \wedge N_{1,p} = 0)$ then
        $N_{0,p} := 1,\ N_{1,p} := 1$
    end if
    Encode $g_{I(p)}$ with probability estimates $\widehat{P}(\mathcal{G}_{I(p)} = 0 \mid \mathcal{G}^+_{I(p)} = N_p)$ and $\widehat{P}(\mathcal{G}_{I(p)} = 1 \mid \mathcal{G}^+_{I(p)} = N_p)$, as defined in definition 2.5.25
    Update $N_{g_{I(p)},p}$
end for
```

- Bound frames of frame blocks are encoded first with an adaptive scheme and based on probabilities of pixels to be significant pixels in each frame.

- The only model information in the output bit-stream is the number of significant pixels in each frame.

- The pixels p in the remaining frames are coded with a context-based adaptive arithmetic coder, using conditional probabilities based on the number of appearances of zeros and ones in the context $C^*_{N_p}$ that were counted until the encoding of p.

- Only the number of significant pixels in the projected context box of a pixel p is used to assign p to a context $C^*_{N_p}$.

- These conditional probabilities are not reset when a frame block is encoded completely.

2.5.10 Entropy coding: Grayscale values

Let $Y \subset X$ be a given set of significant pixels and $\mathcal{D}_Y$ the unique tetrahedralization with vertex set Y. The last encoding step in the AT3D implementation is the entropy coding of the set of grayscale values

$$E_V = \{Q\left(L^*(Y, V)\right)(y) \mid y \in Y\}.$$

Analogous to the pixel positions the set E_V has to be ordered to create an encoding order. This is done by sorting the pixels $y \in Y$ in the order of their addition to $\mathcal{D}_Y$. This sorting procedure induces the sequence

$$\widetilde{E}_V := (v_i)_{0 \le i \le |Y|-1}.$$

The main idea that is used to calculate the probabilities of grayscale values is the assumption, that the length of an edge of a tetrahedron is inversely proportional to the grayscale value-difference along this edge. Before describing the grayscale value encoding in detail, we have to introduce the following notions:

Definition 2.5.26. Let $p \in Y$ be a significant pixel and $\mathcal{D}_Y$ be the unique Delaunay tetrahedralization with vertex set Y. Then the set of ***neighbors*** of p, $\mathcal{N}_p$, is defined as the set of vertices which are incident to p in $\mathcal{D}_Y$.

Definition 2.5.27 (Set of encoded grayscale values)**.** After encoding k values, the ***set of previously encoded grayscale values*** is denoted by $E_V^k \subset E_V, k \in \mathbb{N}_0$ and is defined as follows:

$$E_V^k := \left\{ v_i \in \widetilde{E}_V \;\middle|\; i < k \right\}.$$

The set of significant pixels, which correspond to the encoded grayscale values is called Y_k and is defined as

$$Y_k := \{ y \in Y \mid Q(L^*(Y,V))(y) = v_i, i < k \}.$$

Definition 2.5.28 (Longest causal edge)**.** Let $p \in Y$ with neighbor set $\mathcal{N}_p$ and a given set $Y_k \subset Y$. If $\mathcal{N}_p \cap Y_k \neq \emptyset$, the ***longest causal edge length*** e_{max} of p is defined as

$$e_{max} := \max \left\{ l \in \mathbb{R}^+ \;\middle|\; l = \|p - q\|_2, q \in \mathcal{N}_p \cap Y_k \right\}.$$

The pixel q, which together with p defines the edge of length e_{max} is denoted by q_p. The grayscale difference of p and q_p is denoted by

$$\Delta_{pq_p} := Q(L^*(Y,V))(p) - Q(L^*(Y,V))(q_p).$$

If $\mathcal{N}_p \cap Y_k = \emptyset$, we define $q_p := p$ and

$$\Delta_{pq_p} := Q(L^*(Y,V))(p).$$

Definition 2.5.29. The ***set of grayscale differences*** is denoted by Δ and defined as

$$\Delta := \{ d \in \mathbb{Z} \mid |d| \le Q_{max} \}.$$

The number of times the grayscale difference d was encoded up to the encoding of the grayscale value at pixel p, is denoted by $\widetilde{N}_{d,p}$. For the sake of notational simplicity we omit the formal definition of $\widetilde{N}_{d,p}$ here.

Now we can define the probability estimates for the encoding of grayscale values:

Definition 2.5.30 (Probability estimation for grayscale values)**.** Let $\mathcal{H}$ be a source modeling the generation of grayscale differences, where the non-binary alphabet of $\mathcal{H}_i$ is Δ. Then the following probability estimates are used as a model for the arithmetic coder to code v_k:

$$\widehat{P}(\mathcal{H}_k = d) = \frac{\max\left(1, \widetilde{N}_{d,I^{-1}(k)}\right)}{\sum_{x\in\Delta}\left(1 + \widetilde{N}_{x,I^{-1}(k)}\right)}, \quad \forall d \in \Delta.$$

Note that in order to enable the decoder to uniquely decode a value d, each value $d_i \in \Delta$ has to have a positive probability. This is an instance of the so-called ***zero-frequency problem***, which deteriorates compression performance. Also note that this implementation of arithmetic coding is not context-based, since no conditional probabilities are used.
We can summarize the entropy coding of grayscale values as follows:

Algorithm 8 Arithmetic coding of grayscale values in AT3D

```
Let $Y \subset X$ be a set of sign. pixels with corresponding Delaunay tetrahedralization $\mathcal{D}_Y$
Let $E_V$ be the given corresponding set of grayscale values
Sort $E_V$ and generate the sequence $\widetilde{E}_V$
for $i = 1$ to $|Y|$ do
    Let $p \in Y$ be the pixel corresponding to the grayscale value $v_i$
    Calculate $\Delta_{pq_p}$
    Encode $\Delta_{pq_p}$ using the probability estimate $\widehat{P}(\mathcal{H}_i = \Delta_{pq_p})$ given by definition 2.5.30
    Update $\widetilde{N}_{\Delta_{pq_p},p}$
end for
```

Concludingly we can describe the encoding of grayscale values as follows:

- An adaptive arithmetic coding scheme, which is not context-based, is used.
- For each pixel p the grayscale difference along the longest causal edge is encoded by using the number of occurrences of all grayscale differences to calculate the probability estimates.
- If no such edge exists, the grayscale value of p is encoded.
- All grayscale difference probability estimates are equal in the beginning of the encoding process in order to avoid the zero-frequency problem.
- No model information is transmitted in the bit-stream.
- The counts $\widetilde{N}_{d,p}$ are not reset after each frame block.

Remark 2.5.31. Note that the above description of the grayscale-value compression scheme describes the effective implementation. The code allows the encoding of a value in a context depending on the length of the used causal edge and then applying the above frequency counting mechanism for each encountered length. Yet, it is effectively not used due to the code line `len = (int)(maxLen/CODING_EDGE_LEN_SEG);` in the function `Coding::initContextualFrequencies()`, where `CODING_EDGE_LEN_SEG=2000000`. Deleting this line yields a greatly diminished compression performance, because the number of contexts is too large to profit from significant statistics in certain contexts.

2.5.11 Numerical results regarding stationarity

As we noted before it is important to know whether the AT3D output is stationary or not. One important consequence is the weighting of previously gained statistics for arithmetic coding. If the data is non-stationary, this data cannot be weighted equally. In order to investigate the stationarity of the data, we employ the technique suggested in [Mah02]:
We estimate the probability of matching a random n-gram to an identical n-gram for various distances t. Recall, that stationarity is equivalent to all joint probability distributions of random variables being part of a source being independent of t. We compare two samples from one joint probability distribution to the shifted one. Thus, for a stationary source, we expect the probability to be stable for various values of t.
Our results in tables 2 and 3 clearly show, that neither pixel positions nor grayscale values are emitted by a stationary source. While pixel position matching probabilities show an expected peak for $t = 1$ and the frame width $t = 176$, and its multiple $t = 352$ for both test sequences, the frame-length $t = 25344$ shows a drop for `Football` and an additional peak for `Suzie-90`, which is due to the high amount of motion contained in the former sequence. The probabilities for matching n-grams of grayscale values drop as t grows. We will make use of this information later in the implementation of AT3D_PAQ, which will be introduced in the next chapter.

	`Suzie-90` pixel positions: **p** [%]						
n	t=1	t=10	t=100	t=176	t=352	t=1000	t=25344
1	86.66	85.64	85.28	85.84	86.59	85.15	87.14
3	66.70	65.99	63.94	67.21	66.84	63.84	69.41
5	53.35	52.30	48.84	53.78	53.50	48.84	57.64
7	43.26	42.12	37.58	44.13	43.95	37.85	48.29

	`Suzie-90` grayscale values: **p** [%]					
n	t=1	t=3	t=5	t=10	t=20	t=50
1	2.512	2.342	2.087	2.034	1.708	1.598
2	0.142	0.061	0.082	0.041	0.041	0.072
3	0.006	0.003	0.000	0.005	0.000	0.003

Table 2: Probabilities of matching 1M randomly chosen n-grams at distance t for `Suzie-90` at 90957 significant pixels.

	`Football` pixel positions: **p** [%]						
n	t=1	t=10	t=100	t=176	t=352	t=1000	t=25344
1	81.04	79.56	78.48	79.91	80.64	78.57	74.11
3	54.14	52.45	49.43	54.32	53.65	49.87	42.27
5	37.64	36.47	32.84	38.33	37.27	33.73	25.56
7	27.51	26.14	22.69	27.45	26.84	23.64	15.63

	`Football` grayscale values: **p** [%]					
n	t=1	t=3	t=5	t=10	t=20	t=50
1	2.542	2.445	2.255	2.479	2.206	2.264
2	0.077	0.074	0.056	0.075	0.052	0.060
3	0.000	0.000	0.000	0.004	0.006	0.003

Table 3: Probabilities of matching 1M randomly chosen n-grams at distance t for `Football` at 120319 significant pixels.

3 Development of AT3D_PAQ

In this section we will describe the new lossless coding stage incorporated into AT3D. Research exposed the sub-optimality of the original modeling and prediction stage of entropy coding in AT3D, due to its limitation to a single model. Instead of replacing the single model predicting the probabilities used by the arithmetic coder, the complete lossless coding stage of AT3D was replaced by a readily available scheme called ***PAQ***, which was adapted to the special cases of pixel position- and grayscale value-coding, allowing the implementation of an arbitrary number of models. One of the main results of this thesis is the resulting improved compression performance, which is documented by the numerical results presented in section 3.4.
Originally PAQ was designed as a general-purpose lossless compression scheme for arbitrary data, comparable to the commonly known zip-format. It yields significantly better compression results at the cost of slower compression times and a higher memory consumption. It features a sophisticated ***context mixing*** architecture, a successor of the well-known and successful ***PPM (prediction by partial matching)*** allowing to choose well-predicting models efficiently from an arbitrary set of models during the encoding process, thus ensuring superior adaptivity compared to the original AT3D implementation. PAQ was introduced by Mahoney in 2002 and has been improved continuously since then. In terms of compression performance, the evolution of PAQ climaxed in the development of ***PAQ8***, which is the top-rated compression algorithm in many benchmarks up to today, e.g. in [Ber13] and [Mah13]. It contains models for various kinds of data, including text, images, audio and even executable files. It efficiently combines the individual model predictions with a context mixer based on online convex programming and neural network techniques, and contains so-called adaptive probability maps (APM) to further improve the final prediction. This prediction is then used by an arithmetic encoder to code an input bit. Additionally, it contains features for modeling stationary and non-stationary data efficiently.
We will begin with an overview of the structure of ***PAQ8kx-v7***, a variant of PAQ8 achieving the best compression ratios among all PAQ8-variants. Especially the context mixing algorithm has developed in several stages, and we will briefly describe the motivation and ideas that led to today's implementation. Then recently published theoretic results regarding the compression performance of PAQ will be presented.
An overview of PAQ in as much detail as in this thesis has not been presented before in the literature. After this survey of the architecture of PAQ we will give a summary of our integration of it into the AT3D implementation, including the introduction of our new modeling approach. Finally the results of our numerical experiments will be presented, where we will demonstrate the impact of the new entropy coding stage of AT3D and compare it to the current state-of-the-art video compression scheme, H.264.

3.1 PAQ

In this section we will describe the architecture of PAQ8kx, which is a derivative of PAQ8, developed by Mahoney, Pettis, Ondrus, et al. All versions of PAQ are distributed under the GNU Public License, allowing us to use and modify its code base for our purposes. The most important building block of PAQ is its context mixing algorithm, which will be presented here as well. In order to better understand today's implementation, we will sum up the development of context mixing from the first version of PAQ on.
The basic structure of PAQ8 is simple: It employs numerous models classes $\mathcal{M}_i$, each of which calculates one or more estimates of the probability $P_i(\mathcal{X}_j = 1)$ of the j-th coded bit to be a 1. Then a sophisticated context mixer merges all these predictions into one single prediction $\widehat{P}(\mathcal{X}_j = 1)$ by selecting a set of weights by a context and calculating their linear combination in the logistic domain. This probability is finally filtered by so-called adaptive probability maps (APM), which are also known as secondary symbol estimation, yielding a final probability estimate $\widehat{P}^*(\mathcal{X}_j = 1)$. It is then fed into a range coder, which is essentially equal to an integer implementation of an arithmetic coder. This outline is visualized in figure 6 .

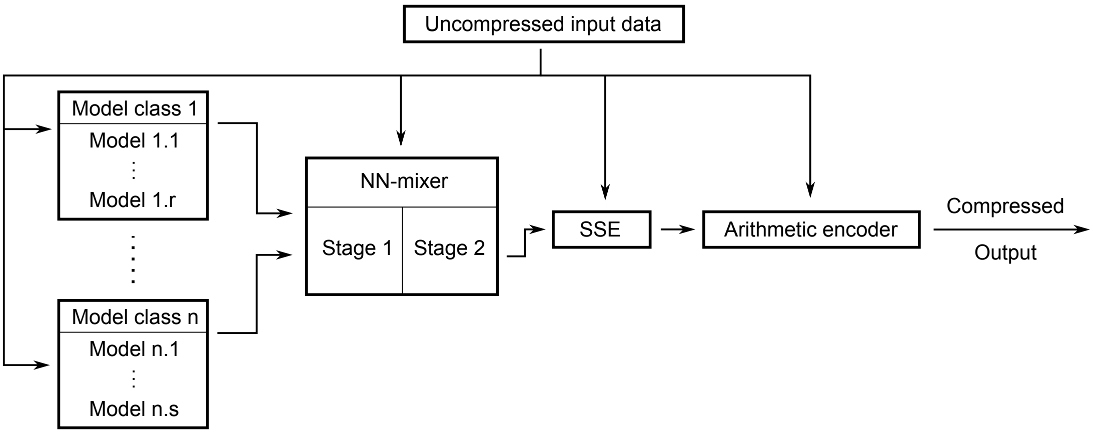

Figure 6: Schematic high-level description of the architecture of PAQ8.

Unfortunately despite the fact that PAQ was introduced more than a decade ago, it is not well understood from a theoretical point of view. Its development was inspired by progress in many connected fields, yielding outstanding compression performance in practice. The program code itself contains bit- and byte-level manipulations, which often make it very difficult to extract the underlying algorithms, if possible at all. Many tuning-parameters were determined ad-hoc by trial-and-error and are now an uncommented part of the implementation of PAQ. We will, however, present all published theoretical investigations of the compression performance of PAQ, mainly [Mat12] and [Mat13] by Mattern. Other publications mainly provide high-level descriptions of PAQ, as [Mah02] and [Mah05] by Mahoney. Also, a different perspective on PAQ is presented in [Kno12], where it is described from a machine learning point of view. The forum in [Enc13] is visited by some of the developers regularly, and there are some discussions on PAQ available.

We will begin with the detailed description of the architecture of PAQ. As in the previous sections, our goal is a mathematically precise formulation of the underlying concepts to extend the sometimes inaccurate description provided in the source code.

3.1.1 PAQ8 - Building blocks

In this section the tools for the calculation of a prediction by a model are described.

Definition 3.1.1 (Input and output bit-stream)**.** The ***input bit-stream*** of PAQ is defined by a sequence $(x)_i$ with $i \in \underline{n-1}$. We denote the source emitting the input bit-stream by $\mathcal{X}$, where each symbol is from the binary alphabet $\mathcal{A} = \{0, 1\}$. The source consists of random variables $\mathcal{X}_t$, with $0 \leq t < n$, whose outcomes are the symbols x_t. The coding procedure results in an output bit-stream $(y)_i$ with $i \in \underline{m-1}$, where m is smaller than n in general. After encoding $k < n$ bits, the $(k+1)$-th bit is predicted, thus we refer to this step as the ***k-th encoding step*** or the ***(k+1)-th prediction step***. We refer to the sequence $(x)_{k-1}$ as the ***encoded*** or ***processed input bit-stream*** and denote it by $(x)^{k-1}$ from now on. Additionally, we define the following notational convention:

$$P(x_k \mid x^{k-1}) = P(\mathcal{X}_k = x_k \mid \mathcal{X}_0 = x_0, \ldots, \mathcal{X}_{k-1} = x_{k-1}).$$

Definition 3.1.2 (Context class)**.** A ***context class*** C is a family of mappings C_i, where i satisfying $0 \leq i < k$ indicates the encoding step number. These mappings are defined as

$$C_i : M_i \to \mathcal{Z},$$

where M_i is an arbitrary finite set, which may depend on the i bits encoded before. A ***context*** is an element from the image of C_i. In the PAQ implementation, we let the ***context range*** $\mathcal{Z} := \underline{2^{32}}$ in general, which we will assume from now on. Based on the image $C_i(M_i)$ we may partition the encoded input bit-stream into equivalence classes.

The above definition describes what is only called "context" in the PAQ documentation, where it is used ambiguously: There are different types of context classes implemented in PAQ, depending on the structure they are fed into:

Definition 3.1.3 (Context class implementations)**.** Let C be a context class. It is called a

- ***mixer context class (mix-context class)***, if it is a context mixer input,
- ***context map context class (cm-context class)***, if it is an input of a context map, which is a complex predefined modeling structure,
- ***general context class (g-context class)***, if it is used to calculate a context-dependent probability, which is then fed into the context mixer.

Remark 3.1.4. A context class C may be used as more than one of the above types at the same time. Yet, due to the implementation the ranges of cm- and g-context classes are significantly larger than the ranges of mix-context classes in general.

A central element in the prediction process of PAQ are so-called ***bit-histories***, which partition the sequence of encoded input bits x^{k-1} into sub-sequences according to the encountered contexts in the encoding process. Note that bit histories are only used for cm- and g-context classes. We formally define them as follows:

Definition 3.1.5 (Context class bit-history)**.** Let C be a context class. Let $\mathcal{B}$ be the set of all sub-sequences of the encoded input bit-stream $(x)^{k-1}$, including the empty sub-sequence $\emptyset$. Then the ***bit histories of*** C, denoted by the mapping B_C^k, are defined as a partition of $(x)^{k-1}$ according to the results of $C_i(m_i)$ for $0 \leq i < k$ and $m_i \in M_i$:

$$B_C^k : \underline{k-1} \times \mathcal{Z} \to \mathcal{B},$$

$$B_C^k(0, z) = \begin{cases} \emptyset, & \text{if } C_0(m_0) \neq z, \\ x_0, & \text{if } C_0(m_0) = z, \end{cases}$$

and for $i > 0$

$$B_C^k(i, z) = \begin{cases} B_C^k(i-1, z), & \text{if } C_i(m_i) \neq z, \\ x_i, & \text{if } C_i(m_i) = z \wedge B_C^k(i-1, z) = \emptyset, \\ (B_C^k(i-1, z), x_i), & \text{if } C_i(m_i) = z \wedge B_C^k(i-1, z) \neq \emptyset. \end{cases}$$

Example 3.1.6. Let $k = 10$, $(x)^{k-1} = (0, 0, 1, 1, 0, 1, 0, 1, 0, 1)$, $M_i = \{0, 1\}$, and $C_i(x_{i-1}) = x_{i-1}$ for $0 < i < k$ and $M_0 = \emptyset$ and $C_0 \equiv \emptyset$. Here the context class C is defined by the previously encoded bit. This yields the bit histories

$$B_C^{10}(9, 0) = (0, 1, 1, 1, 1) \qquad \text{and} \qquad B_C^{10}(9, 1) = (1, 0, 0, 0).$$

Note that since x_{-1} is not defined, the first bit x_0 cannot be assigned to a context, thus the resulting partition only contains the last nine bits of the sequence.

Bit histories are the basis for the calculation of model predictions. Formally a predictor is defined as follows:

Definition 3.1.7 (Predictor)**.** Let C be a context class. A ***predictor for the context-class*** C is a mapping $\mathcal{P}_C^k$, which maps a bit-history to the probability estimate $\widehat{P}(x_k \mid x^{k-1})$ for $0 < k < n$:

$$\mathcal{P}_C^k : \mathcal{B} \to [0, 1], \qquad \mathcal{P}_C^k(B_C^k(k-1, z)) = \widehat{P}(x_k \mid x^{k-1}) \qquad \text{for } z \in \mathcal{Z}.$$

Remark 3.1.8. In a mathematical context, a predictor is comparable to an ***estimator***, which outputs estimates based on previously occurred evidence. Yet we will stick to the denotation defined above to reflect, that often the implemented predictors are not based on mathematical theory, but on experimental results instead.

Remark 3.1.9 (Stationary vs. non-stationary prediction)**.** A central question in the design of a predictor is whether the input data is stationary. In the case of general-purpose entropy coding, this cannot be known in advance, therefore semi-stationary predictors were implemented into PAQ from the first versions on, as pointed out in [Mah02] and [Mah05]. Stationary data requires equal weighting of all elements in a bit-history of a context in the prediction, while non-stationary data requires higher weight on newer elements. Today, most available predictors in PAQ assume stationary data for small bit histories and with increasing bit history-lengths assume more non-stationarity, which yielded the best results in practice. Despite our knowledge of the non-stationarity of data we consider, these highly optimized predictors were not changed, but may be investigated in the future.

Definition 3.1.10 (Model class)**.** A ***model-class*** M is a tuple of sets M_i, each consisting of a (cm- or g-)context class C^i and corresponding predictors $\mathcal{P}^k_{C^i}$. A ***model prediction*** is the output probability estimate $\widehat{P}_{M_i}(x_k \mid x^{k-1})$, calculated from the bit history of the current context:

$$\widehat{P}_{M_i}(x_k \mid x^{k-1}) := \mathcal{P}^k_{C^i}(B^k_{C^i}(k-1, C^i_k(m_k))).$$

Remark 3.1.11 (Bit-history approximation)**.** The accuracy of a predictor depends on its input bit history. In general, a longer bit history yields better prediction results. Yet, with a growing number of context classes the memory-requirements to save bit histories grows. Therefore they may have to be approximated, which is implemented in PAQ efficiently.

Remark 3.1.12. In the documentation of PAQ, the notion "model" refers to a class of models defined above. Figure 7 visualizes the prediction process in a model class.

All of the above elements are part of the encoding process in PAQ and may be adjusted to the source $\mathcal{X}$ generating the input bit-stream. Note that compared to other entropy coding schemes, PAQ allows many degrees of freedom and scalability due to its effective handling of large numbers of contexts (up to 2^{32} for each context class), context classes (up to several hundreds), models (up to several thousands), and even model classes (up to several dozens). These numbers are bounded by memory- and compression time-requirements of the application. We will now answer the question of how to efficiently combine several model predictions into a single prediction.

Definition 3.1.13 (Context mixing)**.** Let $\widehat{P}_0(x_k \mid x^{k-1}), \ldots, \widehat{P}_{s-1}(x_k \mid x^{k-1})$ be estimates of the unknown probability $P(x_k \mid x^{k-1})$. Then a ***context mixer of*** s ***predictions*** is a mapping

$$g : [0,1]^s \to [0,1], \qquad g\left(\widehat{P}_0(x_k \mid x^{k-1}), \ldots, \widehat{P}_{s-1}(x_k \mid x^{k-1})\right) \mapsto \widehat{P}(x_k \mid x^{k-1}).$$

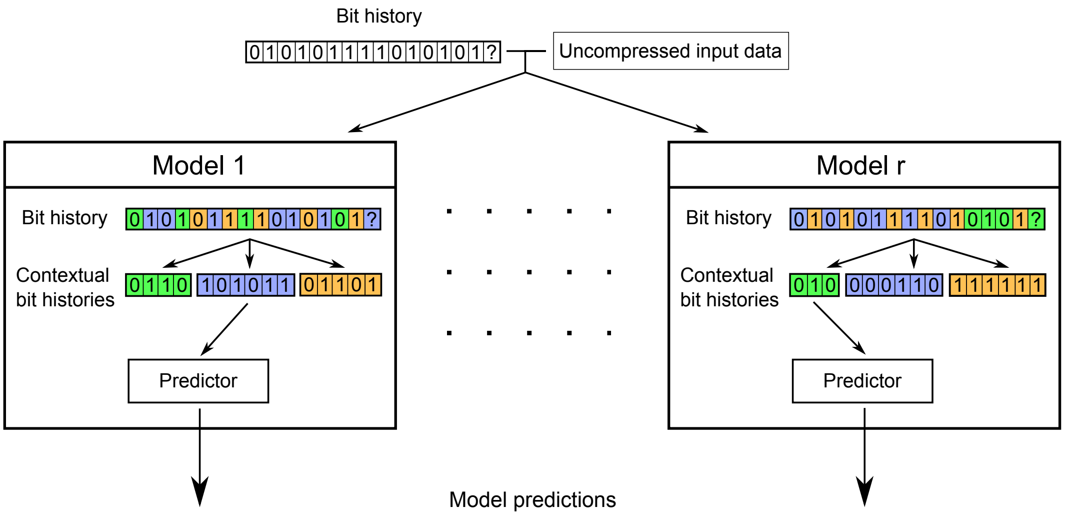

Figure 7: Schematic high-level description of r models in a model class.

Remark 3.1.14. Note that in practical applications a final prediction probability of 0 or 1 is undesirable, because in the case of a misprediction the arithmetic coder is not capable of assigning a nonempty interval to the encoded bit. Thus, in the implementation of PAQ probabilities of 0 and 1 are avoided.

Remark 3.1.15 (Internal representations of probabilities)**.** Internally, probabilities and estimates can only be saved with a certain accuracy. For probabilities in the logistical domain the values are in the range from 0 to 4095. All other probabilities are represented with a range from -2047 to 2047. We will not consider the resulting implementational details in this thesis.

3.1.2 Context mixing in PAQ1-3

In order to understand the context mixing algorithm of PAQ8, we will investigate the development of PAQ context mixing and its motivation in the following sections. Many of the techniques incorporated into PAQ8 can be explained from a mathematical and from a machine learning point of view. We will restrict ourselves to the mathematical description of the algorithms and name references, which yield insight from a machine learning perspective, where applicable.

We will start with the most natural approach to combining various probability predictions into one output prediction: a linear context mixer with fixed weights.

Definition 3.1.16 ((Static) linear context mixer)**.** Let g be a context mixer of s predictions. It is called a ***linear context mixer***, if

$$g_{Lin}\left(\widehat{P}_0(x_k \mid x^{k-1}), \ldots, \widehat{P}_{s-1}(x_k \mid x^{k-1})\right) = \sum_{i=0}^{s-1} w_{i,k} \widehat{P}_i(x_k \mid x^{k-1}),$$

where the $w_{i,k} \in \mathbb{R}^+$ are weights, which satisfy

$$\sum_{i=0}^{s-1} w_{i,k} = 1.$$

If the $w_{i,k}$ are independent of k, the mixer is called ***static***, otherwise it is called ***adaptive***.

The first three versions of PAQ use a combination of a static and an adaptive linear mixer. This is not clearly stated in [Mah02], but from its source code we learn that PAQ1 mixes its ***n-gram models*** with fixed weights of n^2. Here n-grams refer to tuples of previously encoded bytes with length n. The five different model classes are mixed using adaptive weights, which depend on the bit-histories in the corresponding contexts employed by the individual models. These bit-histories serve as a relative confidence of the model in its prediction. More weight is given to models relying on a larger bit-history. It can be shown, that the employed algorithm implements an adaptive linear mixer.

Remark 3.1.17 (Compression gains by context mixing)**.** A natural question immediately arising is how the mixed prediction compares to the individual input predictions. As [Mat12] states, a linear mixture of predictions with weights chosen as prior probabilities of choosing the corresponding model generates a code-length shorter or equal than encoding the model index and the encoded data based on predictions of the best a-posteriori model-choice. In other words a linear mixture prediction is at least as good as the best individual prediction in this setting.

3.1.3 Context mixing in PAQ4-6: Online convex programming and coding cost minimization

The earlier versions of PAQ relied on probability estimates, which were essentially weighted by the confidence of a model in its estimation. This approach yielded good results in practice, but did not target the goal of a compression scheme: minimizing coding cost. In order to take this into account, a new weight update approach was introduced with PAQ4: The updates were computed by an implementation of an ***online gradient descent algorithm*** in order to minimize coding cost, which is an algorithm from ***online convex programming***. It differs from classical offline convex programming in a key aspect: the convex cost function, which is to be minimized, may change in each iteration.

Definition 3.1.18 (Coding cost error)**.** Let x_{k-1} be the last encoded bit with a predicted probability estimate of

$$\widehat{P}(x_{k-1} \mid x^{k-2}) = \sum_{i=0}^{s-1} w_{i,k-1} \widehat{P}_i(x_{k-1} \mid x^{k-2}).$$

Let the (unknown) coding cost of x_{k-1} be $-\log P(x_{k-1} \mid x^{k-2})$. Then the ***coding cost error for bit*** x_{k-1} as a function of the weights is

$$e_k : \mathbb{R}^s \to \mathbb{R}, \qquad e_k(w_{0,k-1}, \ldots, w_{s-1,k-1}) = -\log \widehat{P}(x_{k-1} \mid x^{k-2}) + \log P(x_{k-1} \mid x^{k-2}).$$

Obviously we are interested in minimizing the coding cost error by updating the weights in a beneficial way. But in the above situation we can not apply classical optimization algorithms, e.g. gradient descent: The objective function to be minimized changes in each iteration, since both - the true unknown coding cost of the encoded bit, and the individual probability estimates - change. Thus we are confronted with a different problem class, whose solution is found in the field of machine learning and game theory: online gradient descent. This algorithm was introduced in [Zin03] by Zinkevich and can be applied to a sequence of (sub-)differentiable convex functions, which share a convex domain, which has to be a subset of $\mathbb{R}^n$. Then in each step a single classical gradient descent update is performed, which is called ***greedy projection***. In the following we will present the original problem formulation from [Zin03] and the proposed optimization algorithm.

Definition 3.1.19 (Online convex programming)**.** A ***convex programming problem*** consists of a nonempty convex set $F \subset \mathbb{R}^n$ and a sequence of convex cost functions $c^t : F \to \mathbb{R}$ with $t \in \mathbb{N}$. In each step t, an ***online convex programming algorithm*** calculates a vector $x^t \in F$ and then receives the cost function c^t or its result for input x^t. Additionally, the following assumptions are made:

- The feasible set F is nonempty, bounded and closed.
- Each cost function c^t is differentiable. Zinkevich states, that a modified version of the algorithm also works, if c^t is sub-differentiable.
- The gradient of each c^t is bounded by some $N \in \mathbb{R}^+$:
  $$\left\|\nabla c^t(x)\right\| \leq N \quad \forall x \in F.$$
- For all t there is an algorithm, which calculates $\nabla c^t(x)$ for a given $x \in F$.
- For all $y \in \mathbb{R}^n$ the projection
  $$\operatorname{Proj}(y) = \arg\min_{x \in F} d(x, y)$$
  is well-defined, where $d(x, y)$ denotes the distance between x and y.

Under these assumptions the optimization algorithm calculates the following updates:

Definition 3.1.20. Consider a given online convex programming problem. Initially choose $x^0 \in F$ and a sequence $(\eta)_t$ arbitrarily, where each $\eta_i \in \mathbb{R}^+$. After the reception of c^t, the update is calculated by

$$x^{t+1} = \operatorname{Proj}\left(x^t - \eta_t \nabla c^t(x^t)\right).$$

This immediately raises two questions:

1. Which quantity is optimized by the above algorithm?
2. How is the sequence of step sizes $(\eta)_t$ to be chosen?

Obviously, the optimization goal is different from offline gradient descent. Since the future cost functions c^u with $u > t$ are unknown, it is impossible to determine their minimum in advance. Instead, Zinkevich shows that the greedy projection minimizes ***regret***. It quantifies the sum of the cost differences between the greedy projection output until time T and an algorithm, which knows all cost functions until T in advance, but may only choose one static minimizer $x \in F$. This is formalized as follows:

Definition 3.1.21 (Regret)**.** Let *OCP* be a given online convex programming algorithm to a problem $(F, (c^i)_{i\in\mathbb{N}_0})$. Let $(x^i)_{i\in\mathbb{N}_0}$ be the sequence of output vectors of *OCP*, then the ***cost*** of *OCP* until time T is defined as

$$C_{OCP}(T) = \sum_{t=0}^{T-1} c^t(x^t).$$

The cost of a static feasible solution $x \in F$ is

$$C_x(T) = \sum_{t=0}^{T-1} c^t(x).$$

Then the ***regret of OCP until time*** T is defined as

$$R_{OCP}(T) = C_{OCP}(T) - \min_{x\in F} C_x(T).$$

Therefore if $R_{OCP}(T) = 0$, the online algorithm *OCP* performs as well as the offline convex optimization problem of minimizing $C_x(T)$ with a-priori knowledge of $(c^t)_{0\leq t<T}$. Zinkevich showed, that the average regret per time-step approaches 0 as T grows towards infinity, if $\eta_t = (t+1)^{-\frac{1}{2}}$ for all $0 \leq t < T$:

Theorem 3.1.22. *Let $(F, (c^i)_{i\in\mathbb{N}_0})$ be a given online convex programming problem. Then the regret of the greedy projection is*

$$R_{Proj}(T) \leq \frac{\|F\|^2 \sqrt{T}}{2} + \left(\sqrt{T} - \frac{1}{2}\right) \|\nabla c\|^2,$$

where $\|F\|$ denotes the (bounded) diameter of F and $\|\nabla c\| = \max_{x\in F, t\in\mathbb{N}} \|c^t(x)\|$.
Hence

$$\limsup_{T\to\infty} \frac{R_{Proj}(T)}{T} \leq 0.$$

Additionally it is shown, that if a small amount of change is allowed in the formerly static offline optimization, the average regret approaches a constant bound as $T \to \infty$, which depends on the (arbitrary, but fixed) choice of a step-size η and on $\|\nabla c\|$.

We can now return to the initial problem of finding a beneficial weight update algorithm for PAQ versions 4-6, which minimizes the coding cost error. The update formula given in [Mah05] can be shown to be an instance of online gradient descent. We will omit the formula here due to its notational overhead. Unfortunately, there was no theoretical foundation for the performance of the update formula: It was not shown whether the coding cost error is a convex function of the weights and the step size was chosen independently of the encoding step number k, so the results from [Zin03] cannot be applied.

Remark 3.1.23. Note that for compression with PAQ in practice, the result in theorem 3.1.22 is not relevant. A static minimizer of all coding cost functions in the encoding process is only desirable, if the compressed data is stationary. This is not true in general, thus comparing the update formula to such a static minimizer would not yield results of practical relevance.

In summary, the weight update formula of PAQ4-6 yielded satisfactory results in practice, but there is no theoretical foundation available, which explains these results. Yet, due to the close connections to machine learning, there may be a link to algorithms from this field, which may explain the performance of the context mixing algorithms described above. We will not pursue this investigation, since the algorithm was further modified and we have theoretical evidence for the performance results of this modification.

Remark 3.1.24 (Weight selection)**.** Besides context mixing, PAQ supports a second important mechanism to improve compression: Tuples of model weights $w_{i,k}$ for all i are partitioned into sets according to mix-context classes. Therefore an update of a weight $w_{i,k}$ is performed based on a weight $w_{i,j}$ with $j \leq (k-1)$. In PAQ versions 4 through 6, the number of mix-context classes is considerably smaller than 100.

3.1.4 Context mixing in PAQ7-8: Logistic/geometric mixing

We will now describe the final step in the evolution of the context mixing algorithm in PAQ, the neural network mixer introduced by PAQ7. This mixer is also used in PAQ8 and thus in the implementation of AT3D_PAQ. We will refer to this context mixing algorithm as ***NN-mixer*** from now on. A schematic display of it is given in figure 8.
Artificial neural networks refer to data structures, which resemble the function of biological neurons in the brains of animals. Essentially, an artificial neural network combines input data in a large number of processing units (neurons) according to weights into one or more outputs. These processing units are organized into layers, and only processing units of different layers are connected. A large theory has developed around neural networks, which is far beyond the scope of this thesis. A good introduction can be found in [Roj96]. For the reader familiar with artificial neural networks [Kno12] provides an analysis of the architecture of PAQ8 from a machine learning perspective.
From a mathematical point of view, the changes applied to the structure of previous mixers are to two major ones: The domain of context mixing was changed to the ***logistic*** domain by the application of a transform function before mixing, and the number of mix-context classes was raised by a large factor.

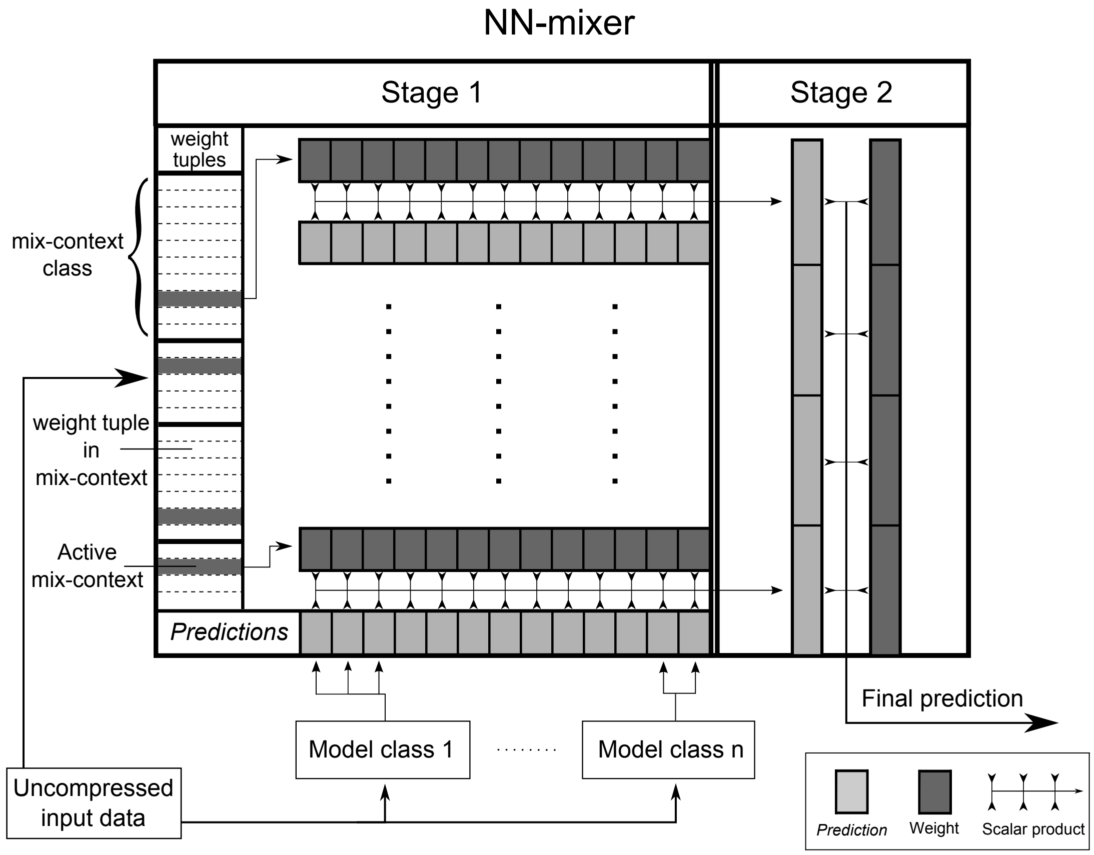

Figure 8: Schematic display of the NN-mixer. Predictions in the logistic domain are marked in italics.

Before we can describe the NN-mixer, we have to introduce the transform functions, which we will name following the PAQ-documentation:

Definition 3.1.25 (Stretch and squash-function)**.** The ***squash***-function is defined as

$$\mathrm{sq} : \mathbb{R} \to (0,1)\,, \qquad \mathrm{sq}(x) = \frac{1}{1+e^{-x}}.$$

The ***stretch***-function, which is the inverse of squash, is defined as

$$\mathrm{st} : (0,1) \to \mathbb{R}, \qquad \mathrm{st}(x) = \ln\frac{x}{1-x}.$$

Squash is also known as the ***logistic function***, stretch as the ***logit***-function in the literature. All arithmetic operations performed using output of the stretch-function, are said to take place in the ***logistic domain***.

Definition 3.1.26 (Logistic mixer)**.** Let g be a context mixer of s predictions. It is called a ***logistic*** context mixer, if

$$g_{Log}\left(\widehat{P}_0(x_k \mid x^{k-1}), \ldots, \widehat{P}_{s-1}(x_k \mid x^{k-1})\right) = \mathrm{sq}\left(\sum_{i=0}^{s-1} w_{i,k}\ \mathrm{st}\left(\widehat{P}_i(x_k \mid x^{k-1})\right)\right),$$

with weights $w_{i,k} \in \mathbb{R}$. Thus a logistic mixer calculates a linear combination of transformed probabilities in the ***logistic*** domain.

The weight update is computed according to the following formula:

$$w_{i,k+1} := w_{i,k} + \alpha \left(x_k - g_{Log} \left(\widehat{P}_0(1 \mid x^{k-1}), \ldots, \widehat{P}_{s-1}(1 \mid x^{k-1}) \right) \right) \text{ st} \left(\widehat{P}_i(1 \mid x^{k-1}) \right), \quad (3.1)$$

where α is a small step-size or ***learning rate***, which is chosen manually. Again, this update formula is an instance of online gradient descent. In [Mat12] Mattern shows, that indeed logistic mixing is optimal in a sense. We will clarify this now, following Mattern's paper, adapted to our notation.

Consider a symbol x from an alphabet $\mathcal{A}$ being the realization of a random variable $\mathbb{X}$, which is distributed according to the true probabilities $P(\mathbb{X} = x')$ for all $x' \in \mathcal{A}$, and given probability estimates $\widehat{P}(\mathbb{X} = x')$ for all $x' \in \mathcal{A}$ yielding a probability distribution $P^{\mathbb{Y}}$ for a random variable $\mathbb{Y}$. Then the expected coding cost $E(u(x))$ is (with the assumption of $\delta = 0$ due to the use of an arithmetic coder) given by

$$E(u(x)) = H(\mathbb{X}) + D(P^{\mathbb{X}} \mid\mid P^{\mathbb{Y}}).$$

Thus the expected coding cost is minimized, if the KL-divergence of $P^{\mathbb{X}}$ and $P^{\mathbb{Y}}$ is minimized. Consider s given models and random variables $\mathbb{Y}_i$ distributed according to the probability distributions induced by the estimates of the models. Then with

$$\mathcal{U} := \left\{ Q : \mathcal{A} \to (0,1) \,\middle|\, \sum_{x \in \mathcal{A}} Q(x) = 1 \right\},$$

$$\mathcal{V} := \left\{ Q : \mathcal{A} \to [0,1] \,\middle|\, \sum_{x \in \mathcal{A}} Q(x) = 1 \right\}$$

we are looking for a random variable $\mathbb{Y}$ with probability distribution $P^{\mathbb{Y}} \in \mathcal{U} \cap \mathcal{V}$, such that

$$P^{\mathbb{Y}} = \arg\min_{Q \in \mathcal{U}} \sum_{i=0}^{s-1} w_i D(Q \mid\mid P^{\mathbb{Y}_i}).$$

Mattern shows, that the minimizer $P^{\mathbb{Y}}$ is a ***geometric mixture***. It is also shown, that the logistic mixer with the weight update formula 3.1 is an instance of a geometric mixture with a weight update implementing online gradient descent. In [Mat13], Mattern additionally provided the first theoretical guarantees for code lengths of data compression with PAQ7 or later.

Remark 3.1.27 (Weight selection)**.** The second major change in the NN-mixer is the possibility of selecting a weight set from a very large number of available sets in each encoding step. In comparison to earlier versions, where essentially each mix-context class constitutes one equivalence class of weights, in the NN-mixer weights are partitioned into equivalence classes of mix-contexts. Effectively, this raised the number of weight equivalence classes by a factor of 10^3. However, no theoretical investigation of weight selection mechanisms for PAQ has taken place up to today, all progress was made by manual optimization.

We will now turn our attention to implementational aspects of PAQ and its integration into the existing AT3D implementation.

3.1.5 PAQ8 - Modeling tools

The source code of PAQ is very modular, which allowed us to implement a modified version, tailored to the requirements of AT3D. PAQ provides numerous highly optimized modeling tools, which we will introduce now. Unfortunately, some of these tools are specialized to byte-wise encoding of values, which means they are not suitable for the use in AT3D entropy coding. This is due to the fact, that pixel position encoding is performed bit-wise and grayscale values use 8 bits at most, depending on the applied quantization. Thus, we will only describe these tools very shortly and remark their incompatibility, where it applies. The available tools can be divided into three classes:

- State maps, which map a bit history state to a probability, thus serve as a predictor.
- Context maps, which serve as very complex models, as defined in 3.1.10. Their input is at least one context from one or more context classes, and they calculate at least one prediction depending on the context, the accumulated bit history in this context, and the context map type.
- Adaptive probability maps (APMs), which are the modeling tools for SSE.

In the following, we will translate implementational details from the source code to our mathematical framework, which we have developed so far. These formulas have not been presented in any publication regarding PAQ yet. Note that probabilities are internally represented by a 12-bit integer number, which causes many factors to be powers of two.

Definition 3.1.28 (State table)**.** A ***state table*** is a structure, which essentially maps bit histories to a ***state***. A state is an approximation of the number n_0 of zeros and the number n_1 of ones in a (context-dependent) bit history and is represented by a number $s \in \underline{255}$. The state is updated incrementally with processed bits from the input bit-stream (according to encountered contexts). The bit counts n_0 and n_1 are exact for $n_0 + n_1 \leq 4$ and for $n_0 + n_1 > 4$ both are approximated. If one of the counts n_i is large and an opposite bit is processed, a non-stationary state-update is applied, which implicitly discards i-valued bits from the bit history by reducing n_i significantly. The exact update rule may be reviewed in the source code of PAQ8.

Definition 3.1.29 (State map)**.** A ***state map*** is a structure, which maps a state $s \in \underline{255}$ from a state table to a probability estimate, thus it is a predictor. In the implementation, it only predicts a 1-bit, the opposing estimate is easily derived.
Let $n_0(s)$ be the number of zeros in the approximated bit history corresponding to s and let $n_1(s)$ the analogous number of ones. Initially, the following estimates are predicted:
If $n_i(s) = 0$, then $n_j(s)$ is multiplied by 2^{-10} for $i, j \in \{0, 1\}$ and $i \neq j$. Then

$$\mathrm{SM}_0 : \underline{255} \to [0,1], \qquad \mathrm{SM}_0(s) = \frac{n_1(s) + 1}{n_0(s) + n_1(s) + 2}.$$

This corresponds to the Laplace estimator. After the k-th prediction step, the state map is updated as follows:

$$\mathrm{SM}_k(s) = \mathrm{SM}_{k-1}(s) + 2^{-12}(x_{k-1} - \mathrm{SM}_{k-1}(s)) + 2^{-5}.$$

We will now investigate context maps. These structures are complex, predefined models, where only the contexts have to be customized. There are three implemented types with different characteristics. Unfortunately, all context map types are highly customized to byte-wise prediction, which makes them unsuitable for AT3D in their original implementation. Yet an adaption for variable-length predictions may be helpful for grayscale value-encoding in AT3D. We will only provide brief summaries of their properties.

Definition 3.1.30 (Run context map)**.** A ***run context map*** bases its predictions only on the length of a consecutive count of a bit value i in a context. A new context may only be set on byte boundaries.

Definition 3.1.31 (Small stationary context map)**.** A ***small stationary context map*** supports a single context class. In contrast to the other context maps it only keeps an implicit bit history per context by updating probabilities directly, based on the last prediction error.

Definition 3.1.32 (Context map)**.** ***Context maps*** are the most complex predefined structure in the implementation of PAQ. They allow predictions in multiple context classes. A run context map is included in each context map. Each context is set at byte boundaries. Context maps provide a sophisticated prediction mechanism and put out five predictions per context class in a single encoding step.

3.1.6 PAQ8 - Adaptive probability maps

The tools included in PAQ to implement secondary symbol estimation (SSE) are called adaptive probability maps (APM). These map a probability estimate computed by the NN-mixer and a context from a single context class to a new probability estimate, which is then fed into the arithmetic encoder. It is possible to use cascades of APMs or to combine the output of several APMs linearly. Yet, optimization of the SSE-stage is purely based on manual adjustments. APMs compute their probability output by linear interpolation as defined below:

Definition 3.1.33 (Adaptive probability map)**.** An ***adaptive probability map (APM)*** is defined as the following mapping:

$$\mathrm{APM}_i : [0,1] \times \mathcal{Z} \to [0,1]\,, \qquad \text{for } 0 \leq i \leq k.$$

The function APM_0 is the linear spline interpolant, which satisfies

$$\mathrm{sq}(p_j) = \mathrm{APM}_0(p_j), \quad \text{for } p_j = -2048 + 16j \quad \wedge \quad 0 \leq j < 33.$$

Let $\lambda \in \{2^a \mid 1 \leq a < 32\}$ be the ***learning rate*** of the APM. After calculating $\mathrm{APM}_i(p_i, z)$, with $p_j \leq p_i \leq p_{j+1}$, the APM is updated, such that it is the linear spline interpolant satisfying

$$\mathrm{APM}_{i+1}(p_l, z) = \begin{cases} \mathrm{APM}_i(p_l, z), & \text{for } l \in \underline{33} \setminus \{j, j+1\}\,, \\ \mathrm{APM}_i(p_l, z) + \frac{(1+2^{-16-\lambda})x_i - 2^{-15} - \mathrm{APM}_i(x_p, z)}{2^\lambda}, & \text{for } l \in \{j, j+1\}\,. \end{cases}$$

3.2 Integration of PAQ into AT3D

AT3D and PAQ were developed by different programmers for differing purposes. Therefore the software architectures of both programs are dissimilar, creating some challenges when merging both into one program, which we will call AT3D_PAQ for the remainder of this thesis. In this section we will give a brief overview of the new entropy coding stage in AT3D_PAQ from a purely technical point of view, and describe necessary changes applied to the original software architecture of PAQ. Note that structures in the figures of this section are named by their class names in the implementation, they do not necessarily correspond to notions and definitions from this thesis.

PAQ contains code, which is necessary for handling input files, detecting special file types, and writing the compressed output to an archive. Since this is not necessary in AT3D_PAQ, this code was removed first. An overview of the software architecture of the encoding unit of PAQ in its original implementation is given in figure 9.

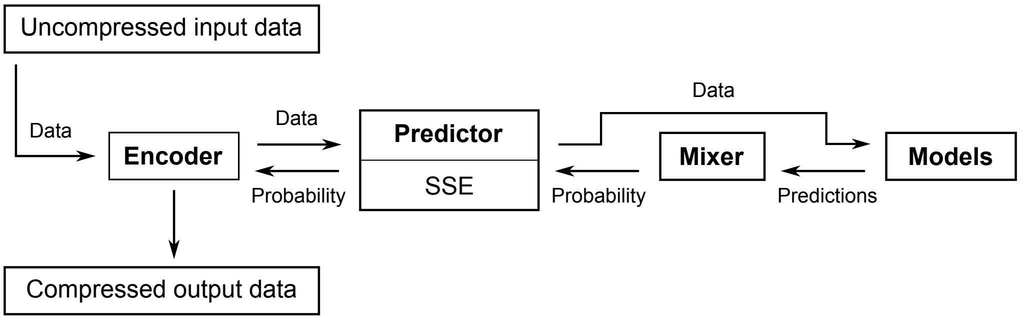

Figure 9: Schematic display of the original software architecture in PAQ. Class names from the implementation are displayed with boldfaced letters.

Two major architectural issues had to be solved:

1. PAQ was designed to either encode or decode data during one program execution. Thus an option to reset all models and structures was missing.
2. It was additionally designed to only rely on already processed data for prediction purposes. The input data feed level was strictly separated from the modeling level, as displayed in figure 9. Especially for grayscale value coding in AT3D such a link between input data level and modeling level is desirable in order to be able to access geometry information from the tetrahedralization.

The first problem was solved by implementing a `reset()`-function in the encoder-class, which calls a `reset()`-function in the predictor and is passed further down the structural chain of PAQ. These calls finally reach the lowest structures in the call-chain where again `reset()`-functions were implemented, allowing a full reset of all structures inside PAQ. This is the only solution to this problem without major architectural changes, because many functions use `static` variables, which can not be accessed from outside these functions.

The second issue was resolved by a similar idea: By using `set()`-functions, which are called by AT3D, the models inside PAQ gain access to the information via pointers to the video dimensions and the tetrahedralization. These pointers are member variables of `Encoder` and `Predictor`. This is visualized in figure 10.

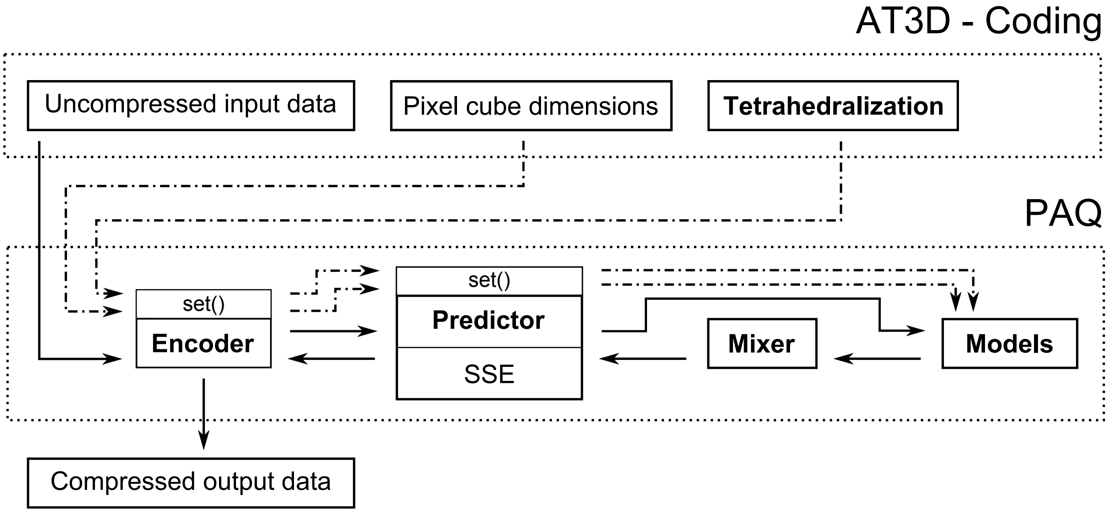

Figure 10: Schematic display of the merged software architecture in AT3D_PAQ. Class names from the implementation are displayed with boldfaced letters.

The interface between AT3D and PAQ is simple: Instead of calling the old entropy coding functions, new ones were created: they are named `encodePositionsPAQ()` and `encodeValuesPAQ()`. These create an `Encoder` object, whose `code()`-function is called for each bit to code. Between pixel position and grayscale value encoding, all PAQ-internal structures are reset. The decoding functions are named and work analogously.

3.3 Modifications of PAQ after the integration

After merging AT3D and PAQ, the remaining code of PAQ was cleaned up, so that it is divided into several code- and header-files in AT3D_PAQ instead of being contained in a single file. In addition to the already removed code, some unnecessary models (.exe, .jpeg, 8bit-bitmap, 24bit-bitmap, text) were deleted from the code base. As a result, the number of additional code lines added by PAQ was reduced from about 3700 to about 2100.

The next and most important step was to make use of the modular structure of the PAQ source code and to replace the original models by code tailored to the requirements of AT3D. This included the following measures:

- Removal of all models, which were irrelevant to compression performance,
- separation of pixel position encoding and grayscale value encoding,
- reordering of grayscale value encoding,
- design and implementation of customized models and mix-context classes for pixel position encoding and grayscale value encoding,
- tweaked weight initialization,
- custom SSE-stages for pixel position encoding and grayscale value encoding.

In the following sections we will describe the above modifications in more detail.

3.3.1 Modeling pixel positions

In this section we will describe which models we implemented into AT3D_PAQ for efficient pixel position encoding. When comparing the approaches to modeling in AT3D and AT3D_PAQ, we have to take their different architectures into account. AT3D only supports one model with a single context class and a small number of contexts. In particular, it only generates a single prediction. As a consequence the context class has to be chosen very carefully, and it is likely, that the context class is not capable of effectively modeling all video sequences. Our numerical experiments in section 3.4 will demonstrate, that this is exactly the case. A special sequence type is encoded very efficiently, while many others are not. Our goal is to achieve improvements for all types of sequences, and our numerical experiments will show, that we were successful.

Remark 3.3.1 (Importance of pixel position coding)**.** This thesis is equally focused on pixel position encoding and grayscale value encoding, despite the fact, that the latter is vastly more complicated. This is justified by the ratio of sizes of encoded pixel positions to the total compressed file size: With quantization setting `q1` this ratio is about 1/2, yet for the more practical `q16` it is close to 2/3 or even 3/4. Thus even significantly larger improvements on grayscale value encoding yield a smaller overall improvement, which is one of the aims of this thesis.

AT3D_PAQ relies on a large number of models and context classes. It is not necessary that all models provide precise predictions at all times. Instead, we may design models which only fit into particular situations and rely on the NN-mixer to give more weight to these models when they predict well. First we have to consider the following questions:

- Which is the most efficient pixel scanning order?
- Which dependencies arise from a scanning order?

Our goal is to group structural similarities between the pixel positions closely together on the one hand, and to make as much use of previously gained information as possible on the other hand. We then want to design contexts and models to exploit these dependencies and pieces of information. A starting point for a scanning order producing such a grouping is the AT3D implementation: Recall that there frames were scanned row-by-row and the first and last frame were encoded first, followed by the remaining frames in order. We will call the first and the last frames of blocks (or sequences in our case) ***bound frames***.
The test sequences presented later show a variety of significant pixel distributions across frames. Still sequences contain most significant pixels in bound frames, structured images have a mostly even distribution and for moving sequences the distribution is entirely dependent on the motion. Based on these observations, we propose three frame ordering schemes:

1. Natural order,
2. order by number of contained significant pixels,
3. bound frames first order.

The examples indicate, that the intra-frame scanning-order may be chosen arbitrarily, so we will keep it in a row-by-row order. To determine the most efficient frame order, we tested the above suggestions (after modeling) and obtained best results by the ***bound frames first ordering***. Low-entropy sequences benefited in particular, but all other sequences also showed equal or slightly better results compared to alternative frame orderings. We also tried an ordering, where the frames containing most significant pixels were encoded first, but this approach did not yield satisfying results.

Now we will describe our modeling approach: Again the examples suggest, that the projected context box-approach of the original implementation is very promising for still sequences, since most significant pixels in non-bound frames appear to be randomly distributed in the area ”shadowed” by the projected context boxes. The approach effectively reduces the area, in which significant pixels are predicted. For these reasons we implemented projected context box models into AT3D_PAQ.
As noted before, the performance of this approach is very likely to deteriorate for other sequence types. The test sequences suggest two further types of dependencies that may be exploitable: intra- and inter-frame structure. The former refers to clustering of significant pixels in structured areas of a frame, the latter to a similar clustering for objects with little translational motion or with changing textures, e.g. a structured rotating object, where the structure changes between frames. In order to exploit the former dependencies, we implemented restricted 2d context boxes. The latter are captured by 3d context boxes, which are restricted 2d context boxes extended by full projected 2d context boxes in previous frames.
One type of sequence is not ideally captured by the models described above: `Highway2` show a zooming-characteristic in the motion of its significant pixels with distinct groups of pixels following the same translational motion. This behavior clearly suggests the need for models, which capture this translational motion. To this end, we implemented a first

version of a significant pixel grouping detection in combination with a motion estimator. However, the numerical results yielded minimal advantages for `Highway2` and severely deteriorated performance for other sequence types, indicating that the above models are already capturing some of the existing dependencies (e.g. due to small translation between frames and largely overlapping pixel groups). We also considered global measures, such as (remaining) significant pixel counts in frames or the whole sequence, but these appeared to be inferior to local models.

We will now describe the implementation of our models. It employs state tables and state maps as follows:

Algorithm 9 Bit-wise model predictions using state tables

```
Let $S$ be a state table, $SM$ be a state map, and let $x_{k-1}$ be the last encoded bit
Let $c \in A = \underline{2^{32}}$ represent a context from the prediction of $x_{k-1}$
Let $t_c \in \underline{255}$ represent a bit-history state
Update bit history state $t_c$ according to last encoded bit $x_{k-1}$ and $S$
Calculate current context $c$
Obtain bit history state $t_c$ from $S$
return $\widehat{P}(x_k \mid x^{k-1}) = \mathrm{stretch}(SM(t_c))$
```

All models described below use the predictor from algorithm 9. The context box types were implemented as position-specific context boxes, where each individual bit of the number c represents a pixel position surrounding the currently predicted position, cf. figure 11. Essentially a bitmask is applied to the surrounding pixels using a bit-wise `AND`-operation to generate the number c. We implemented 11 different models in this class, which use the current, the last, the last but one, the first, and the last frame of a block and consider pixel positions q satisfying $\|p - q\|_\infty \leq 3$ when encoding p. The total number of these contexts is 8936352. For the detailed model implementations we refer to our source code.
Additionally we use the significant pixel count within the context box types corresponding to the five frame positions mentioned above as contexts c. This makes up five more models with a total of 75 contexts, because the largest considered number of significant pixels within a context box is capped at 15. Our mix-context classes are set as follows:

- Four classes are exact context boxes, making up 2056 contexts.
- Five classes are significant pixel counts in the five frame types mentioned above, yielding 75 contexts.
- Three classes are sums of significant pixel counts, making up 105 contexts.
- The total number of mix-contexts is 2236 in 13 classes.

Numerical experiments also yielded that pixel position encoding benefits from SSE significantly. Therefore we also implemented an exact context box model with radius 1 around the currently encoded position and use the significant pixel counts from several context box types into an SSE-stage modifying the context mixing prediction of the NN-mixer of AT3D_PAQ as shown in figure 12.

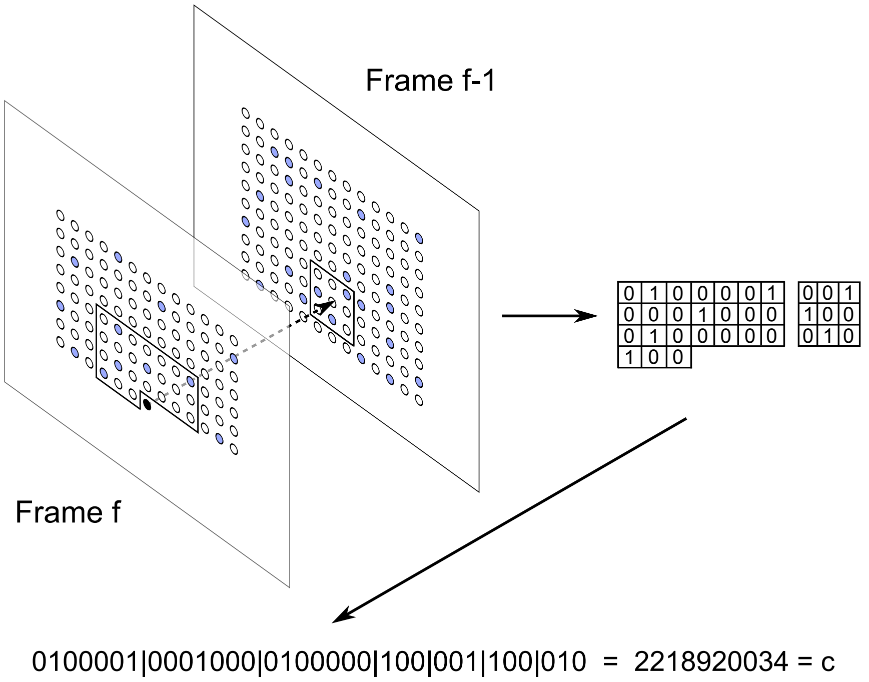

Figure 11: Visualization of position-specific context box model. From the context box a binary number *c* is generated, which is then used as a context.

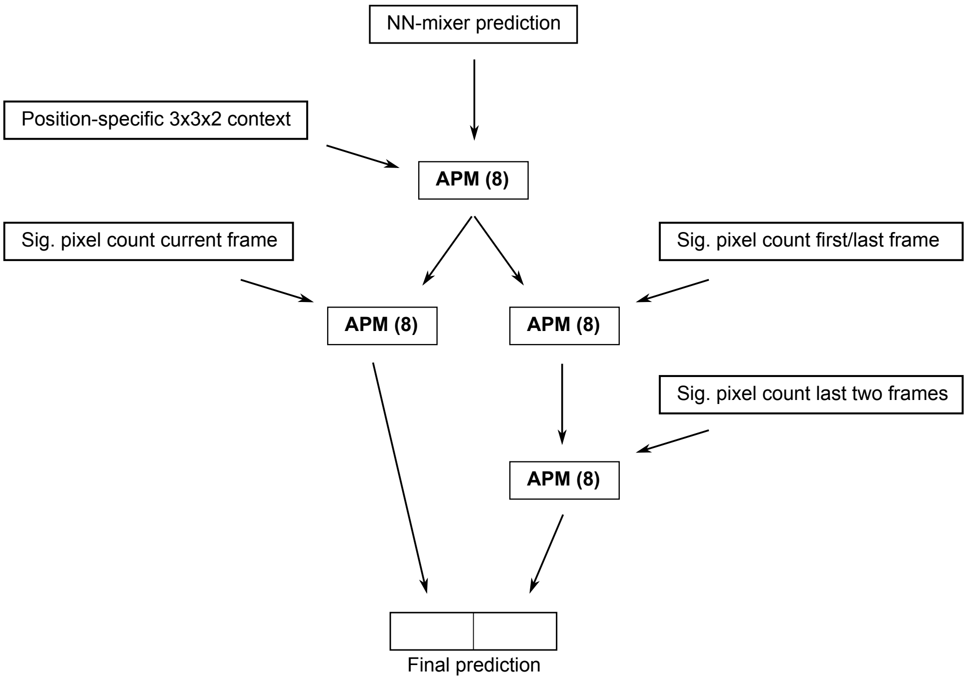
NN-mixer prediction
Position-specific 3x3x2 context
APM (8)
Sig. pixel count current frame
Sig. pixel count first/last frame
APM (8)
APM (8)
Sig. pixel count last two frames
APM (8)
Final prediction

Figure 12: Visualization of cascaded SSE-stage of pixel position modeling. Numbers in brackets denote the experimentally determined optimal learning rates λ.

The details regarding context box sizes and SSE-design were developed by numerical experiments and show the best results across all tested sequences.

Remark 3.3.2. Before removing the original models of PAQ from AT3D_PAQ, their impact on compression performance was tested. None of the test video sequences benefited from those anymore. This is a clear indication, that the employed models capture existing redundancies very well and no remaining 'arbitrary' redundancies exist in the pixel positions.

3.3.2 Modeling grayscale values

In this section we will describe our approach to grayscale value modeling. We will start with some important remarks regarding the differences to pixel position encoding and then proceed with a presentation of our models.

An implementation of models for grayscale values is fundamentally different from the modeling introduced in the previous chapter. Grayscale values have a fixed length of ρ bits, where 2^ρ is the number of colors used in a sequence. Note that opposed to the arithmetic coder employed in AT3D, the arithmetic encoder implementation of PAQ only allows encoding single bits per encoding step. As a consequence, each bit in the grayscale value is modeled separately, but not independently of the others.

The first choice to be made before any model can be developed regards the values to be encoded: The natural approach is plain grayscale value encoding, where ρ bits have to be encoded per value. On the other hand the two-dimensional adaptive thinning implementations (e.g. in [Hin10]) rely on coding of value differences along edges. The latter choice was based on numerical results, which indicated that grayscale value differences between two vertices as a function of the edge length of the edge connecting these vertices is a decreasing function.

Remark 3.3.3. Note that when encoding grayscale value differences, $\rho + 1$ bits have to be encoded per value, because the difference $d \in \{-(2^\rho - 1), \dots, 2^\rho - 1\}$. Thus the sign bit has to be coded additionally. The entropy of this additional sign bit is likely to be close to 1 bit, since for a large range of values either sign is equiprobable.

First numerical tests indicated, that encoding grayscale value differences in an order governed by the longest available edge length yielded acceptable results for very few examples and produced significantly worse results than AT3D for most sequences. Further investigations showed, that the number of relatively short edges with large grayscale value difference averages $\mu \approx 35$ and standard deviations $\sigma \approx 40$ make up a large portion of the overall value numbers. Therefore due to the additional sign bit and the large deviations of the difference values an efficient coding is not possible. Instead, investigations indicated that a different approach was more promising: relying on neighborhood information.

Our new approach for AT3D_PAQ predicts grayscale values (not differences) in an order based on the number of already encoded neighbor values. Recall the notion $E_V^k \subset E_V$,

which is the set of vertices, whose grayscale values have already been encoded. In each iteration, the value at vertex $y \in E_V \backslash E_V^k$ with the largest number of neighbors $y_k \in E_V^k$ is chosen to be encoded next. If y_k is not unique, the vertex with the largest average edge length of all its connecting edges is chosen. This approach was compared to the one from [Hin10] based on several test sequences using simple models adapted to either approach and proved to be significantly more efficient.
We introduced an algorithm allowing a very efficient selection of the significant pixel with most encoded neighbors. It keeps one linked list N_i for each number i of computed neighbors of non-computed significant pixels and save i for each of these. After encoding a grayscale value at a pixel position, all of its non-computed neighbors p_j are removed from the list N_i they were in and added to N_{i+1}. The computed pixel is then removed from the list it was in. The following value to be encoded is then selected from the list i_{max}. It contains the pixels y_k with most computed neighbors and selects the next grayscale value by calculating the average edge length of the edges each y_k is a vertex of and selecting the pixel with the largest average. While the addition to a list is a constant time operation, runtimes for removal and next pixel selection are dependent on the average list size $M \ll N^2$. The quantity M is hard to estimate, because it is geometry-dependent. Due to our satisfying numerical results regarding compression time, we will settle with noting, that the selection algorithm has sub-quadratic complexity.

We will now describe our modeling approach for grayscale value encoding. We use two main ideas: All models predict an estimate of the encoded grayscale value v, which is then encoded bit-wise. The probabilities for all individual bit predictions are set manually, reflecting that the first bits are more likely to be predicted correctly than the last ones. The employed models can be divided into several classes:

- Single value predictions,
- predictions based on incidental tetrahedra,
- context-based predictions,
- unmodified DMC- and chart-model from PAQ8kx.

The first two types develop a ρ-bit grayscale value prediction $\widetilde{v}$ before encoding the first bit of the true grayscale value v. Each individual bit prediction $\widetilde{v}_i$ of $\widetilde{v}$ is then generated by the formula

$$\widetilde{v}_i = \pm f_i,$$

with

$$300 \leq f_i \leq f_j \leq 1500, \qquad \text{for } 0 \leq j < i \leq \rho.$$

The values f_i were chosen based on manual optimization and denote predictions in the logistic domain, the sign is negative, if a 0 is predicted and positive, if a 1 is predicted. The single value predictions are simply

- the grayscale value at the pixel with the longest edge connected to the current pixel,
- the average of already encoded neighboring grayscale values.

The second model-class calculates predictions based on previously encoded grayscale values in tetrahedra incidental to the currently encoded pixel. This is motivated by observations, which suggested that tetrahedra with several closely grouped grayscale values tend to have a similar grayscale value at the currently encoded position. These predictions are calculated as presented in algorithm 10.

Algorithm 10 Grayscale value prediction in AT3D_PAQ: Cell-based prediction generation

```
Let $v$ be the grayscale value at pixel position $p$ to be encoded
$\Delta_v := 15$
for each tetrahedron $T_j \in \mathcal{C}_p$ do
    Determine number $n_j$ of already encoded grayscale values $w_k$ in $T_j$
    Sort $w_k$ for $0 \leq k < n_j$
    Determine largest count $\widetilde{n}_j$ of $w_k$ satisfying $|w_k - w_l| < 2\Delta_v$ for $k \neq l$ and choose
      $\widehat{v}_j$ as the average value of this largest group
    if i-th bit of $\widehat{v}_j$ is 1 then
        $\mathrm{sgn}_j := 1$
    else
        $\mathrm{sgn}_j := -1$
    end if
    if $(n_j = 3$ and $\widetilde{n}_j \geq 2)$ or $(n_j = 2$ and $\widetilde{n}_j = 2)$ then
        $\widehat{p}_j := 1500\ \mathrm{sgn}_j$
    else if $(n_j = 2$ and $\widetilde{n}_j = 1)$ or $(n_j = 1)$ then
        $\widehat{p}_j := 800\ \mathrm{sgn}_j$
    else
        $\widehat{p}_j := 300\ \mathrm{sgn}_j$
    end if
end for
return Sequences $(\widehat{p})_j$ of probability estimates and $(\widehat{v})_j$ of corresponding predicted
    average values with $j \in \underline{|C_p| - 1}$
```

In algorithm 10 the probability estimates $\widehat{p}_j$ indirectly represent the confidence in the prediction of a bit, and are then further modified depending on n_j and $\widehat{v}_j$, cf. algorithm 11. Again the modification factors were found by manual optimization. For the first bit, we reduce the estimate $\widehat{p}_j$, if $\widehat{v}_j$ is close to $2^{\rho-1}$, since in this case the first bit of $\widehat{v}_j$ cannot be predicted safely. Additionally, we check for $0 < i < 3$, if the previous bit was predicted correctly. If it was not, we still assume our prediction was close and consequently invert the following bit-prediction, which improves compression of predictions $\widehat{v}_j \approx m2^{\rho-i-1}$ for $m \in \mathbb{N}$. One of the disadvantages of this algorithm is, that the number of predictions varies with $|\mathcal{C}_p|$, which means that weights for the last predictions in the NN-mixer are inaccurate. This motivated the final sorting procedure. Yet, this type of prediction improves compression on several sequences despite its shortcomings. This is most likely due to the fact, that the NN-mixer either uses the full model class, or no predictions from it, depending on the sequence.

Algorithm 11 Grayscale value prediction in AT3D_PAQ: Bit-wise cell-based predictions

```
Let $v$ be the grayscale value at pixel $p$ to be encoded, $N = |\mathcal{C}_p|$, and $\{\widehat{p}_j \in \mathbb{N} \mid 0 \le j < N\}$ and $\{\widehat{v}_j \in \underline{2^\rho - 1} \mid 0 \le j < N\}$ be sets of probability estimates in the logistic domain and corresponding average values computed by algorithm 10
Let $0 \le i < \rho$ be the number of the currently encoded bit
if previously encoded bit was predicted correctly then
    $o_{j-1} := 1$
else
    $o_{j-1} := -1$
end if
for $0 \le j < N$ do
    if $(i = 0$ and $|\widehat{v}_j - 2^{\rho-1}| < \frac{2^\rho}{5})$ then
        $\widehat{p}_j := \frac{\widehat{p}_j}{4}$
    else if $(i = 0$ and $|a_j - 2^{\rho-1}| \ge \frac{2^\rho}{5})$ then
        $\widehat{p}_j := 1.2\widehat{p}_j$
    else if $i = 1$ then
        $\widehat{p}_j := 1.2\widehat{p}_j o_{j-1}$
    else if $i = 2$ then
        $\widehat{p}_j := 1.4\widehat{p}_j o_{j-1}$
    else if $i > 2$ and $i < 6$ then
        $\widehat{p}_j := \frac{\widehat{p}_j}{5}$
    else
        $\widehat{p}_j := \frac{\widehat{p}_j}{7}$
    end if
end for
Sort $\widehat{p}_j$ descending in the corresponding $n_j$ and predict them to the NN-mixer, i.e. predictions based on a larger number $n_j$ are predicted first
```

The context-based predictions employ algorithm 9, analogous to the pixel position contexts. We implemented the following context classes:

- The last encoded bit x_{k-1},
- the grayscale value $v_{e_{max}}$, whose pixel position is connected to the currently encoded pixel p along the longest edge incidental to p with length e_{max} - each bit position has its own context value,
- same as above, but $v_{e_{max}}$ is bit-shifted to the right with at most 5 bits remaining,
- edge length e_{max} divided by 10 and capped at 200.

Additionally, there are two more model classes which predict the same values as the single value prediction models, but use state maps as described in algorithm 9. These only yield minimal additional improvements, therefore we refer to the source code for details regarding their implementation.

In order to make use of the weight selection mechanisms of the NN-mixer, we furthermore implemented the following mix-context classes:

- Last encoded byte (independent of ρ),
- number of already computed neighbors,
- current bit position,
- number of predictions with $n_j = 3$,
- quantized cell average edge length, average value and their standard deviations,
- first bits of $v_{e_{max}}$ with individual contexts for each bit position.
- Total number of mix-contexts is 2135 in 10 classes.

Our SSE-stage is very simple, it contains a single APM with an update rate of 9, which uses the current bit-position as a context and yields improvements of about 1-1.5%.

3.4 Numerical results: AT3D_PAQ vs. AT3D

In this section we will present the central results of this thesis, the compression results of AT3D_PAQ. We obtained improved overall compression results in all considered examples. In 0.24% of the conducted tests the pixel position size grew by at most 1.54%, all other examples showed an improvement. The improvements depend strongly on the number and the distribution of significant pixels for pixel position coding and on the geometric properties of the resulting tetrahedralization and the quantization setting for grayscale value coding. We will show which example classes benefited most, and which ones did not. The considered test video sequences are of different characteristics, ranging from sequences with little structure and motion to high-entropy sequences containing complex structures and a large amount of motion. Some of these are real sequences, others are cartoon sequences from the movie ***Sita sings the blues*** by Nina Paley. It is published under the Creative Commons license (CC0), which allows us to use it as we wish. Additionally we considered geometric sequences, generated by a self-developed sequence generator, allowing us to investigate improvements on simple affine transformations of objects like polygons and ellipses. A complete list of the employed test sequences is presented in appendix B. After describing the test environment, we will begin this chapter with a summary of the overall results, continue with some observations regarding the pixel position and grayscale value encoding results, and then present a detailed view on selected examples. The majority of the compression results are listed in appendix C, for the full table containing all results we refer to the disc which is supplied with this thesis.

Remark 3.4.1 (Sequence naming conventions)**.** Most sequences we considered are in `qcif`-resolution and 30 frames long. These are simply denoted by their name in verbatim letters. A different video length is denoted by appending `-[NumOfFrames]` to the sequence name, a different resolution by appending `_[Res]` to it. The various resolution abbreviations are listed in appendix B. Sequences from the cartoon movie *Sita sings the blues* are named with the prefix `SSTB` directly followed by the number of its first frame in the full movie.

The test settings can be summarized as follows:

- A total of 44 test sequences was considered. These include natural, cartoon, and artificial sequences, frame sizes vary from 50 × 50 pixels to 704 × 576, and frame numbers from 6 to 120.
- The sequences were chosen such that they contain various types and amounts of content, structure, and motion. A rough summary of these properties is also listed for each sequence in appendix B.
- Each sequence was encoded in 6 to 8 steps to achieve PSNR-values ranging from above 40 dB to below 30 dB, which ranges from very good to very poor visual quality. The number of removed pixels in each step was chosen manually depending on the sequence. In each step the significant pixel distribution from the previous encoding step was used as the starting point for adaptive thinning.
- In each step, 3 different quantization settings were used, `q1`, `q4`, and `q16`.
- The total number of results is 864. These were generated by using batch scripts and then extracting the resulting numbers by a tool developed by the author of this thesis. All employed tools are included on the appended disc.
- All sequences were encoded in a single block of frames. All results refer to plain adaptive thinning output without any pre- or post-processing except for calculating the best approximation.
- The following set of parameters was used to generate the results:
  ```
  at3d.exe -co -c%numOfThinnedPx% -a6 -o1 -g-1 -p -m -l -f600 -%quantSetting%
  %inputFile.pgmv% %nodesFile.nodes3d%
  ```

The overall average results are summarized in the following table. For the remainder of this section all improvements are relative to the AT3D results, positive numbers indicate that AT3D_PAQ yielded an improvement, negative numbers indicate a deterioration.

Encoding stage	Improvement [%]		
	Average	**Minimal**	**Maximal**
Pixel positions	14.95	-1.54	55.48
Grayscale values	16.66	5.61	60.46
Total	15.24	1.58	56.06

Table 4: Overall average compression improvements of AT3D_PAQ over AT3D, split into pixel position, grayscale value, and total encoding improvement.

A closer analysis of these results is necessary, and we will present it later. First we begin with partitioning the grayscale value encoding results into the different quantization setting results:

Quantization setting	**Avg. improvement [%]**
q1	14.40
q4	13.66
q16	21.93

Table 5: Average grayscale value encoding improvements, split into three tested quantization values.

Clearly the encoding of values quantized with `q16`, resulting in 4-bit values benefited most from our optimizations. This is a satisfying result, since for low bit-rate encodings setting quantization to `q16` still only results in negligible deteriorations of the PSNR-values.

3.4.1 Pixel position encoding results

As our results in table 4 showed, the improvement varies greatly depending on the individual sequences. In this section we will present pixel position encoding results for selected sequences, which suggest pixel position distribution characteristics that are compressed more efficiently by AT3D_PAQ. We will begin with an analysis of the problematic results we obtained compressing the `Suzie`-sequences and `MissAmerica`.
In this setting the bound frames of a block contain significantly more information than the remaining frames (cf. figure 14), and pixels in these are very likely to be within the same area of a frame, that contained significant pixels in the bound frames. Yet, within this area, their distribution appears to be random, resulting in minimal improvements or even a small deterioration by AT3D_PAQ over AT3D, when considering pixel position encoding. This is the only situation among all test cases, where this may be observed. An example showing the same pixel distribution, but significantly more improvement, is `SSTB94115`. This is due to the distribution of significant pixels within the frames. For all non-bound frames in the former cases the pixel distribution appears to be random, while in `SSTB94115` clusters of significant pixels are formed within the frames due to the existing texture. Example frames are displayed in table 6. This indicates, that we achieved major improvements for sequences of frames containing complex texture. Another indicator for that is the fact, that removing all models aimed at capturing local structures from AT3D_PAQ has a minimal impact on the compression results on `Suzie` and `MissAmerica`. We will further substantiate this claim with more numerical evidence later.

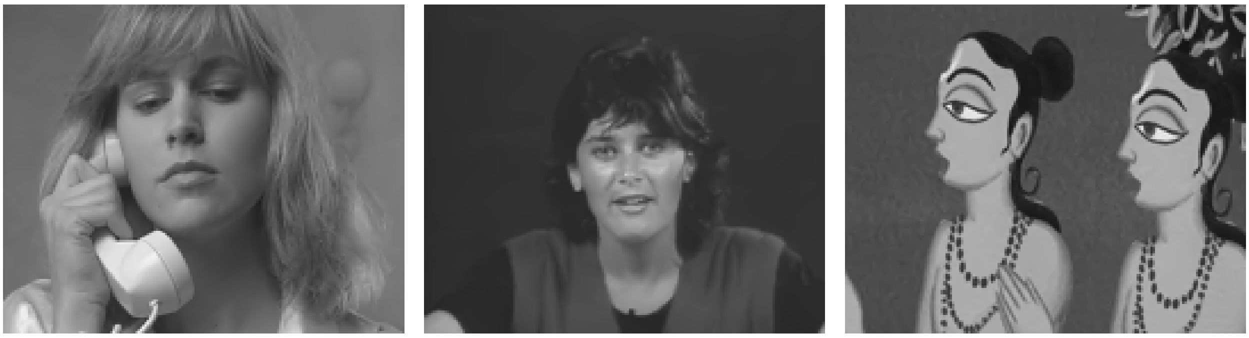

Figure 13: First frames of `Suzie`, `MissAmerica`, and `SSTB94115`.

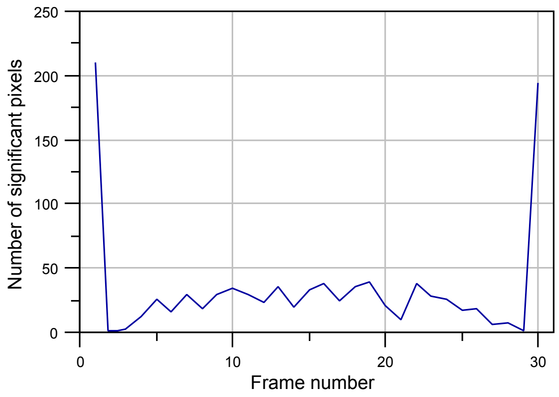

Figure 14: Significant pixel distribution in `Suzie` with 1020 significant pixels.

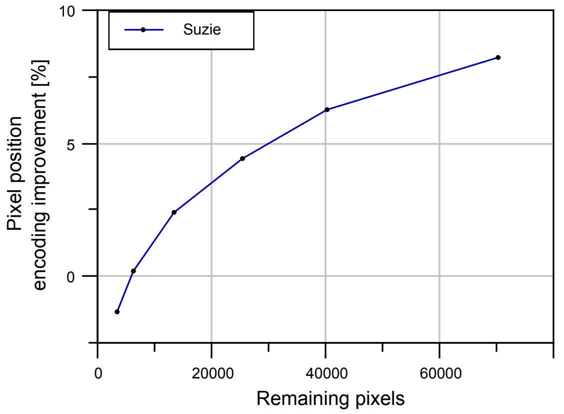

Figure 15: Pixel position encoding improvement of AT3D_PAQ vs. AT3D on `Suzie`.

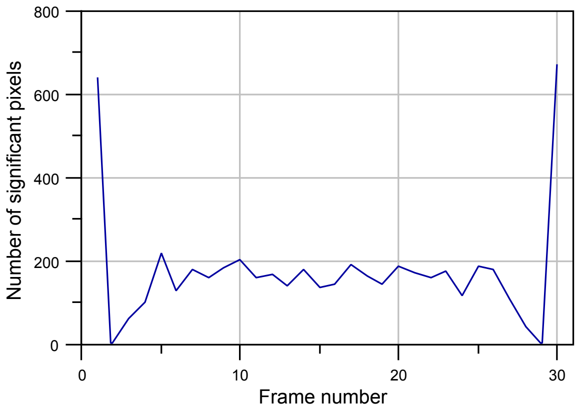

Figure 16: Significant pixel distribution in `SSTB94115` with 5319 significant pixels.

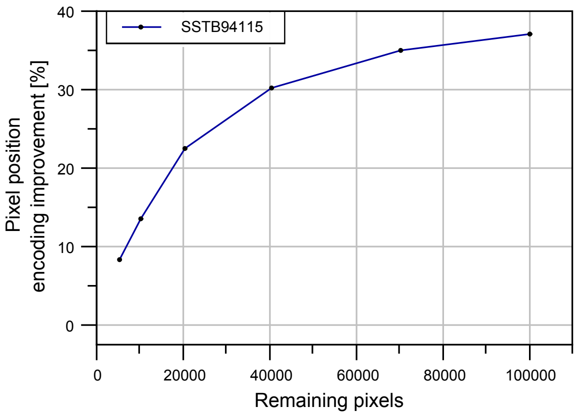

Figure 17: Pixel position encoding improvement of AT3D_PAQ vs. AT3D on `SSTB94115`.

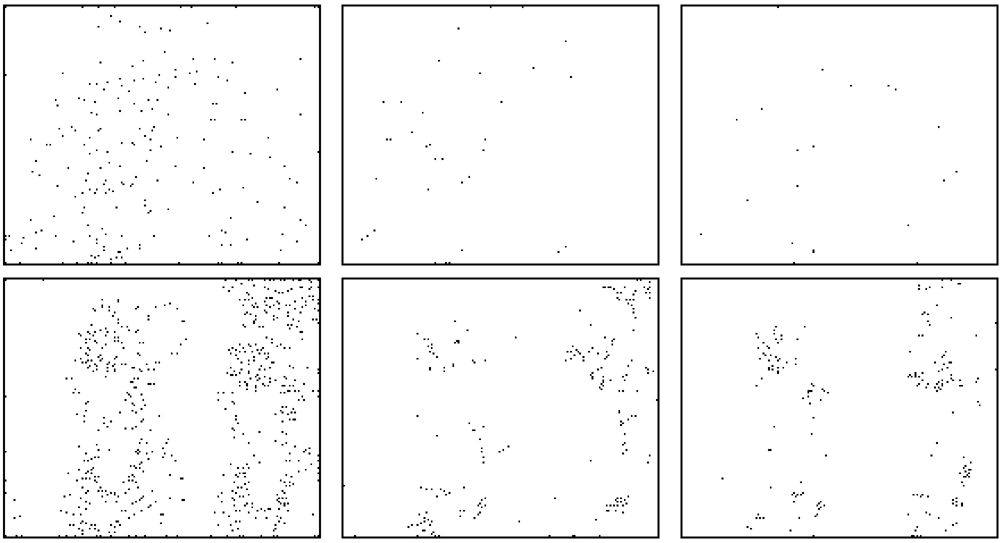

Table 6: Significant pixels in frames 0, 9 and 19 from `Suzie` (top) and `SSTB94115` (bottom).

Remark 3.4.2 (Improvements decreasing in the number of removed pixels)**.** In 82% of the tested sequences the improvement of pixel position coding decreases, as more pixels are removed from the tetrahedralization. In general, this is caused by pixel clusters getting thinned. The other examples are in general very high entropy sequences, which still contain a large number of significant pixels for low quality output, which allow improvements by the local pixel position models of AT3D_PAQ.

The above observations may also be applied to the most improved test sequence, `TranslEll_100x100`. Note that no motion estimation was implemented in AT3D_PAQ, yet the existing projected context boxes captured some inter-frame dependencies. One of the reasons is the slow translational speed of the ellipse. Another one is, that the motion only induces false predictions and the left boundary of the ellipse (which moves towards the right), since in all other positions significant pixels remain within the context boxes projected onto previous frames.

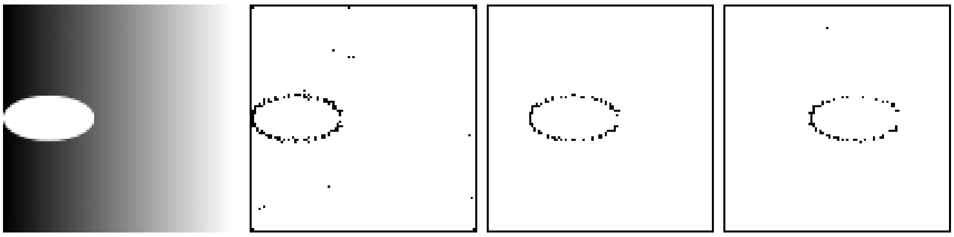

Figure 18: First frame and significant pixels in frames 0, 9 and 19 in `TranslEll_100x100`.

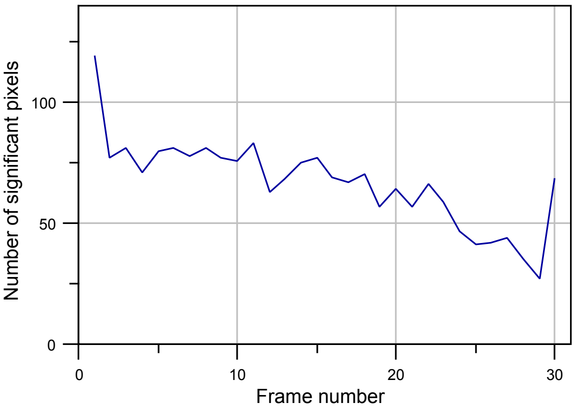

Figure 19: Significant pixel distribution in `TranslEll_100x100` with 2000 sign. pixels.

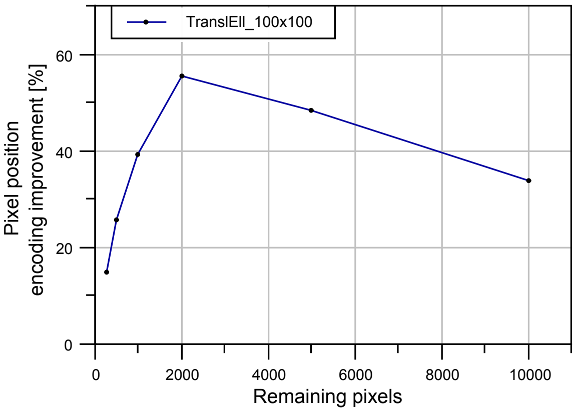

Figure 20: Pixel pos. encoding improvement of AT3D_PAQ vs. AT3D on `TranslEll_100x100`.

On the next page we will present a qualitative comparison of pixel position coding cost for three different sequences. It visualizes the characteristics of both implementations, yet it is hard to deduct consequences for compression performance optimization from it.

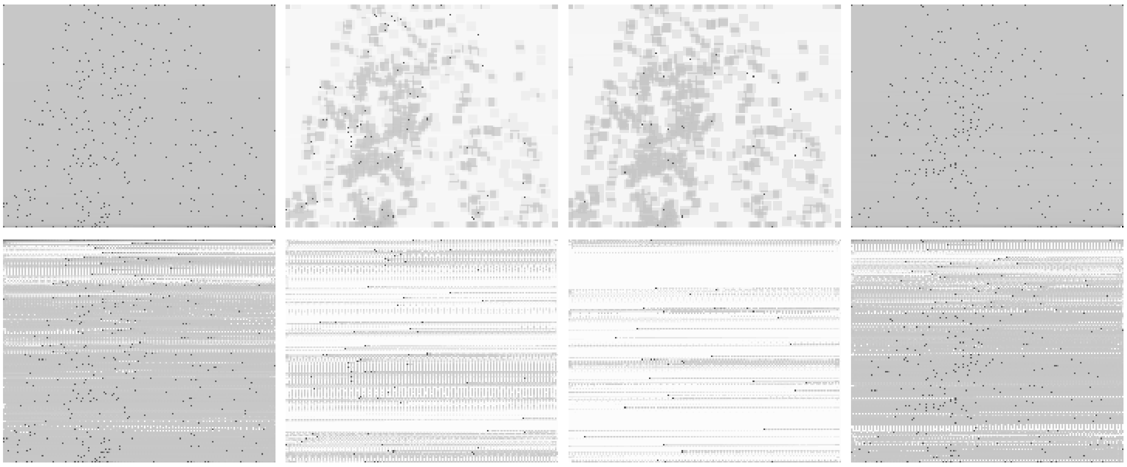

Table 7: Qualitative pixel position coding cost visualization for `Suzie` with 1820 sign. pixels. Top: AT3D, bottom: AT3D_PAQ. Darker colors indicate higher coding cost, scale is nonlinear.

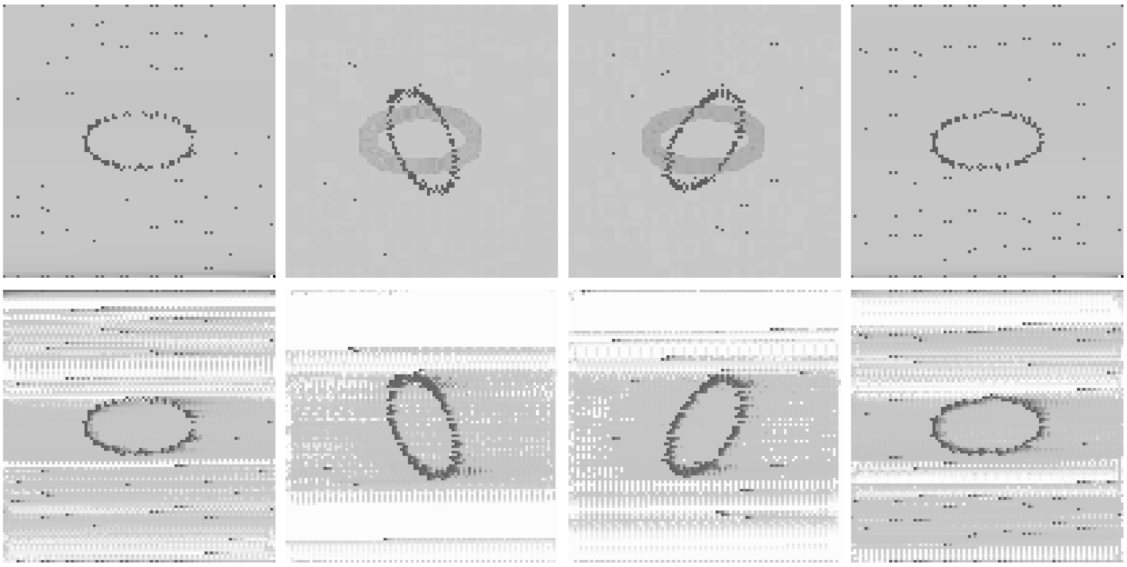

Table 8: Qualitative pixel position coding cost visualization for `RotEll_100x100` with 5000 sign. pixels. Top: AT3D, bottom: AT3D_PAQ. Darker colors indicate higher coding cost, scale is nonlinear.

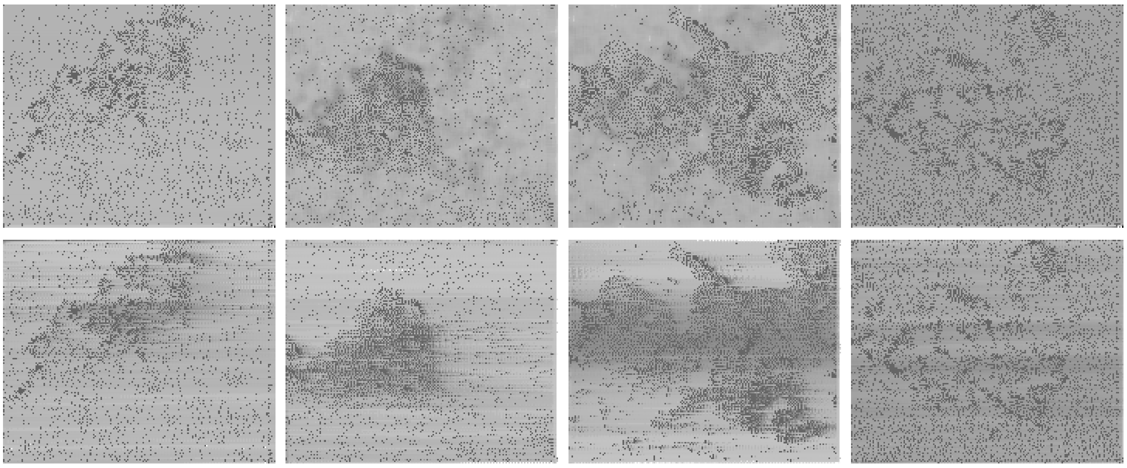

Table 9: Qualitative pixel position coding cost visualization for `Football` with 120319 sign. pixels. Top: AT3D, bottom: AT3D_PAQ. Darker colors indicate higher coding cost, scale is nonlinear.

We can summarize the observations from this section as follows:

- Pixel position encoding with AT3D_PAQ captures local dependencies caused by texture in frames significantly better than AT3D.
- Sequences, which have close to random significant pixel distributions and little structure rarely benefit from AT3D_PAQ.
- Motion is captured significantly better by AT3D_PAQ due to its 3d context boxes.
- For low-entropy sequences with a small number of remaining significant pixels, improvements diminish.
- In general, higher resolution sequences benefit more than their corresponding low resolution counterparts.
- Improvements remain stable, when changing the number of frames of a sequence by a factor of at most 2. For a larger factor, e.g. for `Suzie-90`, results are slightly improved compared to the shorter versions.

3.4.2 Grayscale value encoding results

The analysis of the grayscale value encoding results is significantly more difficult than the previous one. Similar to pixel position encoding, the minimal and maximal improvement varies immensely, depending on the sequence and quantization setting used.

- A connection between improvements and visible characteristics of sequences cannot be made, yet artificial sequences benefit most.
- The worst improvement is 5.61%, which is still a considerable gain.
- Improvements decrease only for 13% of the test sequences, as the number of significant pixels decreases. In the remaining cases they mostly remain stable.
- Video size and length have a negligible effect on improvements. The 90-frame sequences tend to improve slightly more than their shorter counterparts.

Instead of presenting the worst- and best-case sequences here, we will continue with an overview of total compression for selected sequences and present our summary regarding grayscale value encoding in the overall result summary.

3.4.3 Total compression results for selected sequences

On the following pages we will present test sequence results, which show interesting characteristics. The results include a comparison of AT3D and AT3D_PAQ at different thinning stages per sequence and a representative distribution of significant pixels at one thinning stage. Additionally the geometry of significant pixels is visualized in combination with the compressed frame output for selected frames of the sequence. Finally we present rate distortion graphs and a comparison of improvements of AT3D_PAQ over PAQ for different quantization settings and pixel position encoding at the various thinning stages.

Highway2						
Sign. pixels	120320	60320	35320	20320	12320	7320
Px. pos. old [b]	56807	36667	25041	16559	11203	7375
Px. pos. new [b]	44647	28497	19811	13401	9344	6412
%	**21.41**	**22.28**	**20.89**	**19.07**	**16.59**	**13.06**
Val. `q1` old [b]	102835	54737	32903	19283	11760	6970
Val. `q1` new [b]	88198	46935	28096	16272	9774	5733
%	**14.23**	**14.25**	**14.61**	**15.61**	**16.89**	**17.75**
Val. `q4` old [b]	72952	39608	23971	14088	8560	5044
Val. `q4` new [b]	61148	33371	20157	11683	6998	4104
%	**16.18**	**15.75**	**15.91**	**17.07**	**18.25**	**18.64**
Val. `q16` old [b]	44007	24844	15285	9063	5478	3196
Val. `q16` new [b]	32015	18708	11578	6732	3975	2302
%	**27.25**	**24.70**	**24.25**	**25.72**	**27.44**	**27.97**

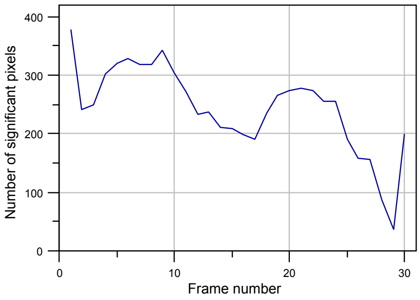

Table 10: Compression results and sign. pixel distribution for `Highway2` at 7320 pixels.

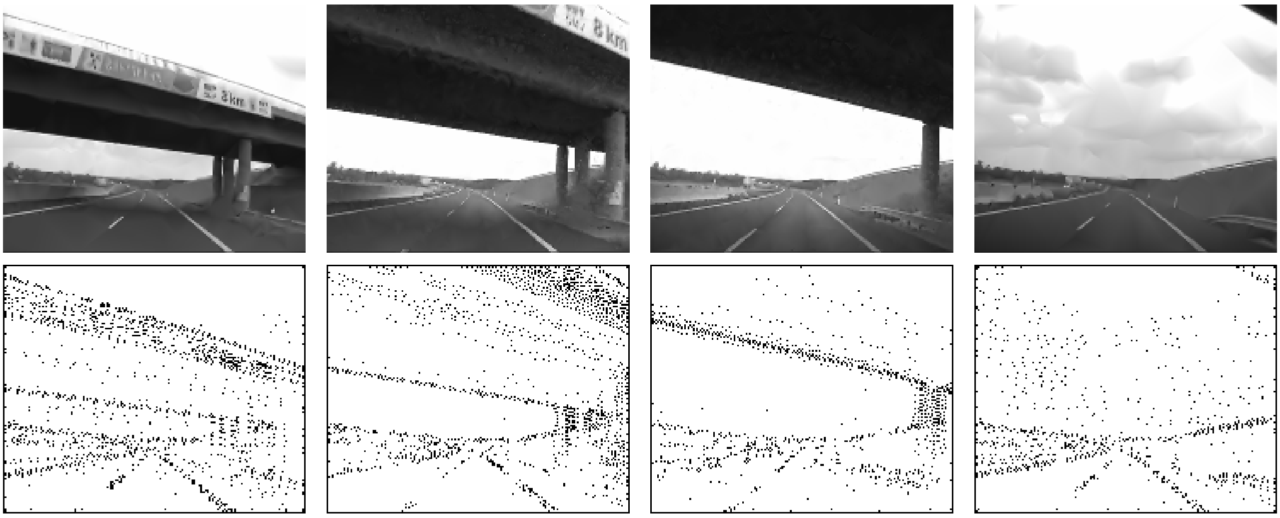

Table 11: 35320 sign. pixels, `q4`, PSNR 35.1 dB. Top: Reconstructed frames 0, 9, 19, 29. Bottom: Sign. pixels.

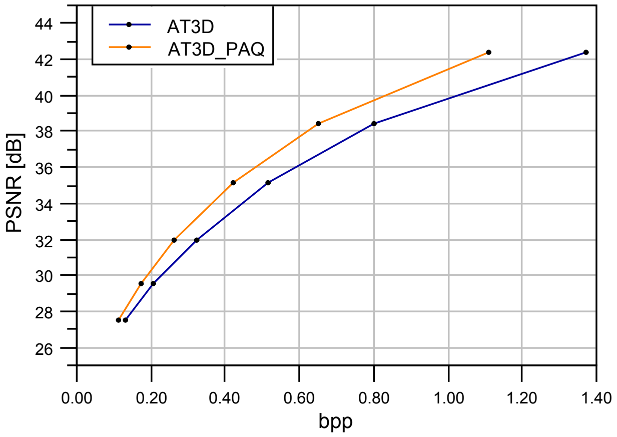

Figure 21: Comparison of bits per pixel vs. PSNR of old and new implementation at `q4`.

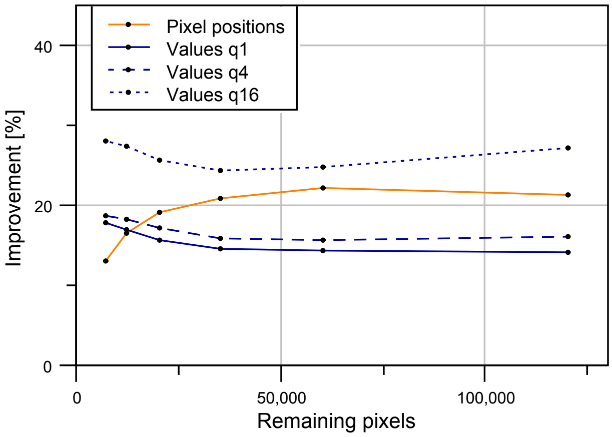

Figure 22: Improvement of AT3D_PAQ vs. AT3D on `Highway2`.

SSTB1450						
Sign. pixels	80320	55320	35320	20320	10320	5320
Px. pos. old [b]	45958	35631	25662	16767	9769	5666
Px. pos. new [b]	39984	32600	24455	16347	9600	5624
%	**13.00**	**8.51**	**4.70**	**2.50**	**1.73**	**0.74**
Val. `q1` old [b]	58435	40668	26525	16265	8965	4837
Val. `q1` new [b]	47218	33154	22160	13997	7825	4238
%	**19.20**	**18.48**	**16.46**	**13.94**	**12.72**	**12.38**
Val. `q4` old [b]	38970	27152	17785	11114	6248	3386
Val. `q4` new [b]	29757	21120	14484	9580	5530	3001
%	**23.64**	**22.22**	**18.56**	**13.80**	**11.49**	**11.37**
Val. `q16` old [b]	22810	16075	10709	6762	3806	2054
Val. `q16` new [b]	14050	10112	7208	4916	2991	1668
%	**38.40**	**37.09**	**32.69**	**27.30**	**21.41**	**18.79**

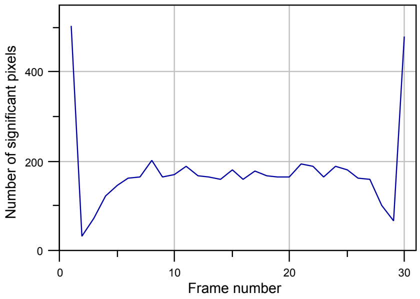

Table 12: Compression results and sign. pixel distribution at 5320 pixels.

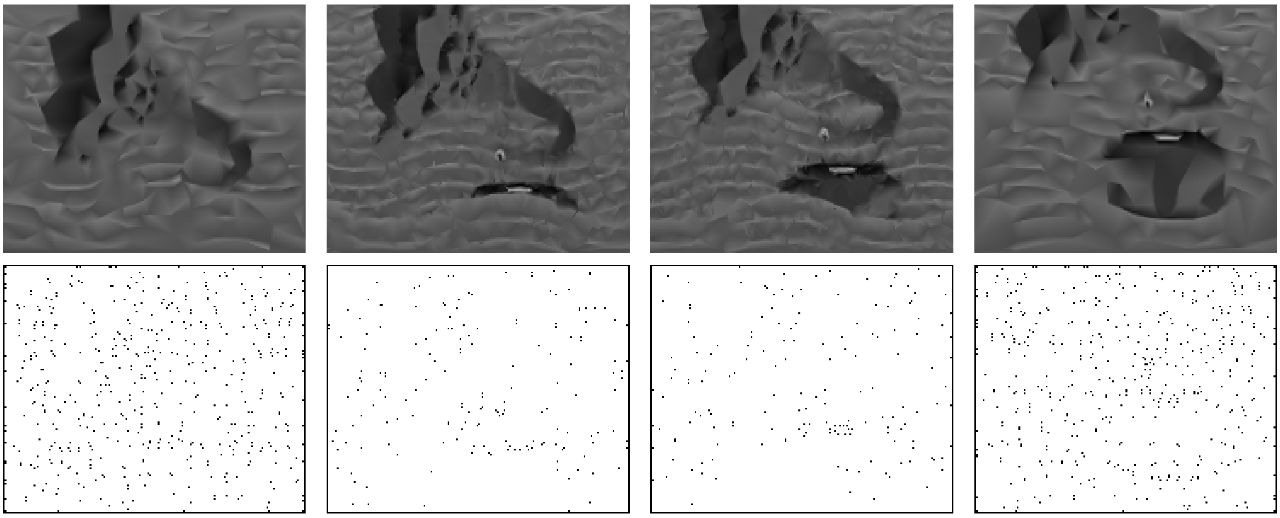

Table 13: 5320 sign. pixels, `q4`, PSNR 28.15 dB. Top: Reconstructed frames 0, 9, 19, 29. Bottom: Sign. pixels.

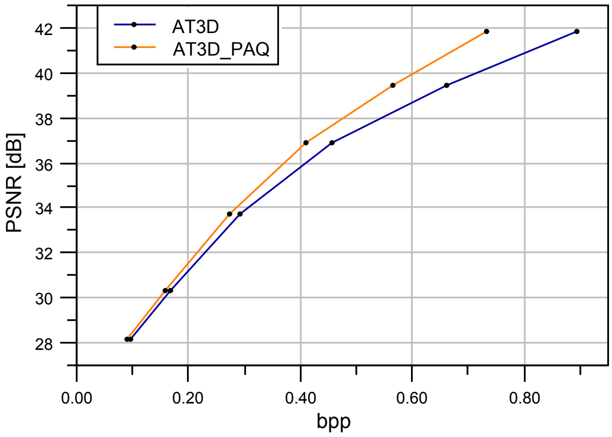

Figure 23: Comparison of bits per pixel vs. PSNR of old and new implementation at `q4`.

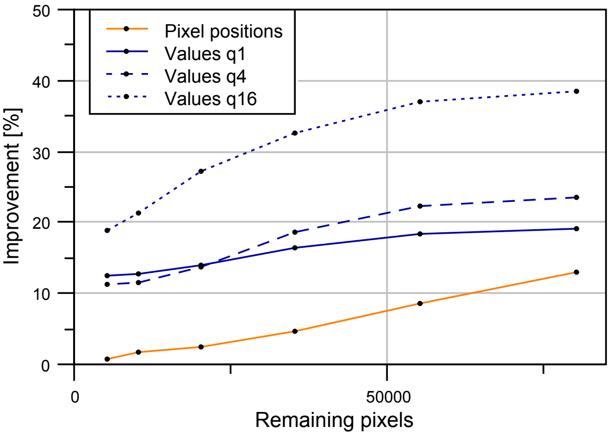

Figure 24: Improvement of AT3D_PAQ vs. AT3D on `SSTB1450`.

`RotGrowBox-120_50x50`						
Sign. pixels	50000	35000	25000	17000	10000	6000
Px. pos. old [b]	23964	19280	15425	11760	7917	5321
Px. pos. new [b]	11578	11374	9192	5578	4328	3538
%	**51.69**	**41.01**	**40.41**	**52.57**	**45.33**	**33.51**
Val. `q1` old [b]	16008	12365	11982	9318	6165	4734
Val. `q1` new [b]	7549	5193	5132	3684	2606	2743
%	**52.84**	**58.00**	**57.17**	**60.46**	**57.73**	**42.06**
Val. `q4` old [b]	13803	10690	9410	7453	4629	3232
Val. `q4` new [b]	7418	5415	4656	3362	2076	1773
%	**46.26**	**49.35**	**50.52**	**54.89**	**55.15**	**45.14**
Val. `q16` old [b]	11096	8538	6937	5390	3326	2100
Val. `q16` new [b]	5495	4172	3280	2319	1391	953
%	**50.48**	**51.14**	**52.72**	**56.98**	**58.18**	**54.62**

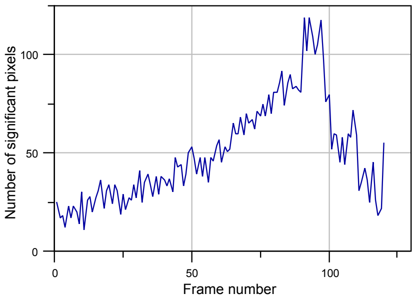

Table 14: Compression results and significant pixel distribution at 6000 pixels.

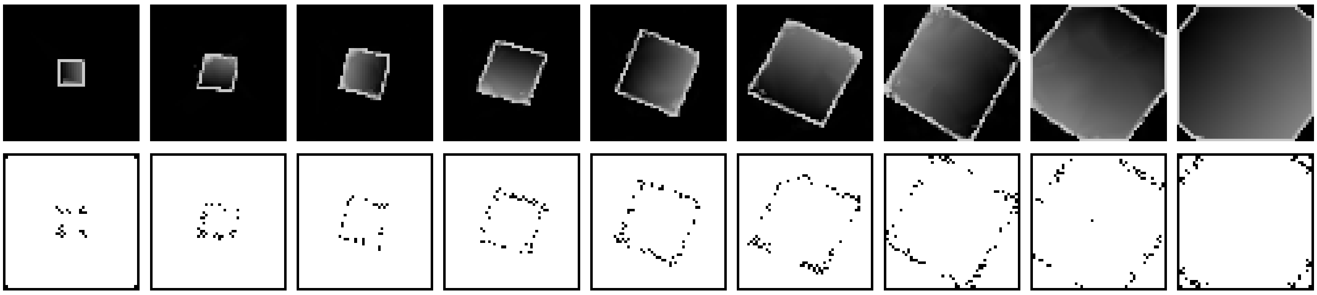

Table 15: 6000 sign. pixels, `q4`, PSNR 26.33 dB. Top: Reconstructed frames 0, 14, 29, 44, 59, 74, 89, 104, 119. Bottom: Significant pixels.

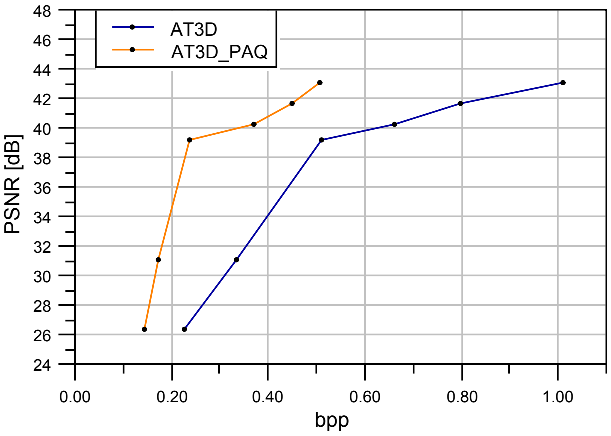

Figure 25: Comparison of bits per pixel vs. PSNR of old and new implementation at `q4`.

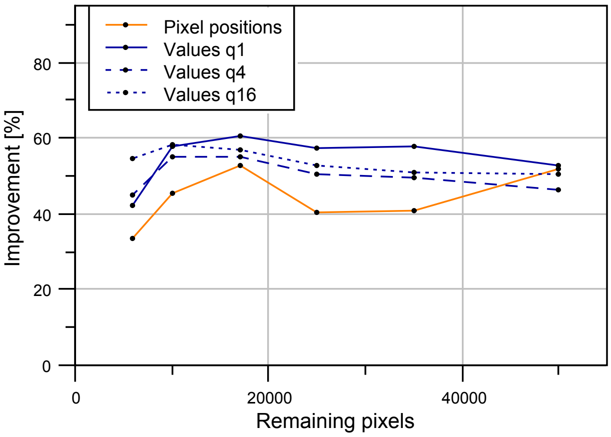

Figure 26: Improvement of AT3D_PAQ vs. AT3D on `RotGrowBox-120_50x50`.

`Tennis-20_sif`						
Sign. pixels	399600	324600	249599	149598	99597	64597
Px. pos. old [b]	161467	143542	121260	84694	62987	45956
Px. pos. new [b]	131057	112879	90659	57948	41580	31166
%	**18.83**	**21.36**	**25.24**	**31.58**	**33.99**	**32.18**
Val. `q1` old [b]	356282	296269	234612	147832	101153	66624
Val. `q1` new [b]	313917	261469	207625	130551	87996	56730
%	**11.89**	**11.75**	**11.50**	**11.69**	**13.01**	**14.85**
Val. `q4` old [b]	256416	215083	172106	110287	76105	50327
Val. `q4` new [b]	220163	185837	149890	96552	65637	42550
%	**14.14**	**13.60**	**12.91**	**12.45**	**13.75**	**15.45**
Val. `q16` old [b]	160419	136520	111210	73344	51430	34321
Val. `q16` new [b]	121538	105058	87191	58596	40492	26264
%	**24.24**	**23.05**	**21.60**	**20.11**	**21.27**	**23.48**

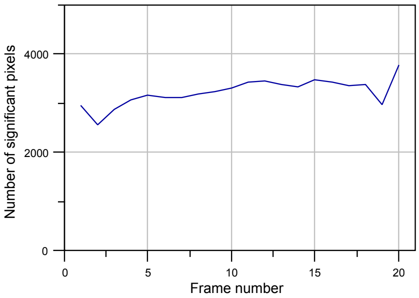

Table 16: Compression results and sign. pixel distribution at 64597 pixels.

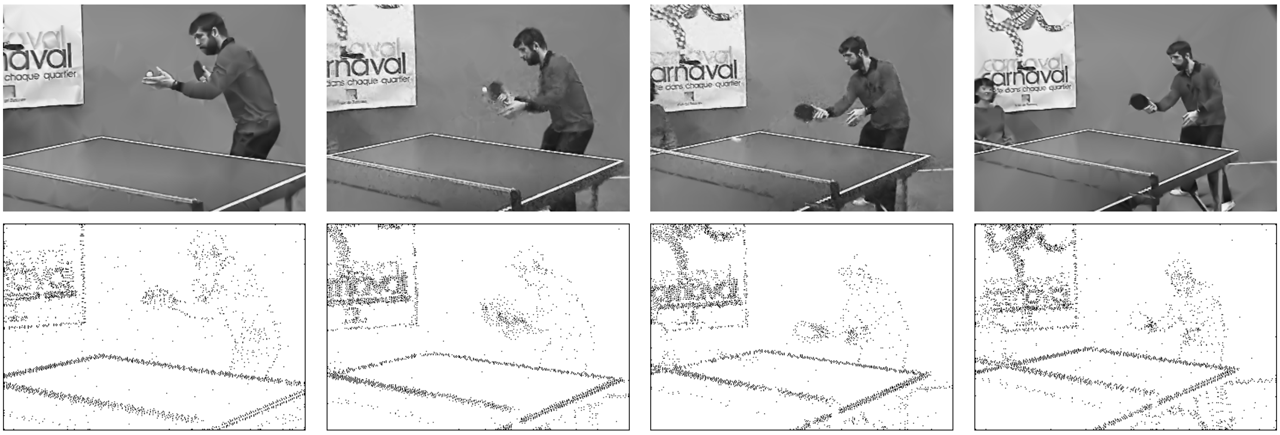

Table 17: 64597 sign. pixels, `q4`, PSNR 28.33 dB. Top: Reconstructed frames 0, 7, 13, 19. Bottom: Sign. pixels.

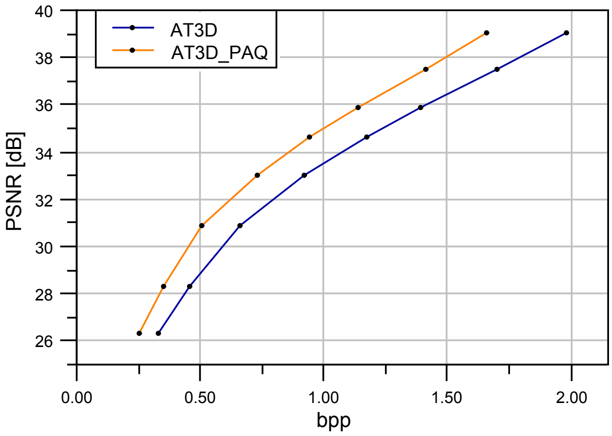

Figure 27: Comparison of bits per pixel vs. PSNR of old and new implementation at `q4`.

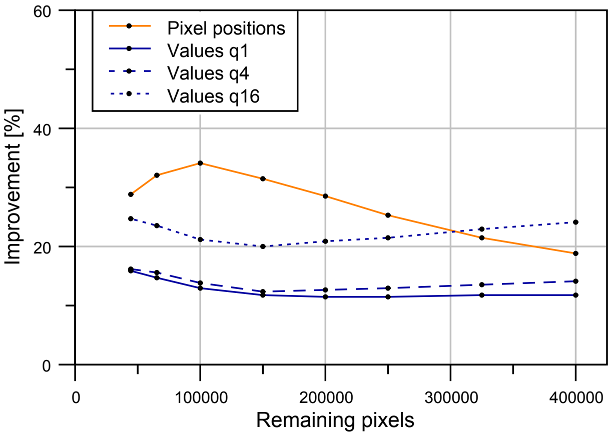

Figure 28: Improvement of AT3D_PAQ vs. AT3D on `Tennis-20_sif`.

3.4.4 Summary and analysis of overall results

The examples presented above represent a variety of result classes. These are not necessarily related to visible properties of a sequence. A sequence showing results similar to `Suzie` and `MissAmerica` is `SSTB1450`. They share the same distribution of significant pixels among frames with apparently random distributions in intermediate frames, which explains the decaying improvements, as more pixels are removed. Yet grayscale value encoding benefited more, which may be related to the recurring small contrasts within the wave-structures moving in the background. The PSNR-value difference is only significant when using high-quality settings, reaching more than 1.5 dB from 38 dB upwards.
A sequence with consistently high improvements is `Tennis-20_sif`. Significant pixels are clustered closely at all tested quality settings and the number of significant pixels is almost constant throughout the frames. Due to the higher `sif`-resolution the clusters remain large at low quality settings, resulting in a better improvement. Additionally due to camera zoom and player motion, the projected context boxes of AT3D are not a suitable model. Combined with the constant grayscale value encoding improvement, a constant gain of 1.6-2 dB is achieved.
A sequence with similar properties for the first 23 frames is `Highway2`. Yet in the last frames the number of significant pixels drops due to the disappearing bridge at the top of the sequence. Their distribution appears more random in the cloud-areas, explaining the slightly worse improvement values in pixel position encoding. The PSNR-value difference is between 0.8 dB for low quality output and up to 1.8 dB for higher quality output.
All artificially generated sequences containing simple affine transformations applied to basic geometric objects showed a huge improvement. One representative of this result class is `RotGrowBox-120_100x100`. Pixel position encoding improvements can be explained with the same reasons as for `Tennis-20_sif`. The significant pixel distribution results in either very large or very small contrasts between neighboring pixels, which may be a cause for the vast improvement in grayscale value encoding. Other artificial sequences show similar significant pixel distributions and significantly smaller grayscale value encoding improvements, while not containing as many very large or very small contrasts. Again we have to note, that grayscale value encoding in particular requires a more detailed analysis, which we suggest for future work.

3.4.5 Compression time comparison: AT3D_PAQ vs. AT3D

The starting point for our development, PAQ8kx, is a slow general-purpose compressing scheme. Due to the smaller number of used models the lossless compression stage of AT3D_PAQ is faster than the original PAQ implementation, but still slower than the one from AT3D. Yet, the increased amount of time is negligible compared to the time needed for adaptive thinning. By removing the DMC- and chart-model from the grayscale value encoding, AT3D_PAQ may be accelerated at the cost of decreasing its compression performance. In table 18 we listed some compression time evaluations.

Sequence	Sign. pixels [M]	Time [s]	
		AT3D	AT3D_PAQ
`Tempete-20_cif`	0.458	193.8	212.6
	0.208	43.6	72.5
	0.038	3.5	14.4
`Soccer`	0.760	80.4	187.2
	0.230	46.6	89.6
	0.090	8.9	25.1
	0.025	1.5	7.8

Table 18: Comparison of compression times for two sequences at different thinning stages with quantization setting `q1`.

3.5 Numerical results: AT3D_PAQ vs. H.264

In this section we will compare the performances of AT3D_PAQ and H.264 on a variety of test sequences. After describing the test environment, we will present detailed results of several test sequences, then continue with some additional results of more sequences and finally analyze and summarize these.

Remark 3.5.1 (H.264 settings)**.** As noted in appendix A, we used the newest version of FFmpeg with its included free H.264-encoder `libx264`. It provides numerous settings, which influence the encoding performance. In this comparison we will therefore use two sets of parameters for H.264:

1.
```
ffmpeg.exe -s %size% -pix_fmt gray -i %inputFilename%.yuv -s %size% -vcodec
libx264 -b:v %bit-rate% -preset medium -psnr %outputFile%.h264
```

2.
```
ffmpeg.exe -s %size% -pix_fmt gray -i %inputFilename%.yuv -s %size% -vcodec
libx264 -b:v %bit-rate% -preset placebo -tune psnr -psnr %outputFile%.h264
```

The first setting describes the standard settings of H.264. It optimizes the output for visual quality, which in general reduces the PSNR. Thus when comparing the PSNR-value of a sequence, FFmpeg suggests using the `-tune psnr`-switch, which even optimizes the output towards a better PSNR-value. In order to get the best possible output, we additionally set the preset to `placebo`, which increases time needed for compression. In summary, we have a regular H.264-setting and a best-case-setting. Our primary goal is the comparison of AT3D_PAQ and the regular H.264 setting.

On the following pages, we will present detailed comparisons showing the filesizes of compressed output of AT3D, AT3D_PAQ, H.264 Regular, and H.264 Best and the improvement or deterioration of AT3D_PAQ compared to both H.264 settings. Additionally we will provide the original and reconstructed frames of AT3D_PAQ and H.264 Regular and rate-distortion diagrams comparing all four schemes.
All AT3D-results were obtained by enabling the pixel exchange with radius $r = 3$. H.264 output at a fixed PSNR-value was generated by manually adjusting the bit-rate parameter, until a suitable PSNR-value was achieved.

`Suzie-10`						
PSNR [dB]	Filesize [bytes]					
	AT3D	AT3D_PAQ	H.264 Reg.	%	H.264 Best	%
42.2	20657	18717	9590	-95.17	**7949**	-135.46
39.6	10877	10083	5966	-69.01	**4795**	-110.28
36.8	5089	4775	3524	-35.50	**3010**	-58.64
35.2	3079	2915	2653	-9.88	**2321**	-25.59
33.8	1930	**1824**	2150	15.16	1889	3.44
31.4	1002	**947**	1591	40.48	1402	32.45

Table 19: Compression results, %-columns show improvement/deterioration of AT3D_PAQ over H.264 results. Best result per row is marked by boldface letters.

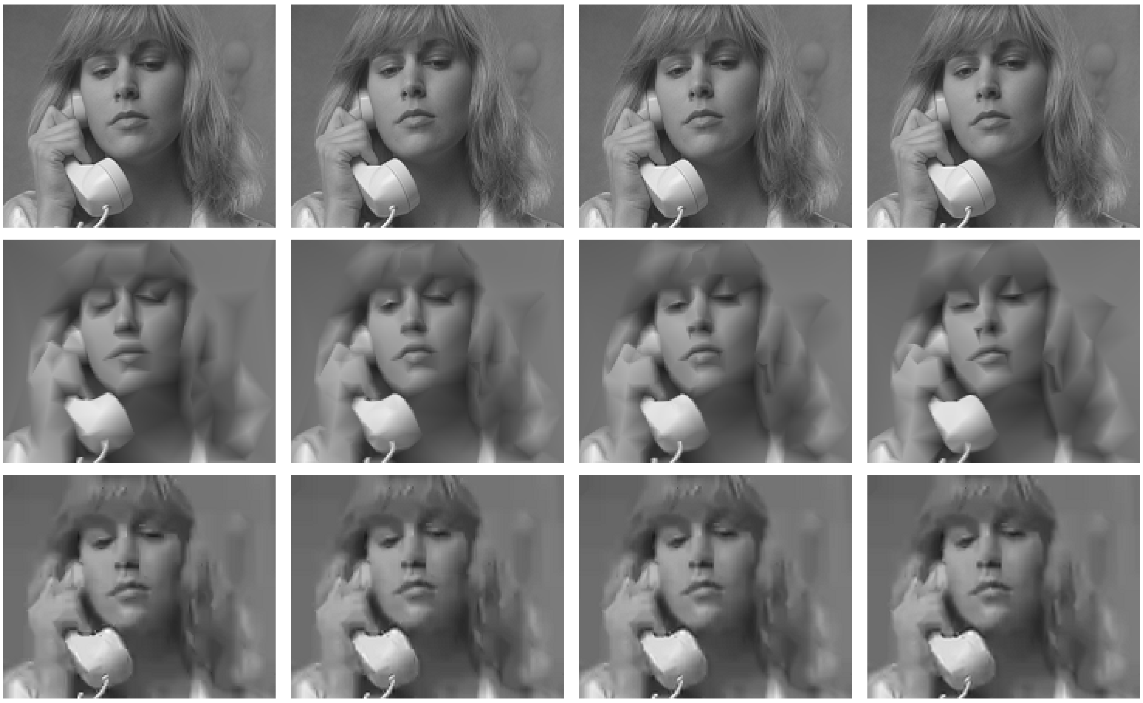

Table 20: PSNR 31.4 dB, 540 sign. pixels. Top: Original frame (0,3,6,9), middle: reconstructed AT3D_PAQ frame, bottom: reconstructed H.264 Regular frame.

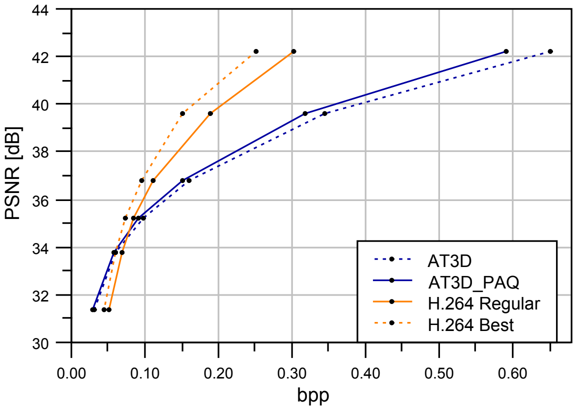

Figure 29: Rate-distortion diagram of AT3D-implementations at `q4` vs. both H.264-settings.

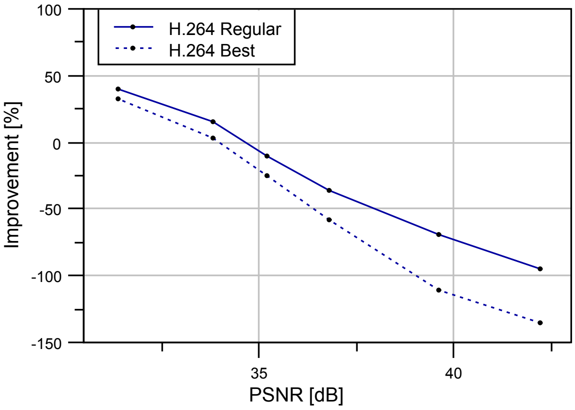

Figure 30: Improvement/deterioration of AT3D_PAQ vs. H.264 Regular and H.264 Best.

`City-10`						
PSNR [dB]	Filesize [bytes]					
	AT3D	AT3D_PAQ	H.264 Reg.	%	H.264 Best	%
42.6	83025	75438	28146	-168.02	**25335**	-197.76
38.3	55086	51070	14601	-249.77	**12694**	-302.32
35.5	38016	35774	9154	-290.80	**8286**	-331.74
32.9	23424	22238	6190	-259.26	**5450**	-308.04
30.8	14490	13862	4301	-222.30	**3892**	-256.17
29.5	9570	9214	3487	-164.24	**3120**	-195.32

Table 21: Compression results, %-columns show improvement/deterioration of AT3D_PAQ over H.264 results. Best result per row is marked by boldface letters.

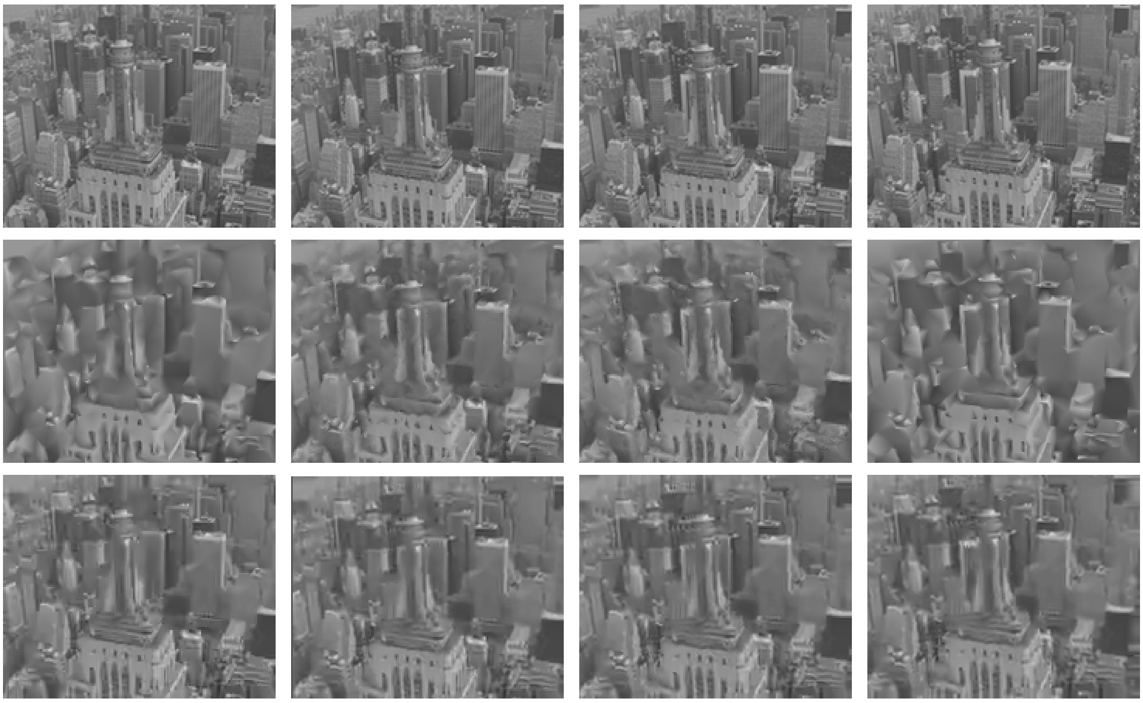

Table 22: PSNR 29.5 dB, 6440 sign. pixels. Top: Original frame (0,3,6,9), middle: reconstructed AT3D_PAQ frame, bottom: reconstructed H.264 Regular frame.

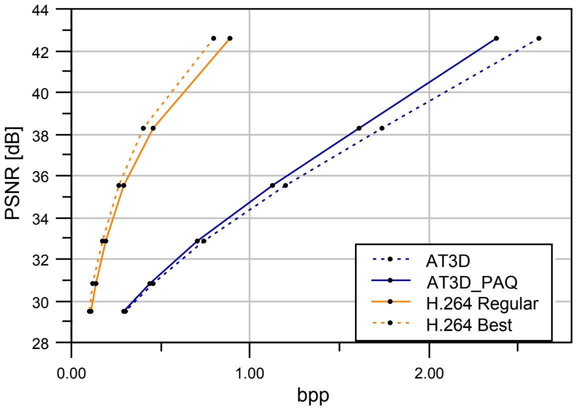

Figure 31: Rate-distortion diagram of AT3D-implementations at `q4` vs. both H.264-settings.

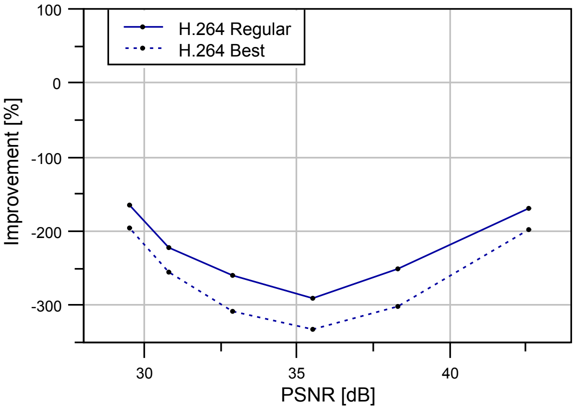

Figure 32: Improvement/deterioration of AT3D_PAQ vs. H.264 Regular and H.264 Best.

SSTB6355						
PSNR [dB]	Filesize [bytes]					
	AT3D	AT3D_PAQ	H.264 Reg.	%	H.264 Best	%
41.2	110803	90837	84042	-8.09	**67384**	-34.80
39.1	83147	69656	66444	-4.83	**53254**	-30.80
36.4	57785	49217	47125	-4.44	**36984**	-33.08
34.1	41033	35411	33469	-5.80	**25229**	-40.36
32.1	29233	25145	23281	-8.01	**16894**	-48.84
30.0	19057	16506	14463	-14.13	**9919**	-66.41

Table 23: Compression results, %-columns show improvement/deterioration of AT3D_PAQ over H.264 results. Best result per row is marked by boldface letters.

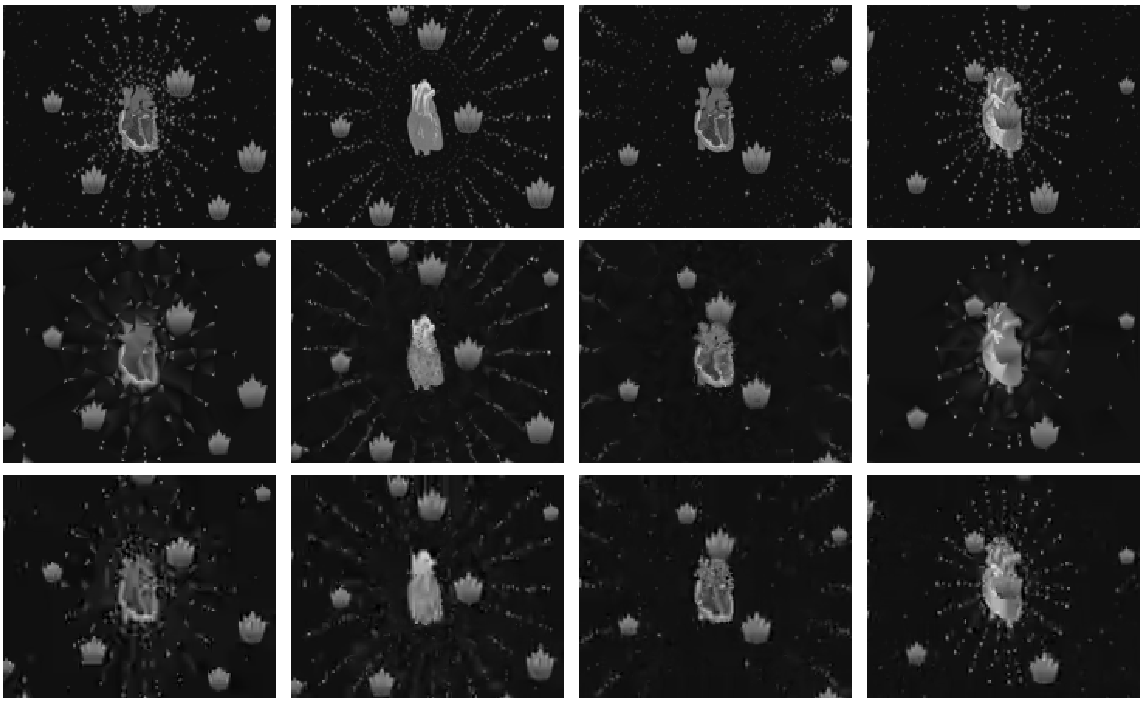

Table 24: PSNR 30 dB, 12320 sign. pixels. Top: Original frame (0,9,19,29), middle: reconstructed AT3D_PAQ frame, bottom: reconstructed H.264 Regular frame.

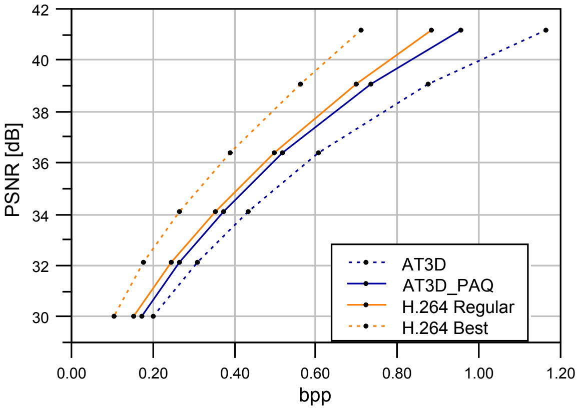

Figure 33: Rate-distortion diagram of AT3D-implementations at q4 vs. both H.264-settings.

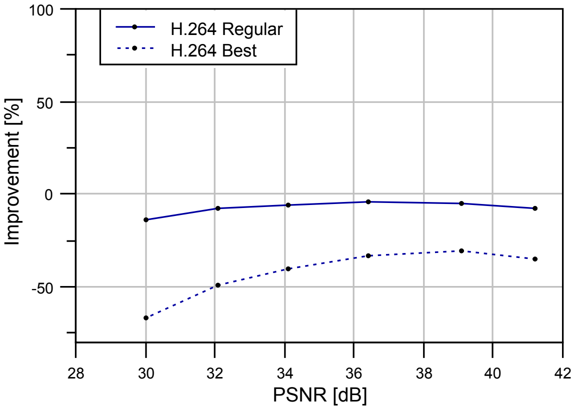

Figure 34: Improvement/deterioration of AT3D_PAQ vs. H.264 Regular and H.264 Best.

SSTB101980						
PSNR [db]	Filesize [bytes]					
	AT3D	AT3D_PAQ	H.264 Reg.	%	H.264 Best	%
42.9	14379	12270	10193	-20.38	**7189**	-70.68
40.4	8128	7324	7904	7.34	**5648**	-29.67
38.1	4657	**4375**	6264	30.16	4560	4.06
36.0	2871	**2760**	5056	45.41	3710	25.61
34.0	1765	**1698**	4202	59.59	3093	45.10
32.0	1093	**1040**	3435	69.72	2600	60.00

Table 25: Compression results, %-columns show improvement/deterioration of AT3D_PAQ over H.264 results. Best result per row is marked by boldface letters.

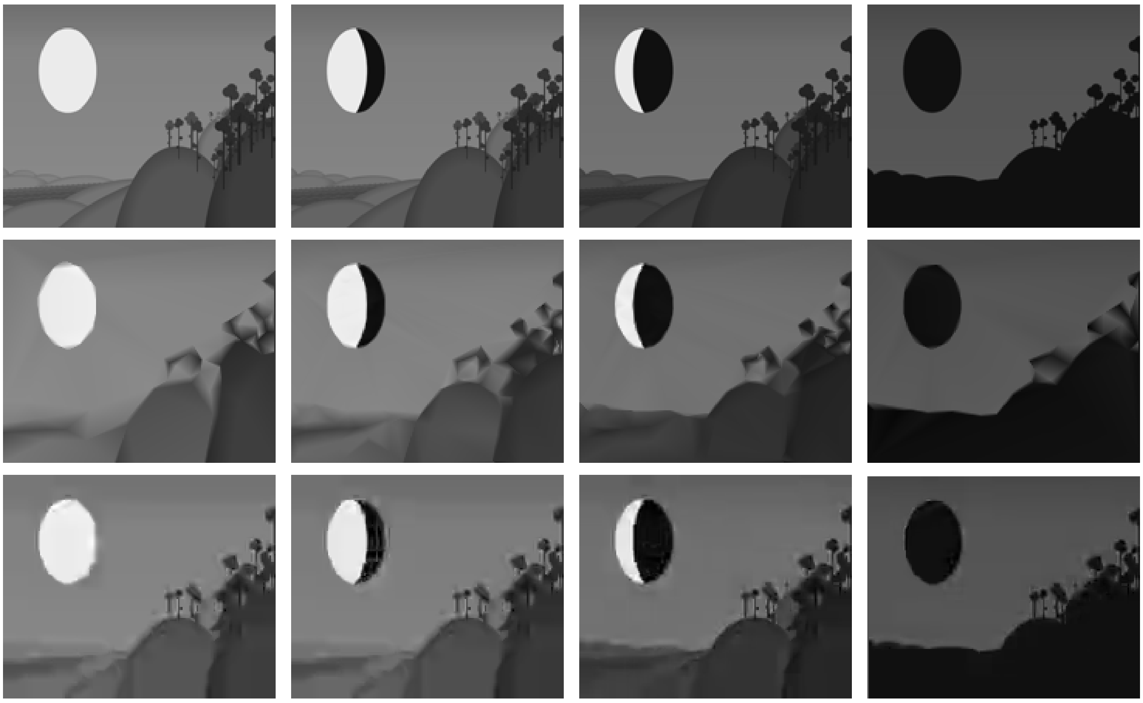

Table 26: PSNR 32 dB, 520 sign. pixels. Top: Original frame (0,9,19,29), middle: reconstructed AT3D_PAQ frame, bottom: reconstructed H.264 Regular frame.

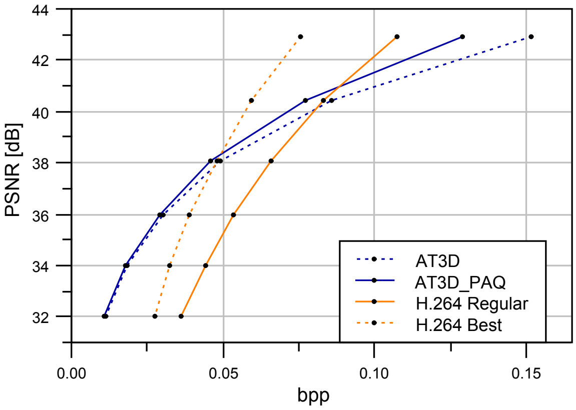

Figure 35: Rate-distortion diagram of AT3D-implementations at q4 vs. both H.264-settings.

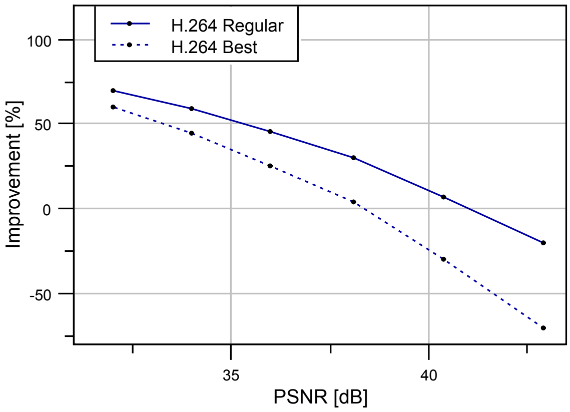

Figure 36: Improvement/deterioration of AT3D_PAQ vs. H.264 Regular and H.264 Best.

`RotGrowEllipse_100x100`						
PSNR [dB]	Filesize [bytes]					
	AT3D	AT3D_PAQ	H.264 Reg.	%	H.264 Best	%
48.4	13735	**8421**	10643	20.88	8480	0.70
44.2	7468	**4897**	8027	38.99	6442	23.98
38.1	3297	**2414**	5145	53.08	4003	39.70
33.3	1844	**1456**	3293	55.78	2800	48.00
30.2	1027	**858**	2570	66.61	2272	62.24
27.2	568	**493**	1997	75.31	1926	74.40

Table 27: Compression results, %-columns show improvement/deterioration of AT3D_PAQ over H.264 results. Best result per row is marked by boldface letters.

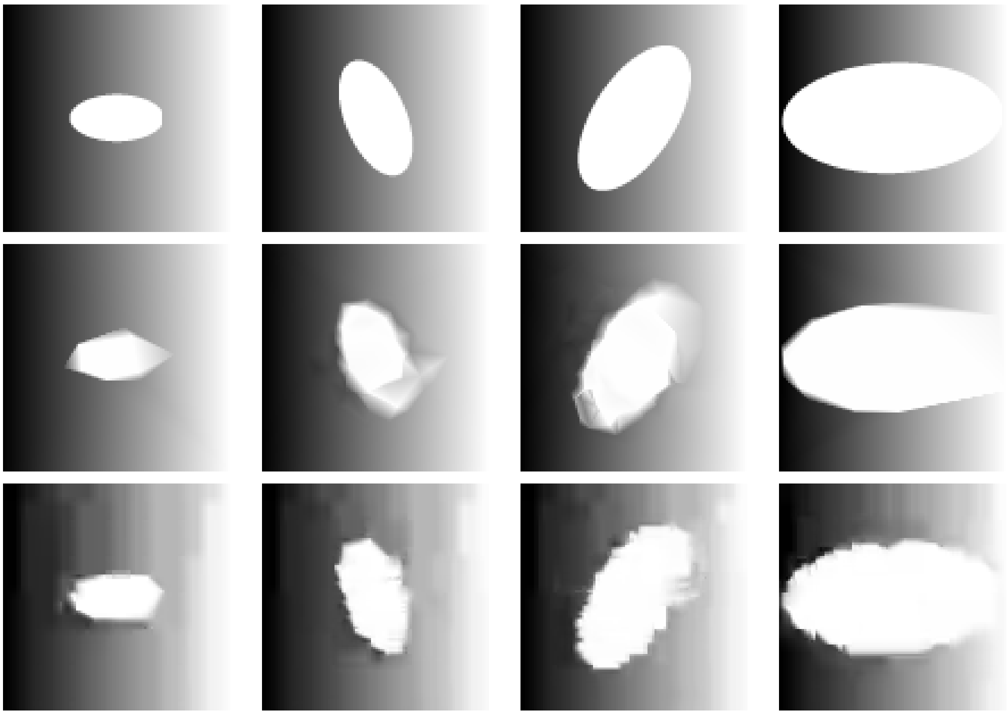

Table 28: PSNR 27.3 dB, 250 sign. pixels. Top: Original frame (0,9,19,29), middle: reconstructed AT3D_PAQ frame, bottom: reconstructed H.264 Regular frame.

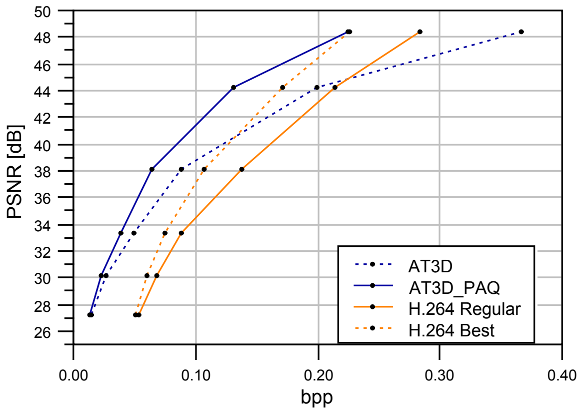

Figure 37: Rate-distortion diagram of AT3D-implementations at `q4` vs. both H.264-settings.

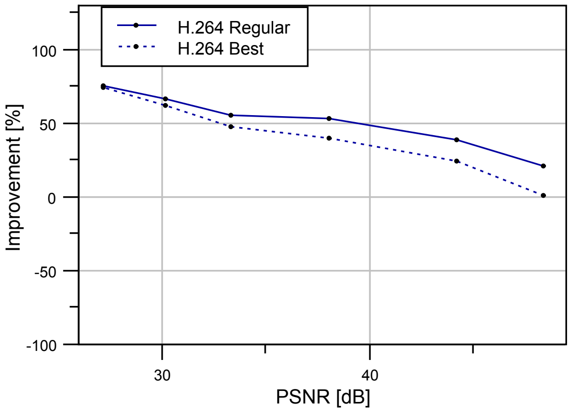

Figure 38: Improvement/deterioration of AT3D_PAQ vs. H.264 Regular and H.264 Best.

On this page we will present further comparison results between H.264 Regular and AT3D_PAQ, but without pixel exchange due to its computational complexity. These results are between 0.5 and 2 dB worse compared to the exchanged results.

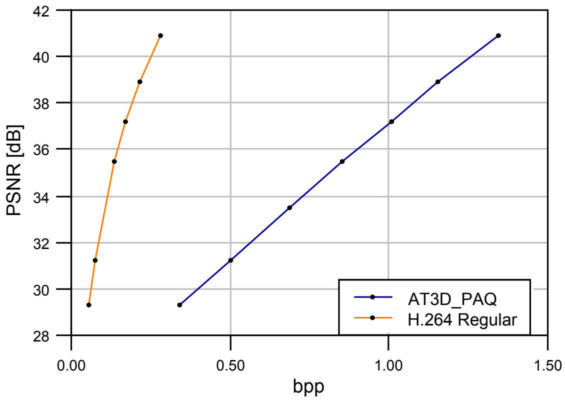

Figure 39: Rate-distortion diagram of AT3D_PAQ at `q4` and H.264 Regular on `WashDC`.

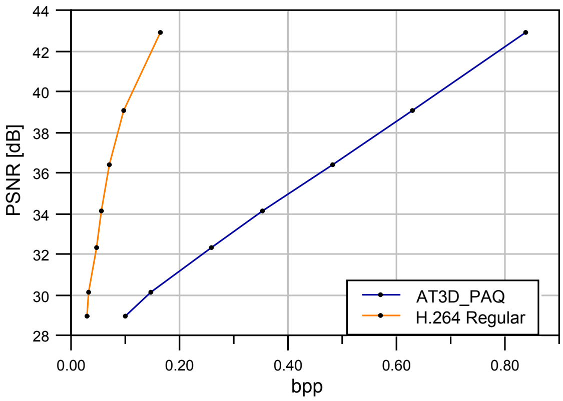

Figure 40: Rate-distortion diagram of AT3D_PAQ at `q4` and H.264 Regular on `SSTB88250`.

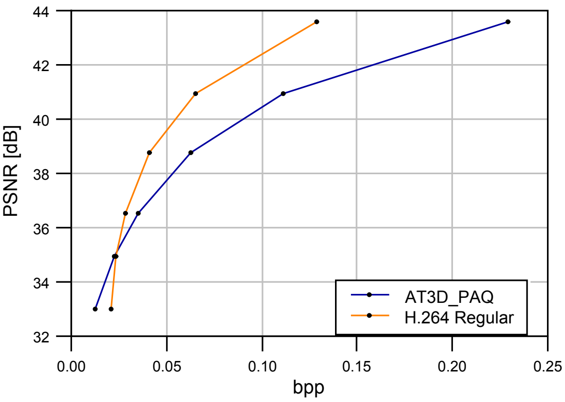

Figure 41: Rate-distortion diagram of AT3D_PAQ at `q4` and H.264 Regular on `MissAmerica`.

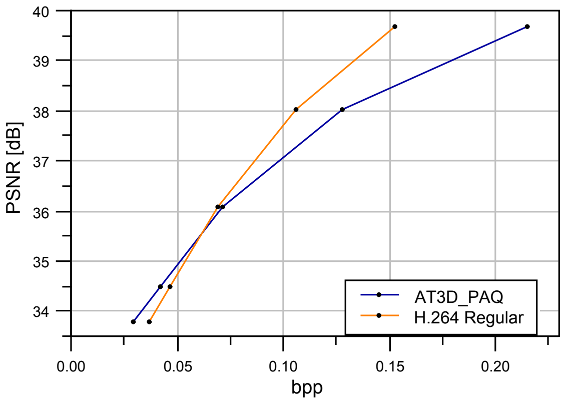

Figure 42: Rate-distortion diagram of AT3D_PAQ at `q4` and H.264 Regular on `SSTB111130`.

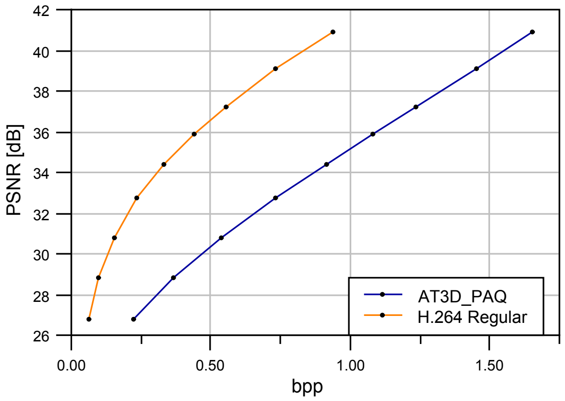

Figure 43: Rate-distortion diagram of AT3D_PAQ at `q4` and H.264 Regular on `Tempete-20_cif`.

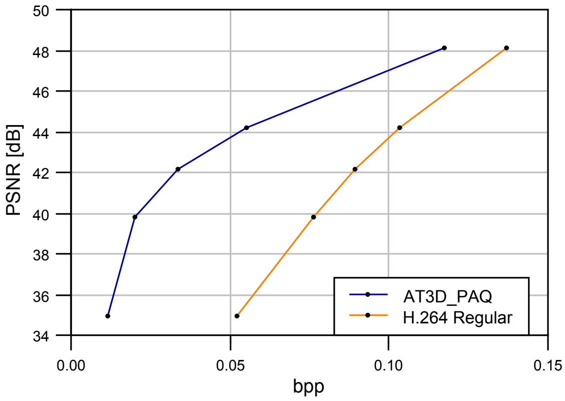

Figure 44: Rate-distortion diagram of AT3D_PAQ at `q4` and H.264 Regular on `TranslRotEll_100x100`.

3.5.1 Summary and analysis of results

The above comparison of AT3D_PAQ and H.264 showed, that the results are highly dependent on the type of the tested sequence. H.264 shows significantly better performance in about 73% of the considered cases, in particular for high-entropy sequences, which contain motion. The biggest difference is seen in sequences, which contain little motion and show objects with fine texture. Here H.264 with regular settings outperforms AT3D_PAQ by as much as 300%. However, for low-entropy sequences, which allow reasonable PSNR-values with a small number of significant pixels (below 0.002% of the original number of pixels), AT3D_PAQ shows improvements over H.264 Regular and Best by as much as 40%. For higher quality output, H.264 is still superior.

We were able to identify one class of test sequences, where AT3D_PAQ outperforms both H.264 settings by far: artificially generated test sequences showing low-complexity scenes with affine transformations applied to simple geometric objects. Due to the large improvements of AT3D_PAQ compared to AT3D for these sequence types, the former is now competitive to even H.264 Best for all quality ranges. Note, that the geometric test sequences were chosen in a resolution of 100x100 pixels, which leads to decreased H.264 performance due to its block sizes. This shows another advantage of AT3D(_PAQ), whose performance is independent of the video resolution. A brief investigation for `RotEllipse_qcif` yielded, that even at `qcif`-resolution AT3D_PAQ yields better compression performance than the best H.264 quality settings, but results are closer for lower quality output.

3.6 Conclusion

Our numerical investigations have shown, that AT3D_PAQ yields significant improvements for all tested sequences, which included a wide variety of different standard sequences, as well as cartoon-sequences from the movie *Sita sings the blues*, and artificially generated geometric sequences. Scenes containing fine texture or large amounts of motion are now compressed significantly better due to more efficient pixel position encoding, while all sequences benefited from improved grayscale sequence coding. Most notably the geometric sequences stand out with consistently high improvements.

The comparison of H.264 and AT3D_PAQ showed, that both schemes have very different compression characteristics. H.264 produces dramatically smaller output for high-entropy sequences and scenes containing textured elements, while for low-entropy sequences it is only superior at high-quality settings. For geometric sequences and low-quality output of low-entropy sequences AT3D_PAQ beats H.264 by as much as 75%. Geometric sequences in particular benefited from the new lossless encoding stage in AT3D_PAQ and contributed significantly to the superiority over H.264.

3.7 Outlook

Despite all improvements achieved as a result of this thesis, there are many starting points for further investigations.

- Pixel position encoding is unlikely to be improvable significantly. However, a more sophisticated motion estimation algorithm than the one tested may yield small improvements for sequences containing translational motion of significant pixels.
- An in-depth analysis of grayscale value encoding with AT3D_PAQ is necessary. Results indicate, that the introduced models work very well for certain sequences, while improving other sequences less. After understanding the involved structural characteristics further compression improvements may be possible.
- The architecture of AT3D_PAQ now allows for additional models, which do not deteriorate the existing compression performance, but only enhance it. Thus the implementation of additional models is likely to further improve the compression performance.
- It may be possible to modify adaptive thinning, such that it produces lower entropy output by e.g. not removing the more predictable of two pixels, if their significance is (almost) equal. If one pixel is located within a group of significant pixels, and the other is not, the latter should be removed.
- When comparing AT3D_PAQ and H.264, there are many starting points for optimization of the former. The adaptive thinning algorithm has to be further optimized for motion and high-entropy input to be competitive.
- AT3D_PAQ introduces moving artifacts in parts of sequences, which do not contain motion. It may be possible to remove these by further modifying adaptive thinning.
- It may be possible to improve the compression performance, if the uniqueness of the Delaunay tetrahedralization is not required during the encoding process, and instead in the end an index resulting from a canonical numbering of possible tetrahedralizations of a cell is saved. Alternatively it would be possible to only enforce the uniqueness in the last encoding step and still benefit from improved compression during the encoding process.

4 Runtime optimization

Independent of all previously presented results we will now begin investigating a different aspect of the improvement of AT3D: the runtime of adaptive thinning.
It was shown in [Dem11], that the theoretical complexity of AT3D and the pixel exchange is $\mathcal{O}(N \log(N))$, where N is the number of removed pixels. Unfortunately, the implicitly contained constant is very large, resulting in slow runtimes. There are several reasons for the computational complexity of the algorithm in each removal iteration:

- Large amount of memory allocations and deallocations,
- calculation of cell-tetrahedralizations,
- evaluation of linear splines at thinned pixel positions.

The second central result of this thesis is a reorganization of some of the code, which reduced runtimes considerably without changing the used core algorithms, adaptive thinning and pixel exchange, themselves. In this section the changes applied to the old implementation will be explained in detail, a theoretical investigation will yield the inefficiency of the AT3D implementation when evaluating linear splines, and numerical evidence from the computation of a set of examples will be presented. We will also analyze the impact of those changes in different stages of the compression scheme: Some speed up the tetrahedralization-heavy beginning of the adaptive thinning procedure, others improve the efficiency of the assignment of thinned pixels to tetrahedra, which speeds up later stages of adaptive thinning when many pixels are thinned already. We will provide numerical evidence, that the last thinning stage benefits from the improvements remarkably more than the others. Since the AT3D code base may be compiled for several platforms and processors, we additionally provide some runtime results with different processor-specific optimization levels and compare the performance of compilations in 32- and 64-bit.
The next section contains the description of the new implementation and some theoretical evidence for its advantages. We will then select two test environments based on brief numerical investigations, which constitute a worst-case and a best-case setting for runtime improvements. Afterwards the numerical results will be presented, starting with summarized tables and illustrating graphs. The individual results are listed in appendix E.

4.1 Improvements of the AT3D implementation

In this section we will list all implemented changes, that had an impact on computational time needed for compression. Changes with greater impact will be listed first.

4.1.1 Assignment of pixels to tetrahedra

One of the core functionalities needed in the adaptive thinning algorithm is the calculation of the significance of a vertex of the tetrahedralization that has not been thinned yet. In order to do this, the vertex is removed from its cell, the cell is re-tetrahedralized and the interpolation error arising from this procedure is calculated. In order to evaluate the grayscale value at a thinned pixel it is necessary to evaluate the correct piece of the trivariate linear spline interpolant, which requires the assignment of each thinned pixel to a tetrahedron of the re-tetrahedralization of the cell.

In this section we will describe the new implementation of assigning a query point $q \in \mathbb{R}^3$ to a tetrahedron $T \in \mathcal{C}_p$ or to decide that $q \notin \mathcal{C}_p$, where $p \in Y$ is a significant pixel and $\mathcal{C}_p$ is its cell. In particular all lemmas, theorems, and proofs in this section are a result of this thesis, unless denoted otherwise.
We start with establishing some geometric properties of planes and tetrahedra:

Definition 4.1.1 (Orientation)**.** Let $p_0, p_1, p_2 \in \mathbb{R}^3$. Then the tuple (p_0, p_1, p_2) is called ***right-hand oriented***, iff their box product satisfies $\langle (p_0 \times p_1), p_2 \rangle > 0$. It is called ***left-hand oriented***, iff $\langle (p_0 \times p_1), p_2 \rangle < 0$. It is called ***coplanar***, if $\langle (p_0 \times p_1), p_2 \rangle = 0$.

Remark 4.1.2. Note that the box product is not commutative. If the order of the tuple (p_0, p_1, p_2) is permuted once and only once, the orientation of the tuple is switched.

Let us first sum up how the query point assignment procedure was realized in the original implementation. The following definition of a bounding box is a generalization of what is used in the original implementation of AT3D.

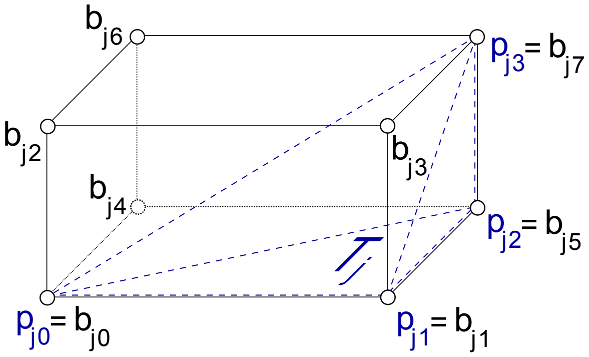

Figure 45: Minimal bounding box $\mathfrak{B}_{T_j}$ of a single tetrahedron T_j.

Definition 4.1.3. Let $\mathbb{T} := \bigcup_{j\in\mathbb{N}} T_j$ be a finite set of regular tetrahedra T_j, which are spanned by the vertices $p_{ji} \in \mathbb{R}^3$. We can then define $\mathfrak{B}_{\mathbb{T}}$, the ***minimal bounding box containing*** $\mathbb{T}$, as follows:

$$\mathfrak{B}_{\mathbb{T}} := \operatorname{conv}(\{b_i \mid i \in \underline{7}\}),$$

with

$$\begin{aligned}
b_0 &= \left(\min_{i,j}(p_{ji})_x, \min_{i,j}(p_{ji})_y, \min_{i,j}(p_{ji})_z\right), & b_1 &= \left(\max_{i,j}(p_{ji})_x, \min_{i,j}(p_{ji})_y, \min_{i,j}(p_{ji})_z\right)\\
b_2 &= \left(\min_{i,j}(p_{ji})_x, \max_{i,j}(p_{ji})_y, \min_{i,j}(p_{ji})_z\right), & b_3 &= \left(\max_{i,j}(p_{ji})_x, \max_{i,j}(p_{ji})_y, \min_{i,j}(p_{ji})_z\right)\\
b_4 &= \left(\min_{i,j}(p_{ji})_x, \min_{i,j}(p_{ji})_y, \max_{i,j}(p_{ji})_z\right), & b_5 &= \left(\max_{i,j}(p_{ji})_x, \min_{i,j}(p_{ji})_y, \max_{i,j}(p_{ji})_z\right)\\
b_6 &= \left(\min_{i,j}(p_{ji})_x, \max_{i,j}(p_{ji})_y, \max_{i,j}(p_{ji})_z\right), & b_7 &= \left(\max_{i,j}(p_{ji})_x, \max_{i,j}(p_{ji})_y, \max_{i,j}(p_{ji})_z\right).
\end{aligned}$$

A visualization for a single tetrahedron is shown in figure 45, and for a set of tetrahedra in figure 47. For convenience, we will denote the bounding box by $\mathfrak{B}_T$, if the set only contains one tetrahedron.

Note that in the implementation of AT3D only bounding boxes for single tetrahedra were considered. Before we can summarize the old evaluation implementation, we need the following definition:

Definition 4.1.4. In thinning step m the set of all thinned pixels in the minimal bounding box containing T_j, denoted by Q_{T_j}, is defined as

$$Q_{T_j} := \left\{q_i \in ((X \backslash X_{N-m}) \cap \mathfrak{B}_{T_j})\right\}.$$

In the implementation of AT3D an ordered sequence $\widetilde{Q}_{T_j}$ is calculated instead of the set Q_{T_j}. This sequence is ordered with respect to the indices $I(q)$ for $q \in Q_{T_j}$. We may now summarize the old evaluation algorithm as follows:

Algorithm 12 AT3D implementation of query point evaluation

```
Let $p \in Y \subset X$ be a significant pixel with cell $\mathcal{C}_p$
for each tetrahedron $T_j \in \mathcal{C}_p$ do
    Calculate $\widetilde{Q}_{T_j}$
    for each $q \in \widetilde{Q}_{T_j}$ do
        if $q$ is not marked and $q \in T_j$ then
            Evaluate linear spline interpolant at $q$
            Mark $q$
        end if
    end for
end for
```

In this algorithm, checking if $q \in T_j$ is the most complex part. The above implementation made a large number of these geometric tests necessary, and our goal is a significant reduction of these. This approach suffers from several problems, which we will now investigate. For example the order, in which the tetrahedra were checked was fixed and was not adapted to geometric information gained in the assignment process. To be able to give some theoretical evidence we need the following lemmas:

Lemma 4.1.5. *Let $d_0, d_1, d_2 \in \mathbb{R}^2$, such that they span a regular triangle D and the minimum bounding rectangle containing D, called R. The area of D is denoted by A_D, while A_R denotes the area of R. Then*

$$\forall A_R > 0 : \forall \varepsilon > 0 : \exists d_0, d_1, d_2 \in \mathbb{R}^2 : \frac{A_D}{A_R} < \varepsilon.$$

In other words: The area of the triangle D can be arbitrarily small compared to the area of its minimum bounding rectangle R, if R is any fixed regular rectangle.

Proof:
Let $\varepsilon > 0$. Let $A_R > 0$ and R be spanned by two edges with nonzero lengths a and b, such that $A_R = ab$. Let the vertices of R have positive components w.l.o.g. with one vertex being $(0,0)$. Choose $d_0 := (0,0), d_1 := (a,b)$. Now the minimum bounding rectangle of the triangle D, that is spanned by d_0, d_1, d_2, is the rectangle R, iff $d_2 \in R$:

$$R := \text{conv}\,\{d_0, c_0, c_1, d_1\} \quad \text{and } c_0 = (0,b) \in \mathbb{R}^2, c_1 = (a,0) \in \mathbb{R}^2.$$

Hence d_2 can be chosen such that it minimizes the area of D, as visualized in figure 46. Let

$$m := \frac{1}{2}(d_1 - d_0) = \left(\frac{a}{2}, \frac{b}{2}\right) \in \mathbb{R}^2,$$
$$d_2 := \left(\frac{a}{2} - \frac{ab^2\varepsilon}{a^2+b^2}, \frac{b}{2} + \frac{a^2b\varepsilon}{a^2+b^2}\right).$$

In this definition d_2 is not necessarily located inside R. W.l.o.g. we can continue our proof, as shown in the end. Then, due to the orthogonality of $\overline{d_0d_1}$ and $\overline{md_2}$, which can be verified easily, we have

$$A_D = \frac{1}{2}\overline{d_0d_1}\;\overline{md_2} = \frac{1}{2}\sqrt{a^2+b^2}\sqrt{\frac{a^2b^4\varepsilon^2 + a^4b^2\varepsilon^2}{(a^2+b^2)^2}} = \frac{1}{2}\sqrt{a^2+b^2}\sqrt{\frac{a^2+b^2}{(a^2+b^2)^2}}ab\varepsilon = \frac{1}{2}ab\varepsilon.$$

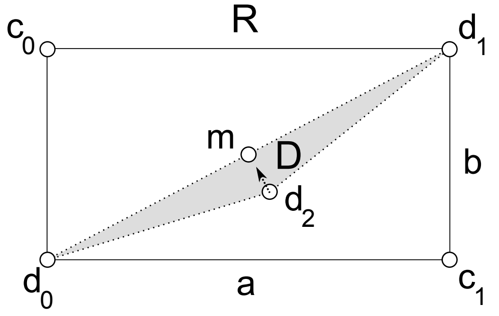

Figure 46: Visualization of proof idea of lemma 4.1.5.

With $A_R = ab$ we get

$$\frac{A_D}{A_R} = \frac{1}{2}\frac{ab\varepsilon}{ab} < \varepsilon.$$

Note that d_2 is not necessarily inside R. In this case R is not the bounding rectangle of D, so if it is outside, the vector

$$t := \left(-\frac{ab^2\varepsilon}{a^2+b^2}, \frac{a^2b\varepsilon}{a^2+b^2} \right)$$

has to be compressed by a factor $\delta < 1$, such that $\widetilde{d_2} := m + \delta t$ is inside R and D is spanned by $d_0, d_1, \widetilde{d_2}$. Hence the height of D is smaller than the height used in the above calculation, so the area of D must be even smaller than the calculated area, while A_R remains unchanged. So the assertion must be true as well.

□

Now we can consider the three-dimensional case, which is relevant for AT3D:

Lemma 4.1.6. *Let $p_0, p_1, p_2, p_3 \in \mathbb{R}^3$, such that they span a regular tetrahedron T and the minimum bounding box containing T, called $\mathfrak{B}_T$. Their respective volumes are denoted by V_T and $V_{\mathfrak{B}_T}$. Then*

$$\forall V_{\mathfrak{B}_T} > 0 : \forall \varepsilon > 0 : \exists p_0, p_1, p_2, p_3 \in \mathbb{R}^3 : \frac{V_T}{V_{\mathfrak{B}_T}} < \varepsilon.$$

In other words, the area of the tetrahedron T can be arbitrarily small compared to the area of the minimal bounding box $\mathfrak{B}_T$, if $\mathfrak{B}_T$ is fixed.

Proof:

First we note that from a geometric perspective, a tetrahedron T is a special cone. Therefore its volume is given by

$$V_T = \frac{1}{3}Ah,$$

where A denotes its base area and h its height, the Euclidean distance between base area and the fourth vertex of T. Our goal is to show for a given bounding box $\mathfrak{B}_T$ spanned by the vertices p_0, p_1, p_2, p_3 of T that there exists a distribution of these vertices that results in an arbitrarily small volume of T compared to $\mathfrak{B}_T$.

Let $\varepsilon > 0$ and $\mathfrak{B}_T$ be fixed and hence $V_{\mathfrak{B}_T} > 0$. We assume w.l.o.g. that p_0, p_1, p_2 span the triangular base area of T, which we call D. Again w.l.o.g. the plane spanned by those vertices is the plane E defined by $z = 0$ (if it is not it can be transformed into it by applying a suitable linear mapping). This plane contains w.l.o.g. one of the faces of $\mathfrak{B}_T$.

Let p_3 be located on the opposite face of $\mathfrak{B}_T$. If the orthogonal projection of p_3 into E is inside D, the minimal bounding box of T is defined by the minimum bounding rectangle R of the triangle D spanned by p_0, p_1, p_2, and by p_3, whose z-component we call $(p_3)_z$. Hence the total volume of the minimal bounding box, $V_{\mathfrak{B}_T}$, is given by

$$V_{\mathfrak{B}_T} = A_R (p_3)_z.$$

On the other hand, the volume of T is

$$V_T = \frac{1}{3} A_D (p_3)_z.$$

Thus

$$\frac{V_T}{V_{\mathfrak{B}_T}} = \frac{\frac{1}{3} A_D}{A_R}.$$

Since p_0, p_1, p_2 are located in a two-dimensional plane, we can apply lemma 4.1.5 with R given by the above construction and with $A_R = (V_{\mathfrak{B}_T}/(p_3)_z) > 0$ and obtain suitable p_0, p_1, p_2, such that

$$0 < \frac{A_D}{A_R} < \varepsilon.$$

Now let p_3 be located in the plane defined by $z = (p_3)_z$ and let the orthogonal projection of p_3 into E be inside D, which is possible, since $A_D > 0$. Then we get

$$\frac{V_T}{V_{\mathfrak{B}_T}} = \frac{\frac{1}{3} A_D}{A_R} < \frac{1}{3}\varepsilon < \varepsilon.$$

□

Corollary 4.1.7. *Let T be a given regular tetrahedron with minimal bounding box $\mathfrak{B}_T$. Then the ratio of corresponding volumes satisfies*

$$0 < \frac{V_T}{V_{\mathfrak{B}_T}},$$

and cannot be bounded away from 0.

This result yields that arbitrary few thinned pixels may be located inside a tetrahedron T. If this tetrahedron is always checked first, all performed geometry tests on it will fail. This is obviously not desirable. We can extend the above corollary to determine the largest possible ratio of volumes:

Lemma 4.1.8. *A given tetrahedron T with minimal bounding box $\mathfrak{B}_T$ satisfies*

$$\frac{V_T}{V_{\mathfrak{B}_T}} \leq \frac{1}{3}.$$

The largest possible ratio of the tetrahedral volume divided by its bounding box volume is $\frac{1}{3}$.

Proof:

Consider a fixed cuboid $\mathfrak{B}_T$ with one vertex being $(0,0,0)$, positive vertex coordinates and (positive) edge lengths $x_{max} =: x_m, y_{max} =: y_m, z_{max} =: z_m$. Thus

$$V_{\mathfrak{B}_T} = x_m \cdot y_m \cdot z_m.$$

We will first prove, that

$$\frac{V_T}{V_{\mathfrak{B}_T}} \leq \frac{1}{3}$$

holds under weaker conditions than posed by demanding $\mathfrak{B}_T$ to be the bounding box of T, and afterwards showing that the inequality is sharp.
To show the first part, let T be a tetrahedron with a vertex $a = (0, 0, 0)$ and with vertices b, c, d distributed such that $\mathfrak{B}_T$ is the minimal bounding box of T. The volume of T is given by

$$\begin{aligned} V_T &= \left| \frac{(b-a)\left((c-a)\times(d-a)\right)}{6} \right| \\ &= \frac{|b(c\times d)|}{6} \\ &= \frac{|b_x(c_y d_z - c_z d_y) + b_y(c_z d_x - c_x d_z) + b_z(c_x d_y - c_y d_x)|}{6} \end{aligned} \tag{4.1}$$

The volume of T is maximized, if $a, c, b, d \in \partial\mathfrak{B}_T$: If there is w.l.o.g. vertex $d \notin \partial\mathfrak{B}_T$, the volume of T is calculated by multiplying the base area of the tetrahedron, which we consider to be spanned by the remaining vertices of T, with the height of T, which is the Euclidean distance between the plane defined by the base area and d. By projecting d along the height onto $\partial\mathfrak{B}_T$, the base area remains unchanged, while the height increases, hence the volume increases.
Thus the components x, y, z of the vertices have to satisfy

$$x \in \{0, x_m\} \quad \wedge \quad y \in \{0, y_m\} \quad \wedge \quad z \in \{0, z_m\}\,. \tag{4.2}$$

We can maximize 4.1 by maximizing its numerator:

$$6V_T = \left| \underbrace{b_x(c_y d_z - c_z d_y)}_{:=V_1} + \underbrace{b_y(c_z d_x - c_x d_z)}_{:=V_2} + \underbrace{b_z(c_x d_y - c_y d_x)}_{:=V_3} \right| \overset{!}{=} \text{max}. \tag{4.3}$$

We only cover the maximization of a positive argument of the absolute value above, the converse is handled analogously. We can now show that under the constraints 4.2 the maximum value of 4.3 is $6V_T = 2x_m y_m z_m$. Note that these constraints are weaker than the constraints induced by the bounding box property of $\mathfrak{B}_T$, therefore possibly leading to an upper bound, which is larger than the one resulting from the true constraints.

1. We first maximize V_1, which is obviously achieved by choosing $b_x := x_m$, $c_y := y_m$, $d_z := z_m$ and $(c_z = 0 \vee d_y = 0)$, due to the positivity of the vertex components. Maximizing V_2 is then achieved by choosing $b_y := y_m$, $d_x := x_m$ and $c_z := z_m$, necessarily resulting in $d_y := 0$, and $c_x := 0$. Thus in V_3 we have $c_y d_x = x_m y_m$, so $V_1 + V_2 + V_3$ is maximized by choosing $b_z := 0$. Hence

   $$V_1 + V_2 + V_3 = 2x_m y_m z_m.$$

2. After maximizing V_1 as in the above case, we can maximize V_3 next: This is achieved by choosing $b_z := z_m$, $c_x := x_m$, $d_y := y_m$, resulting in $c_z := 0$ and $d_x := 0$. Then in V_2 we get $c_z d_x = 0$ and $c_x d_z = x_m z_m$, so the sum $V_1 + V_2 + V_3$ is maximized by choosing $b_y = 0$, resulting in

   $$V_1 + V_2 + V_3 = 2x_m y_m z_m.$$

3. We start by maximizing V_2 by choosing $b_y := y_m$, $c_z := z_m$, $d_x := x_m$ and $(c_x = 0 \vee d_z = 0)$. We continue with maximizing V_3, by choosing $b_z := z_m$, $c_x := x_m$, resulting in $d_z := 0$ and $d_y := y_m$. Then in V_1 we get $c_y d_z = 0$ and $c_z d_y = y_m z_m$, thus V_1 is maximized by choosing $b_x := 0$, resulting in
$$V_1 + V_2 + V_3 = 2 x_m y_m z_m.$$

Switching the maximization orders of the first two V_i in the above cases leads to identical results. Thus all cases, which maximize $V_1 + V_2 + V_3$ and hence 4.3, are handled.
Therefore we have
$$\begin{aligned}\frac{V_T}{V_{\mathfrak{B}_T}} &= \frac{|V_1 + V_2 + V_3|}{6} \frac{1}{x_m y_m z_m} \\ &\leq \frac{2 x_m y_m z_m}{6} \frac{1}{x_m y_m z_m} \\ &= \frac{1}{3}.\end{aligned}$$
Now, since the above inequality was derived based on too weak constraints on the vertex components, we have to show that the inequality is sharp. This is achieved by choosing the vertices of T as follows:
$$a = (0,0,0), \quad b = (x_m, 0, z_m), \quad c = (x_m, y_m, 0), \quad d = (0, y_m, z_m).$$
It is verified easily, that $\mathfrak{B}_T$ is the bounding box of this tetrahedron T and that
$$V_T = \frac{1}{3} x_m y_m z_m.$$

□

Now we can extend corollary 4.1.7:

Corollary 4.1.9. *Let T be a given regular tetrahedron with minimal bounding box $\mathfrak{B}_T$. Then the ratio of corresponding volumes satisfies*
$$0 < \frac{V_T}{V_{\mathfrak{B}_T}} \leq \frac{1}{3}.$$
and cannot be bounded away from 0.

In our practical application the lower bound can actually be bounded away from 0, because we are working with an integer grid:

Lemma 4.1.10. *Let $p_0, p_1, p_2, p_3 \in \mathbb{Z}^3$ span a regular tetrahedron T with minimal bounding box $\mathfrak{B}_T$. Then*
$$\forall\, V_{\mathfrak{B}_T} > 1 : \exists c > 0 : \exists p_0, p_1, p_2, p_3 \in \mathbb{Z}^3 : 0 < c \leq \frac{V_T}{V_{\mathfrak{B}_T}}.$$
This means the volume of V_T cannot be arbitrarily small compared to $V_{\mathfrak{B}_T}$ when considering regular tetrahedra on an integer grid for a fixed bounding box $\mathfrak{B}_T$.

Proof:
W.l.o.g. we will only consider positive integer coordinates in this proof. Let $\mathfrak{B}_T$ be a given bounding box with vertices in $\mathbb{N}_0^3$ and with volume $V_{\mathfrak{B}_T} > 1$. Then $0 < \left|\mathbb{N}_0^3 \cap \mathfrak{B}_T\right| < \infty$. Let M be the set of all regular tetrahedra with bounding box $\mathfrak{B}_T$ with vertices in $\mathbb{N}_0^3$. Since $V_{\mathfrak{B}_T} > 0$ holds, $\mathfrak{B}_T$ is a regular cuboid and $M \neq \emptyset$. Then an optimal choice of vertices $\widetilde{p_0}, \widetilde{p_1}, \widetilde{p_2}, \widetilde{p_3}$, spanning the tetrahedron $\widetilde{T}$ with corresponding bounding box $\mathfrak{B}_T$ resulting in a minimal volume $c := \min_{T \in M} V_T > 0$ must exist. This concludes the proof. □

Yet, as $V_{\mathfrak{B}_T} \to \infty$, it follows that $c \to 0$:

Lemma 4.1.11. *Let all prerequisites be the same as in lemma 4.1.10. Then*

$$\forall c > 0 : \exists V_{\mathfrak{B}_T} > 0 : \exists p_0, p_1, p_2, p_3 \in \mathbb{Z}^3 : \frac{V_T}{V_{\mathfrak{B}_T}} < c.$$

Proof:
We may prove the corresponding statement for triangles, since then the above statement results from applying the argument from lemma 4.1.6. W.l.o.g. we can prove the statement for a positive integer grid. Thus we have to show

$$\forall c > 0 : \exists A_R > 0 : \exists p_0, p, p_1 \in \mathbb{N}_0^2 : \frac{A_D}{A_R} < c.$$

Let $c > 0$ and w.l.o.g. $p_0 = (0, 0)$. Let $p \in \mathbb{N}_0^2$ be fixed with $p \neq p_0$. Now our goal is the construction of a rectangle R whose upper right vertex is $p_1 := (x_{max}, y_{max}) = (x_m, y_m)$ with $x_m, y_m \in N_0^2$ and which is the bounding rectangle of D, defined by p_0, p, p_1. p_1 has to be chosen such that p is close enough to the diagonal of R, which is defined by $\overline{p_0 p_1}$.
The area of D is given by

$$A_D = \frac{1}{2} \cdot \overline{p_0 p_1} \cdot h, \qquad \text{with } h := \overline{mp} \text{ and } m \in \overline{p_0 p_1}.$$

We now have to determine the intersection of the height of D and $\overline{p_0 p_1}$, denoted by $m \in \mathbb{R}^2$. A simple calculation yields

$$m = \begin{pmatrix} p_x \\ p_y \end{pmatrix} + \frac{p_x y_m - p_y x_m}{x_m^2 + y_m^2} \begin{pmatrix} -y_m \\ x_m \end{pmatrix}.$$

Hence

$$\begin{aligned} A_D &= \left| \frac{1}{2} \begin{pmatrix} p_x \\ p_y \end{pmatrix} \cdot \left(\begin{pmatrix} p_x \\ p_y \end{pmatrix} - \left(\frac{p_x y_m - p_y x_m}{x_m^2 + y_m^2} \begin{pmatrix} -y_m \\ x_m \end{pmatrix} \right) + \begin{pmatrix} p_x \\ p_y \end{pmatrix} \right) \right| \\ &= \left| \frac{1}{2} \begin{pmatrix} p_x \\ p_y \end{pmatrix} \cdot \frac{p_x y_m - p_y x_m}{x_m^2 + y_m^2} \begin{pmatrix} y_m \\ -x_m \end{pmatrix} \right| \\ &= \left| \frac{1}{2} \sqrt{x_m^2 + y_m^2} \frac{p_x y_m - p_y x_m}{x_m^2 + y_m^2} \sqrt{x_m^2 + y_m^2} \right| \\ &= \left| \frac{1}{2} \left(p_x y_m - p_y x_m \right) \right|. \end{aligned}$$

Now we want to derive constraints on x_m and y_m, such that

$$\frac{A_D}{A_R} < c \quad \wedge \quad 0 < p_x < x_m \quad \wedge \quad 0 < p_y < y_m \tag{4.4}$$

are satisfied. This yields

$$x_m y_m c > \left| \frac{1}{2} \left(p_x y_m - p_y x_m \right) \right| . \tag{4.5}$$

For the following calculations we assume

$$p_x y_m > p_y x_m \quad \Longleftrightarrow \quad x_m < \frac{p_x y_m}{p_y}. \tag{4.6}$$

Then $(p_x y_m - p_y x_m) > 0$ and we may omit the modulus for following calculations. Solving the above inequality for x_m and combining the result with the above assumption we get

$$\frac{p_x y_m}{p_y} > x_m > \frac{p_x y_m}{p_y + 2c y_m}. \tag{4.7}$$

Now we want to determine conditions on y_m, such that $x_m \in \mathbb{N}$ may be chosen such that it satisfies the above inequality. To ensure this, we need

$$\begin{aligned} & \frac{p_x y_m}{p_y} - 1 > \frac{p_x y_m}{p_y + 2c y_m} \\ \Leftrightarrow \quad & y_m^2 - \frac{p_y}{p_x} y_m - \frac{p_y^2}{2c p_x} > 0. \end{aligned}$$

The calculation of the zeros $(y_m)_{1,2}$ of the above convex polynomial yields

$$(y_m)_{1,2} = \frac{p_y}{2p_x} \pm \sqrt{\frac{c p_y^2 + 2 p_x p_y^2}{4 c p_x^2}}.$$

Obviously $(y_m)_1 < 0$, so we can derive the following condition on y_m:

$$y_m > \frac{p_y}{2p_x} + \sqrt{\frac{c p_y^2 + 2 p_x p_y^2}{4 c p_x^2}} > p_y.$$

Now we have to ensure, that choosing y_m according to 4.8 allows a choice of x_m according to 4.7, such that $x_m > p_x$. Analysis shows, that the lower bound in 4.7 is not necessarily greater than p_x and in this case we can derive a different lower bound on x_m by demanding

$$\frac{p_x}{p_y} y_m - 1 > p_x,$$

which transforms to

$$y_m > p_y + \frac{p_y}{p_x}.$$

Hence we may choose y_m as

$$y_m > \max \left(p_y + \frac{p_y}{p_x}, \frac{p_y}{2p_x} + \sqrt{\frac{c p_y^2 + 2 p_x p_y^2}{4 c p_x^2}} \right) > p_y. \tag{4.8}$$

Thus if y_m is chosen such that it satisfies 4.8, we can choose $x_m \in \mathbb{N}$ according to 4.7, which then implies $x_m > p_x$. Therefore the pair (x_m, y_m) satisfies 4.5 and 4.4, which finally proves the initial claim. An analogous construction of a pair (x_m, y_m) may be derived, if instead of assuming 4.6 the relation is switched. Then all following calculations yield the same result except for a switch of p_x and p_y and x_m and y_m, so x_m is chosen first. □

Remark 4.1.12. Note that using the construction in the above proof, relatively small triangles and bounding boxes may be calculated by choosing p_x and p_y to be small numbers. This suggests, that despite the existence of a lower bound on the volume ratio in practical applications arbitrarily small ratios may occur, leading to an inefficient assignment process.

Finally we are able to formulate the reason for the poor performance of the assignment algorithm in the original AT3D-implementation:

1. Due to the large number of considered tetrahedra, the calculation of all the individual sets Q_{T_j} is relatively expensive.
2. Corollary 4.1.9 shows that even in the very best case $\frac{2}{3}$ of the points checked in one iteration do not belong to the current tetrahedron and will therefore necessarily be considered again in at least one of the following iterations. For a significant pixel p with cell size $|\mathcal{C}_p|$ and k thinned pixels to be assigned to the tetrahedra in $\mathcal{C}_p$ in the worst case for each of the $|\mathcal{C}_p|$ tetrahedra up to k points are checked before they are eventually assigned successfully. Despite the result from lemma 4.1.10 it is possible that no thinned point is located inside a tetrahedron, especially in early adaptive thinning stages.
3. While in the beginning of the thinning process usually $|\mathcal{C}_p| > k$ holds, soon the number of thinned pixels grows while the average cell size remains constant, resulting in $|\mathcal{C}_p| \ll k$. This explains a further deterioration of performance in combination with the previous statement.
4. The assignment of points to tetrahedra follows a trial-and-error scheme, there is no geometric information used and each assignment is independent of the one before. Tetrahedra are checked in the same order for each considered point. This can result in a very inefficient assignment procedure, if many of the points are located in tetrahedra checked at the end.

All of those issues were improved or solved in the new implementation. We will now present a short outline of the new assignment scheme and then work out the details:

1. The minimal bounding box of the cell is calculated in one step and all thinned pixels, which will be the query points of the location procedure are determined.
2. In this process the query points are collected in a special order to greatly reduce the Euclidean distance between all pairs of two consecutive query points.
3. For each query point a procedure called ***Lawson's oriented walk*** is used to efficiently use the geometric information obtained in the location procedure. This is an iterative algorithm that 'walks' through a tetrahedralization to locate a query point by using a geometric criterion to be defined.
4. After a successful assignment the final tetrahedron is remembered and set as a new starting point for the next query point. Thus the proximity of consecutive query points is used.

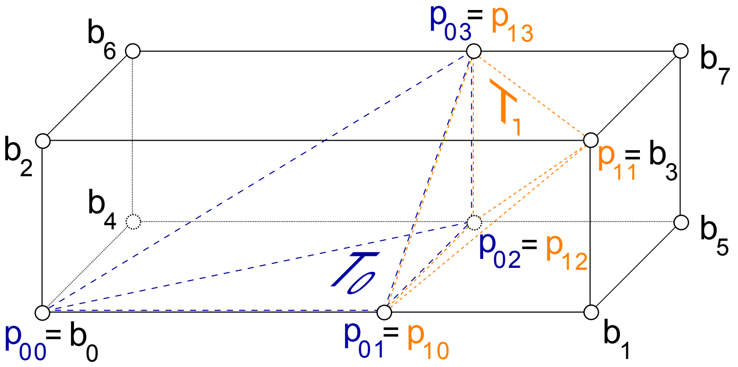

Figure 47: Minimal bounding box $\mathfrak{B}_{\mathbb{T}}$ of a set $\mathbb{T}$ of two tetrahedra.

Recall the definition of a bounding box of a set of tetrahedra, 4.1.3. In order to improve the computational efficiency of the assignment algorithm, we will consider the bounding boxes of sets of tetrahedra. That way we may exploit neighborhood relations within cells of tetrahedra for the walking algorithm.

Lemma 4.1.13. *Consider a finite set* $\mathbb{T} := \bigcup_{j\in\mathbb{N}} T_j$ *of tetrahedra, where all* T_j *belong to the cell* $\mathcal{C}_p$ *of a significant pixel* $p \in \mathbb{T}$*. Then the minimal bounding box containing* $\mathbb{T}$*,* $\mathfrak{B}_{\mathbb{T}}$*, is well defined and satisfies*

$$\bigcup_j \mathfrak{B}_{T_j} \subset \mathfrak{B}_{\mathbb{T}}.$$

Proof:
The bounding box $\mathfrak{B}_{\mathbb{T}}$ is well defined due to the finiteness of $\mathbb{T}$. The statement follows directly from the fact that finite sets $A, B \subset \mathbb{R}$ with $A \subset B$ satisfy

$$\min B \leq \min A \quad \text{and} \quad \max B \geq \max A.$$

□

Remark 4.1.14. Note that $|\mathfrak{B}_{\mathbb{T}}| > \left|\bigcup_j \mathfrak{B}_{T_j}\right|$ may hold, in other words, the total number of pixels to be assigned to tetrahedra may increase in the new implementation. Our numerical experiments will show, that this is only measurable in few cases, and it will not affect the performance (cf. number of geometry tests performed in appendix E).

Given a set $\mathbb{T}$ as defined in Lemma 4.1.13, we first calculate its minimum bounding box $\mathfrak{B}_{\mathbb{T}}$. Let m be a fixed thinning step. Then we calculate the set of thinned pixels in the current bounding box $Q_{\mathbb{T}} := \{q_i \in ((X \backslash X_{N-m}) \cap \mathfrak{B}_{\mathbb{T}})\}$. By lemma 4.1.13 this set satisfies $\bigcup_{j\in\mathbb{N}} Q_{T_j} \subset Q_{\mathbb{T}}$, so all pixels checked in the old implementation will be considered in the new one as well. The sequences $\widetilde{Q}_{T_j}$ were generated by adding the pixels of $\mathfrak{B}_{T_j}$ in a frame-by-frame order and in the frames following a row-by-row scheme, cf. figure 48. This corresponds to an ordering based on the pixel index $I(q)$. This scheme created large Euclidean distances between consecutive pixels in $\widetilde{Q}_{T_j}$ at the end of each row and at the end of each frame, thus rendering the information gained by the successful assignment of pixel useless.

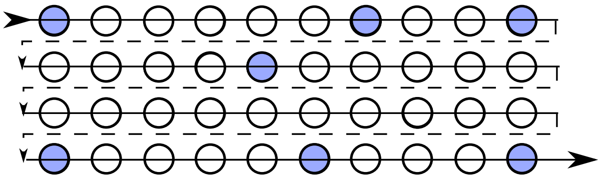

Figure 48: Schematic representation of old pixel scan implementation. Colored circles represent significant pixels, white circles represent thinned pixels.

The new implementation is visualized in figure 49. It eliminates these unnecessary jumps at the end of rows and frames by alternating the order, in which rows and frames are scanned to add pixels to the sequence of pixels, denoted by $\widetilde{Q}_{\mathbb{T}}$, after each row and each frame. The corresponding set of pixels is denoted by $Q_{\mathbb{T}}$. We will take advantage of this scheme later.

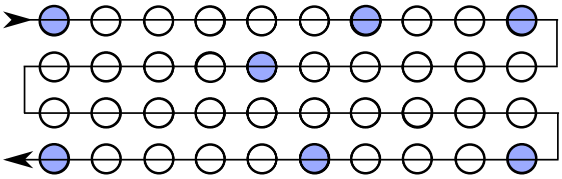

Figure 49: Schematic representation of new pixel scan implementation. Colored circles represent significant pixels, white circles represent thinned pixels.

From a theoretical point of view the calculation of one sequence $\widetilde{Q}_{\mathbb{T}}$ instead of several sequences $\widetilde{Q}_{T_j}$ does not improve the amount of failed assignments. The reason for that is, that it may still be necessary to check the position of each pixel in $\widetilde{Q}_{\mathbb{T}}$ relative to each tetrahedron in $\mathbb{T}$. In fact, as Lemma 4.1.13 suggests, there are even more pixels considered than before, suggesting a higher computational complexity.
Yet, in the practical implementation, just switching to iterating over the points in $\widetilde{Q}_{\mathbb{T}}$ yielded a small performance gain, most likely because only one sequence $\widetilde{Q}_{\mathbb{T}}$ had to be calculated per cell instead of one for each tetrahedron in a cell.
However, the switched iteration order permits the efficient use of geometric information gained by failed assignments. The following definitions and results are necessary to further specify our novel approach:

Definition 4.1.15. Let T be a regular tetrahedron, spanned by its vertices $p_0, p_1, p_2, p_3 \in \mathbb{R}^3$. Then each face $f_i(T)$, for $i \in \underline{3}$, of the tetrahedron is defined by three vertices p_k, p_l, p_m, where $k, l, m \in \underline{3}$ are pairwise distinct. The fourth vertex is p_n, with $n \in \underline{3} \setminus \{k, l, m\}$, and is called the ***vertex opposing*** f_n. Analogously, $f_n(T)$ is called the ***face opposing*** p_n.

Definition 4.1.16. Consider a Delaunay tetrahedralization $\mathcal{D}_Y$ of a set $[X] \subset \mathbb{R}^3$ with $Y \subset X$. If there is a tetrahedron $T_j \in \mathcal{D}_Y$ with a face $f_i(T_j)$ satisfying $(f_i(T_j))^\circ \cap \partial [X] = \emptyset$

and if T_j is spanned by the points $p_0, p_1, p_2, p_3 \in \mathbb{R}^3$, we can define the ***tetrahedron opposing*** p_i as the tetrahedron that shares the face $f_i(T_j)$ opposing p_i with T_j and denote it by T_{j,f_i,p_i}.

Remark 4.1.17. For a tetrahedron $T_j \in \mathcal{D}_Y$ satisfying the above condition, the tetrahedron opposing a vertex p_i is well defined: Consider the tetrahedron T_j with a face $f_i(T_j)$ opposing p_i. Its interior is located in the interior of the convex hull of X. Since the union of all tetrahedra $\bigcup_j T_j$ equals $[X]$, there must be a uniquely determined tetrahedron $T_l \in \mathcal{D}_Y$ with $T_l \cap T_j = f_i$.

Before we can define the walking-algorithm, we have to establish a result concerning the position of a pixel $p \in \mathbb{R}^3$ with respect to a plane $e \subset \mathbb{R}^3$, which is used to determine the tetrahedron investigated in the next iteration of the walking-algorithm:

Definition 4.1.18. Let $p \in \mathbb{R}^3$ and $p_0, p_1, p_2 \in \mathbb{R}^3$ be right-hand oriented points defining a plane $e \subset \mathbb{R}^3$. Let all $q \in e$ satisfy the Hesse normal form $qn_0 - d = 0$. Here the unit normal vector of e is denoted by n_0 and the distance between e and the origin by d.
We call $(\mathbb{R}^3)_{e+}$ the ***positive half-space of*** $\mathbb{R}^3$ ***with respect to*** e and define it as

$$(\mathbb{R}^3)_{e+} := \left\{ x \in \mathbb{R}^3 \,\middle|\, xn_0 - d \geq 0 \right\}.$$

We call $(\mathbb{R}^3)_{e-}$ the ***negative half-space of*** $\mathbb{R}^3$ ***with respect to*** e, it is defined as

$$(\mathbb{R}^3)_{e-} := \left\{ x \in \mathbb{R}^3 \,\middle|\, xn_0 - d < 0 \right\}.$$

Remark 4.1.19. The two subsets $(\mathbb{R}^3)_{e+}$ and $(\mathbb{R}^3)_{e-}$ obviously satisfy $(\mathbb{R}^3)_{e+} \cap (\mathbb{R}^3)_{e-} = e$ and $(\mathbb{R}^3)_{e+} \cup (\mathbb{R}^3)_{e-} = \mathbb{R}^3$.

Remark 4.1.20. Note that all of the following procedures also work, if the points are left-hand oriented and the signs are switched where necessary. However the choice of the orientation has to be consistent. Positive and negative half-space of $\mathbb{R}^3$ w.r.t. e are swapped, if the orientation of the points defining e is switched. We will only use right-hand oriented points.

Now we can specify a criterion to locate a query point $q \in \mathbb{R}^3$ in a tetrahedron:

Definition 4.1.21. Let T be a regular tetrahedron with faces $f_i(T)$ with $i \in \underline{3}$. Let the vertices defining the faces $f_i(T)$ be ordered such that they are right-hand oriented. Let $q \in \mathbb{R}^3$, then $q \in T$, iff q is in the positive half-space of $\mathbb{R}^3$ with respect to all $f_i(T)$, i.e.

$$q \in T \quad \Leftrightarrow \quad q \in (\mathbb{R}^3)_{f_i(T)+} \text{ for all } i \in \underline{3}.$$

Remark 4.1.22. Note that the orientation of a tetrahedral face depends on the order of the vertices that define the face. In the current AT3D implementation the vertices of a tetrahedron are ordered such that the right-hand oriented vertices p_0, p_1, p_2 span a plane e and p_3 is in the positive half-space with respect to e. From this order it can be concluded how to order the tuples defining the tetrahedral faces to satisfy the conditions in the previous theorem:

$$f_0 : (p_1, p_3, p_2), \quad f_1 : (p_0, p_2, p_3), \quad f_2 : (p_0, p_3, p_1), \quad f_3 : (p_0, p_1, p_2).$$

Remark 4.1.23 (Computational cost of geometry test)**.** The algorithm, which determines the location of a point $p \in \mathbb{R}^3$ relative to a plane e requires about 24 multiply- and add-operations, some of them applied to 8-byte integers. When considering, that the number of these geometry tests is in the billions when compressing a typical sequence at `qcif`-resolution, we can clearly see that a significant reduction is likely to have an impact on the overall runtime. Our numerical results support this claim.

Now we can define the ***walking-algorithm***, which is used to find the tetrahedron containing a query point $q \in \mathbb{R}^3$. Note that there are several walking techniques available, Lawson's oriented walk being the most well-known one. It is easy to implement, offers good theoretical and practical performance and all necessary data structures were available in the AT3D implementation, because the same walking technique was used in some other part of the algorithm before.
The central idea of Lawson's oriented walk can be summarized as follows: After starting in an arbitrarily chosen tetrahedron T_j, a face $f_i(T_j)$ is determined, which separates q from T_j. Then the tetrahedron for the following iteration is the one sharing $f_i(T_j)$ with T_j. This procedure is continued, until no such face exists and the algorithm terminates in a tetrahedron U_k containing q.

Remark 4.1.24. Other walking techniques include the ***straight walk***, which defines a straight line between starting point p and query point q and walks along all those tetrahedra, that are intersected by this line. As [Dev02] notes, the implementation of the three-dimensional version of this algorithm is a delicate task due to the need for handling degenerate cases. Also, [Dev02] includes numerical results on the speed of different query point location algorithms which are of the same order for all tested algorithms. While the straight walk performs best in terms of the numbers of visited tetrahedra, the most orientation tests per tetrahedron have to be performed, while the ***orthogonal walk*** visits far more tetrahedra with a smaller number of tests to be performed per tetrahedron. The orthogonal walk divides the straight walk into walks parallel to the coordinate axes, which simplifies the necessary geometric orientation tests drastically. Its biggest drawback is the possibility of leaving the tetrahedralization during the procedure, which has to be handled by the implementation.

Yet several studies indicate that implementations of the orthogonal walk or improved versions of it need less computational time than Lawson's oriented walk, cf. [Sou11]. We chose not to implement an orthogonal walk algorithm for the following reasons:

- The coding effort would be relatively high compared to the expected performance gain.
- Investigations indicated, that a large portion of the point locations only take $1-2$ walking steps, which would not be performed quicker by an orthogonal walk, because its last location steps are very similar to the ones performed by Lawson's oriented walk.

Now we can specify Lawson's oriented walk algorithm.

Theorem 4.1.25 (Lawson's oriented walk)**.** *Let $\mathcal{D}_Y$ be a unique Delaunay tetrahedralization containing $j \in \mathbb{N}$ tetrahedra T_i, whose union equals $[X] \subset \mathbb{R}^3$. Let $q \in \mathbb{R}^3 \cap [X]$. Then the tetrahedron U_k with $q \in U_k$ can be located in finitely many iterations with algorithm 13.*

Algorithm 13 Lawson's oriented walk

```
Let $\mathcal{D}_Y = \bigcup_{0\leq i<j} T_i = [X]$ be a unique Delaunay tetrahedralization
Let $q \in [X]$ be a query point
$U_0 := T_0,\ k := 1,\ o :=$ false
while true do
    for $i = 0$ to 3 do
        if $q \in (\mathbb{R}^3)_{f_i(U_{k-1})-}$ then
            $U_k := U_{k-1,f_i,p_i},\ o :=$ true, $k := k+1$
            break
        end if
    end for
    if $o =$ false then
        break
    end if
end while
return $U_k$
```

Proof:
First we prove the correctness of the above algorithm: Since $\mathcal{D}_Y$ is a Delaunay tetrahedralization of $[X]$, a tetrahedron T_i with $q \in T_i$ must exist in $\mathcal{D}_Y$. The definition of the tetrahedron U_k is well-defined according to Remark 4.1.17. The algorithm only terminates, if $q \in (\mathbb{R}^3)_{f_i(U_k)+}$ for all i. This is equivalent to $q \in U_k$. Therefore the algorithm yields a correct result.
As stated in [Wel98], Lawson's oriented walk terminates in finitely many iterations, iff $\mathcal{D}_Y$ is a unique Delaunay Tetrahedralization.

□

We will use a slightly more efficient modified version of Lawson's oriented walk, which we will call ***Lawson's remembering oriented walk***: When the algorithm walks to a tetrahedron U_i we can skip one orientation test in the following iteration, namely the one

of the tetrahedral face that U_i shares with U_{i-1}. We can not only save the sign of this test, but also the value itself and only have to invert its sign to eventually use it for the evaluation at the query point if the orientation tests with respect to the remaining faces yield that q is inside U_i.
Unfortunately, introducing Lawson's oriented walk for query point location introduces a new problem: Given a significant pixel p and its cell $\mathcal{C}_p = \mathbb{T}$, a query point $q \in Q_{\mathbb{T}} \setminus \mathbb{T}$ may get assigned to a tetrahedron $\widetilde{T} \notin \mathbb{T}$. This behavior is not desirable, since then $q \notin \mathbb{T}$ and no spline evaluation is necessary. It is not possible to limit the considered tetrahedra to the ones in $\mathcal{C}_p$, because if p is removed from its cell and this area is re-tetrahedralized, the walking algorithm may fail and yield invalid results. Up to now only the location of a tetrahedron containing a query point $q \in \mathbb{T} \subset \mathcal{D}_Y$ was considered. Therefore it also has to be determined, if $U_k \in \mathcal{C}_p$ holds. For that reason we modify the algorithm suggested so far to perform the additional check, if $U_k \in \mathbb{T}$ holds:

Algorithm 14 Check, if $U_k \in \mathbb{T}$ holds

```
Let $\mathbb{T} = \mathcal{C}_p \subset \mathcal{D}_Y = [X]$ be a given subset of a unique Delaunay tetrahedralization
Let $q \in [X]$ be a query point
Determine $U_k$ with $q \in U_k$ with algorithm 13
if $U_k \in \mathbb{T}$ then
    return true
else
    if $q \notin \partial U_k$ then
        return false
    else
        if $q$ is located on edge $\overline{p_i p_j}$ of $U_k$ then
            Fill queue $\mathcal{Q}$ with all tetrahedra $U_i$ incidental to $p_i$ and $p_j$ and with $U_i \in \mathbb{T}$
        else
            Fill queue $\mathcal{Q}$ with tetrahedron $U_i$ with $q \in U_i \cap U_k$ and $U_i \in \mathbb{T}$
        end if
        for each tetrahedron $U_i$ in $\mathcal{Q}$ do
            if $q \in U_i$ then
                return true
            end if
        end for
        return false
    end if
end if
```

Note that this algorithm may use the results of the geometric tests performed to determine U_k, which makes it very efficient. More than one additional tetrahedron is only checked in the rare case of q being located on an edge of U_k and additionally only in this case the expensive calculation of tetrahedra incidental to vertices of U_k has to be performed. The superior efficiency of the combination of algorithms 13 and 14 compared to the old implementation is shown in figures 52 and 54, and in tables 36 and 37.

Remark 4.1.26. Also note that in some places of the adaptive thinning algorithm, the assignment of query points $q \in \mathcal{C}_p$ from a cell of a significant pixel p to tetrahedra in $\mathcal{C}_p$ is performed twice in a row: Then only in the first pass algorithm 14 has to be used to determine, if $q \in \mathcal{C}_p$ holds, and each pixel can be marked accordingly. In the second pass (which is necessary after the re-tetrahedralization of the cell) only the marked pixels are assigned, further diminishing the disadvantage of the additional pixel count induced by calculating $\widetilde{Q}_{\mathbb{T}}$ instead of the individual $\widetilde{Q}_{T_j}$.

Now we have all ingredients needed to describe the new assignment scheme:

Algorithm 15 Evaluation of thinned pixels in a cell $\mathcal{C}_p$ in AT3D_PAQ

```
Let $p \in Y$ be a significant pixel
Let $\mathbb{T} = \mathcal{C}_p \subset \mathcal{D}_Y = [X]$ be a given subset of a unique Delaunay tetrahedralization
Calculate $\widetilde{Q}_{\mathbb{T}}$ according to the pixel scanning scheme in figure 49
for each $q \in (\widetilde{Q}_{\mathbb{T}} \cap (X \backslash Y))$ do
    Determine $U_k$ with $q \in U_k$ using algorithm 13
    If necessary, check with algorithm 14 if $q \in \mathbb{T}$ holds, and if so, mark and evaluate it
end for
```

Finally, we can sum up the advantages of the new implementation:

- Geometric information is used efficiently to locate each query point,
- after a successful assignment, the following assignment process very likely starts within close proximity of the query point,
- the sequence of checked pixels is generated faster.
- The number of geometric tests performed during adaptive thinning is thus reduced drastically, as indicated by the overall result tables 36 and 37 and the individual result tables in appendix E. Additionally figures 51 and 53 clearly indicate a significant reduction of computational time needed for adaptive thinning and the pixel exchange, especially in the latest stages.

Remark 4.1.27. In order to implement the above algorithm, in the class `Thinning3D` the function `at6()` was modified and most of the `calculate*Significance*()` functions and subroutines called from there were rewritten.

4.1.2 Custom queue implementation

The original AT3D implementation made heavy use of queues from the C++ standard library. A queue is a data structure containing elements of a certain type (pointers to tetrahedra and to vertices in the case of AT3D). They follow the so-called FIFO-principle (First In - First Out). The standard library implementation of queues is not well-suited for AT3D, because often the memory for elements pushed into the queue has to be allocated

when a new element is added, and has to be freed again when the queue is destroyed. Due to the large number of operations performed using queues the number of slow memory allocations and deallocations is very large.
Therefore the new implementation was designed to reduce this number as much as possible: Since the maximum queue size is bounded above by some $c \ll 1000$ in most cases, the memory for $c := 200$ elements is allocated at the construction of the queue $\mathcal{Q}$ in the new implementation. The queue is implemented as a ring buffer, which means there is a

- write index w,
- read index r,
- counter n for the number of elements already in the queue,
- queue size limit c.

In the unlikely case of a queue needing more space than allocated before hand, its size is increased exponentially and all existing elements are copied into the newly allocated memory. Obviously the number of these re-allocations has to be kept small to avoid this slow procedure.

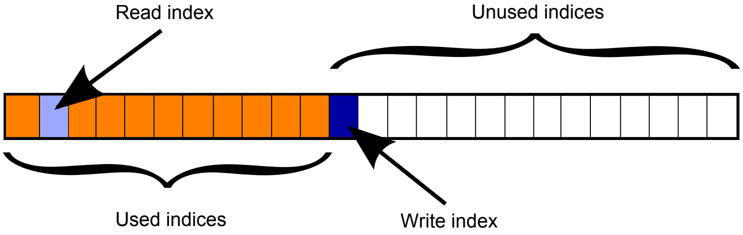

Figure 50: Schematic display of a ring buffer.

The main queue operations are adding new elements (***push***), getting the element at the head of the queue (***front***) and removing the element from the head of the queue (***pop***). The ring buffer implementation works as follows:

Definition 4.1.28 (Queue implementation via ring buffer)**.** A ***push*** is performed by writing the new element into the already allocated memory at index w, incrementing n and then incrementing w. If then $w = c$, then w has to be set to 0 again, which marks the beginning of the allocated memory. If additionally the queue size limit is reached, i.e. if $n = c$, the whole queue is moved to a new memory location, where exponentially more memory is allocated before populating it with the existing elements.
A ***front***-operation is performed by returning the data at the read index.
A ***pop*** is performed by incrementing r and if the end of the allocated memory was reached, i.e. if $r = c$, then r is reset to 0, which marks the beginning of the allocated memory. Then n is decremented.

Additionally the number of created queues was reduced in the new implementation: All used queues have to be empty when exiting a function, so they are in the same state at the start and at the end of this function. Therefore they can be declared as `static` queues, so that they are only constructed once, namely at the beginning of the program execution and destroyed only at the end of the program execution.

While our new implementation is twice as fast as the original when considering non-optimized code, it is still about 20% faster with the full set of compiler optimizations turned on. Those changes had a measurable effect on the tetrahedralization process which takes place in all stages of AT3D, thus leading to a significant decrease in total runtime.

Iterations[M]	0.1	0.2	0.5	1
Standard [ms]	488	1773	9996	39212
Custom [ms]	429	1409	7493	29371
%	12.1	20.6	25.1	25.1

Table 29: Comparison of standard queue implementation vs. custom implementation. In each iteration i about $i/100$ elements were written to the queue and about $i/10$ were read.

4.1.3 Further improvements

In this section we will sum up all other improvements that were implemented into AT3D. Most of them are minor improvements, in total they contribute about $5 - 10\%$ of the performance improvements.

1. In some heavily used functions the number of addition- and multiplication-operations was significantly reduced: In `inSphereDet()` the number was reduced from 71 to 65 and in `reconstructPointValue()` it was reduced from at most 123 to at most 103.

2. Additionally, `inSphereDet()` and `product()` were modified, such that some of the calculations, which only need a 32-bit range, are performed using 32-bit integers instead of 64-bit integers. A manual assembler implementation using SSE2-instructions was considered, but did not yield an improvement. Yet the investigation of CPU-specific optimization switches in the following section suggests, that a highly optimized SSE2-implementation may yield further improvements.

3. In early thinning stages, in which no error is induced by the removal of edges and points, in general the removal of an edge is favored over the removal of a point in the original implementation. To further improve removal efficiency in this setting, now the number of neighbors of the first two edges from the heap is compared and the one with less neighbors is removed, which is equivalent to the reduction of the size of the re-tetrahedralized cell.

4. The simulation of simplicity algorithm sorts a set of five points frequently. In the old implementation a slow bubble sort algorithm was used. The fastest solution in this case is hard-coding a tree of decisions to determine the correct order, which is now implemented and which is about twice as fast as the old sorting function.

5. Queues were allocated and deallocated on each call and return of functions defining them. These were saved by declaring them as `static`, since they had to be empty by then.

6. In several functions vectors from the C++ standard library are filled with points with specific properties. It is very likely, that these vectors become large, especially in later stages of adaptive thinning. Therefore the memory needed for them is pre-allocated, reducing the time needed for memory allocation.

4.2 Numerical results

In this section we will present the overall results of the conducted runtime tests. First an introduction to the technical background of compiler optimizations is given in order to explain the different settings that are presented later. We will then continue with selecting two test environments in which we generated the majority of the results. These are summarized later, while the detailed outcomes are listed in appendix E.

Remark 4.2.1 (Compiler-specific results)**.** A ***compiler*** is a computer program, which translates source code written in a specific programming language into binary code, which is executable by a computer. Thus, the compiler is specific to the hardware and the operating system being used. As specified in appendix A, the native Windows version of the ***GNU Compiler Suite*** was used in the development process of this thesis. Since this suite is not only free and distributed under the GNU license, it is also available for Linux and Mac OS. Therefore it is very likely to achieve similar results when using these operation systems. Yet there might be some more room for compiler-based optimizations when using non-free compilers like the ***Intel compiler suite***.

Remark 4.2.2 (Compiler-optimized code)**.** The conversion of source code into binary code is not unique. Therefore different binary codes may be produced yielding the same computational results, but possibly in different running times. This is the goal of code optimizations performed by a compiler. This process may be compared to entropy coding: Without any specific knowledge of the task performed by the code the compiler minimizes the number of instructions of binary code to be performed at runtime, analogous to entropy coding, which minimizes the number of bits of an encoded bit-stream without specific knowledge of its contents.
Compilers generally offer different optimization levels. Due to the vast runtime improvements by compiler optimizations we will only consider code that was generated by using the maximal level of global optimization switches. Non-optimized code is about two to three times slower, thus it is not of practical relevance. Another set of optimizations is applied, when ***hardware-specific optimization switches*** are activated. These make use of processor instruction sets, which may only be executed on certain processor architectures, e.g. MMX, which is supported by Pentium MMX and newer processors.

Definition 4.2.3. From now on we will consider ***optimized code*** to be code generated by the GNU C++ compiler with the option `-O3`.

All tests were conducted on two different hardware configurations (cf. appendix E), which we will call ***configuration A*** and ***configuration B***. Due to the different CPU micro-architectures, we also considered three different architecture-specific optimization settings of the GCC-compiler and conduct experiments to find the best combination for further experiments:

- `-march=barcelona -mtune=barcelona` for the AMD configuration, abbreviated by ***Ba***,
- `-march=core-avx-i -mtune=core-avx-i` for the Intel configuration, abbreviated by ***Co***,
- `-march=i686 -mtune=i686` for both configurations, abbreviated by ***i686***. This corresponds to the GCC standard settings.

Remark 4.2.4 (32- and 64-bit versions)**.** The x86 hardware architecture, which has been the standard for office and home computers for the last two decades, supports a word length of 32 bit. Thus CPU-registers and memory addresses may assume 2^{32} different values, and due to additional restrictions imposed by the operating system a memory address space of 2^{31} bits = 2.048 GB is available to a 32-bit program. The same limit applies to 32-bit programs executed in a 64-bit operating system. Depending on the input video size, AT3D may need to allocate more memory, therefore a 64-bit compilation was considered. Modern micro-architectures and operating systems support 64-bit word lengths, which removes the limitation on the maximum amount of allocated memory. In return, performing identical operations as the corresponding 32-bit version of a program, more memory is needed, since all pointers are 64-bit values. The memory requirements for AT3D grow by approximately 40%. The size of a sequence to be compressed by AT3D is only limited by the computer's memory when using a 64-bit compilation.

Since a detailed performance analysis of all eight combinations of configuration, optimization and architecture with a sufficient amount of examples is out of the scope of this thesis, we will first select two main combinations by investigating the impact of each choice on a small set of examples. We start by checking the impact of CPU-specific optimizations on a fixed architecture for both, the old and the new implementation.

Remark 4.2.5 (AT3D runtime characteristics and resulting testing methodology)**.** Due to its implementation, AT3D has the properties of producing non-deterministic thinning results, if more than one element in one of the employed heaps has significance 0. This is caused by non-deterministic memory allocation behavior: the minimal element of the heap is not unique. It then depends on internal memory allocation processes, which element is returned as the minimum, resulting in a different thinning process and thus in a different

tetrahedralization. Multiple runs of the same thinning process showed, that if a fixed amount of pixels is removed, thinning-runtimes vary below 0.5%. For that reason we refrained from conducting each test multiple times.
Additionally old and new implementation cause slightly varying thinning and exchange results. This is caused by the summation of floating point values: The significance of a current point or edge is calculated by summing the interpolation-induced errors at each pixel inside a tetrahedron, before and after re-tetrahedralizating the cell of the current point or edge. Since the order of the additions was changed in the new implementation, the employed floating-point arithmetic causes differences in the significance calculation. This may change the point- or edge-order in the corresponding heaps, resulting in a different thinning and exchange process when comparing old and new implementation.
Again, the effect on thinning runtimes is negligible. Yet, the exchange algorithm behavior may vary significantly. In order to minimize the effect of different starting sets of significant pixels, exchange runtime tests were performed in continuation mode of AT3D, based on identical pixel sets for old and new implementation.
Unfortunately, there is no way of measuring the exact runtime improvement on a single test sequence due to the above reasons. Thus individual exchange improvement results have to be considered with care, instead of focusing on minimal and maximal gain, we will only focus on the average performance improvement of the exchange algorithm.

Definition 4.2.6 (Command-line arguments for adaptive thinning runtime tests)**.** Adaptive thinning runtime tests were performed with the following command-line arguments:

```
at3d.exe -c%nbPoints% -a6 -g-1 -f500 -p -l -m -q1 -d %inputFilename%
```

- `%nbPoints%` is the number of removed pixels, chosen manually for each sequence.
- `%inputFilename%` is the sequence to be coded in `.pgmv`-format.
- Sequence is encoded in a single block of frames.
- No edge-detection algorithm is applied.
- Adaptive thinning is used in version 6, cf. algorithm 4.
- Each test was conducted once.

On the next page we will present the results used to determine a test environment based on a small test set.

Sequence	Ba-32 Configuration A			i686-32 Configuration A		
	Old [s]	New [s]	Gain [%]	Old [s]	New [s]	Gain [%]
`City`	28639	14131	50.66	**28420**	**13841**	50.99
`Galleon`	28397	13744	51.60	**27737**	**13573**	51.07
`Highway2`	35004	16106	53.99	**34013**	**15796**	53.56

Table 30: Comparison of thinning performance of removal of 0.75M pixels with 32-bit compilations and `Barcelona`-optimizations versus standard optimizations on configuration A. Boldfaced numbers mark faster result compared to other optimization setting.

Sequence	Ba-64 Configuration A			i686-64 Configuration A		
	Old [s]	New [s]	Gain [%]	Old [s]	New [s]	Gain [%]
`City`	20876	12558	39.84	**19943**	**12455**	37.55
`Galleon`	**20888**	12671	39.34	21243	**12463**	41.33
`Highway2`	**26285**	**14009**	46.70	26342	14257	45.88

Table 31: Comparison of thinning performance of removal of 0.75M pixels with 64-bit compilations and `Barcelona`-optimizations versus standard optimizations on configuration A. Boldfaced numbers mark faster result compared to other optimization setting.

Sequence	Co-32 Configuration B			i686-32 Configuration B		
	Old [s]	New [s]	Gain [%]	Old [s]	New [s]	Gain [%]
`City`	15138	8472	44.03	**14819**	**8192**	44.72
`Galleon`	14948	8259	44.75	**14628**	**8002**	45.30
`Highway2`	17954	9396	47.67	**17410**	**9292**	46.63

Table 32: Comparison of thinning performance of removal of 0.75M pixels with 32-bit compilations and `core-avx-i`-optimizations versus standard optimizations on configuration B. Boldfaced numbers mark faster result compared to other optimization setting.

Sequence	Co-64 Configuration B			i686-64 Configuration B		
	Old [s]	New [s]	Gain [%]	Old [s]	New [s]	Gain [%]
`City`	10979	**6881**	37.33	**10853**	6891	36.51
`Galleon`	10799	**6686**	38.09	**10660**	6704	37.11
`Highway2`	12911	**7574**	41.34	**12886**	7584	41.15

Table 33: Comparison of thinning performance of removal of 0.75M pixels with 64-bit compilations and `core-avx-i`-optimizations versus standard optimizations on configuration B. Boldfaced numbers mark faster result compared to other optimization setting.

Configuration A			
i686-32	Ba-32	i686-64	Ba-64
100%	101.78%	90.66%	90.81%

Table 34: Relation of runtimes between 32- and 64-bit-compilations with both optimization settings.

Configuration B			
i686-32	Co-32	i686-64	Co-64
100%	102.51%	83.10%	82.95%

Table 35: Relation of runtimes between 32- and 64-bit-compilations with both optimization settings.

Some of the results are highly unexpected, and we can draw the following first conclusions:

- The impact of CPU-specific optimizations and architecture on old and new implementation varies greatly, not only in absolute, but also in the performance relative to each other.
- 64-bit compilations perform significantly better than 32-bit compilations.
- CPU-specific optimizations tend to deteriorate overall performance, especially for 32-bit compilations. This indicates, that processor-specific instruction sets like MMX, SSE or AVX are not applied to AT3D compilations by the compiler. A further performance gain may be achieved by an effective assembler-implementation of the inner-loop functions `product()` and `insphereDet()`.
- There is a best-case-improvement scenario, in which the performance-improvement of the new implementation is at 50% or above, and a worst-case-improvement scenario, where the improvement is at 37% and above, the improvement differences between best- and worst-case scenario amount to 12-15%.
- Equal sized sequences get compressed in similar amounts of time. High-entropy scenes increase compression-times by about 10%.

The goal of this section is to determine two suitable combinations of compilation settings for the two computer configurations used. Thus, we choose a best-case-improvement setting with best performance of the new implementation and the worst-case-improvement setting:

Definition 4.2.7 (Best-case-improvement setting, CfgA-32)**.** We will denote the 32-bit compilation, which is run on configuration A with the following optimization switches by ***CfgA-32***:

```
-O3 -march=i686 -mtune=i686.
```

Definition 4.2.8 (Worst-case-improvement setting, CfgB-64)**.** We will denote the 64-bit compilation, which is run on configuration B with the following optimization switches by ***CfgB-64***:

```
-O3 -march=i686 -mtune=i686.
```

Before presenting the final results, we fix the settings for our exchange runtime tests:

Definition 4.2.9 (Command-line arguments for exchange runtime tests)**.** Pixel exchange runtime tests were performed with the following command-line arguments:

```
at3d.exe  -co -r3 -c0 -a6 -g-1 -f500 -p -l
          -m -q1 -d %inputFilename% %nodeInputFilename%
```

- `%inputFilename%` is the sequence to be coded in `.pgmv`-format.

- `%nodeInputFilename%` is the `.nodes3d`-file which was put out by an earlier (adaptive thinning) run with n iterations and defines the set of significant pixels X_n.
- Sequence is encoded in a single block of frames.
- The exchange radius is set to 3.
- The sizes of the sets X_n vary, depending on the sequence. Thus $|X_n|$ is additionally denoted in the result tables for each test.
- Each test was conducted once.

With these settings, we produced the results presented in appendix E. They are summarized in the following tables. The notion of failed geometry tests refers to the calculation, which assigns a query point $q \in \mathbb{R}^3$ to a half-space with respect to a plane $e \subset \mathbb{R}^3$ spanned by a tetrahedral face. Each test requires several dozens of multiplications and additions, so reducing the number of failed geometry tests is expected to directly influence runtime results. Our following numerical investigations clearly support this claim.

CfgA-32	
AT3D stage	**Avg. improvement** [%]
Initial heap construction	44.01
AT6	52.66
Exchange	43.41
Total (Heap, AT6, exchange)	48.88
Number failed geometry tests AT6	71.81
Number failed geometry tests exchange	73.44

Table 36: Average best-case improvements, sorted by AT3D stage.

Note that on configuration B we were able to conduct tests on larger sequences. These may be reviewed in appendix B. The following average results consider all tested sequences on this configuration.

CfgB-64	
AT3D stage	**Avg. improvement** [%]
Initial heap construction	25.47
AT6	39.83
Exchange	36.93
Total (Heap, AT6, exchange)	38.46
Number failed geometry tests AT6	71.52
Number failed geometry tests exchange	72.07

Table 37: Average worst-case improvements, sorted by AT3D stage.

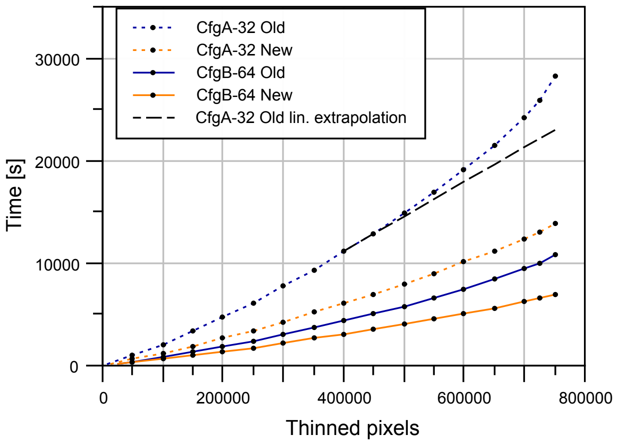

Figure 51: Progression of compression time for AT6 applied to `City`.

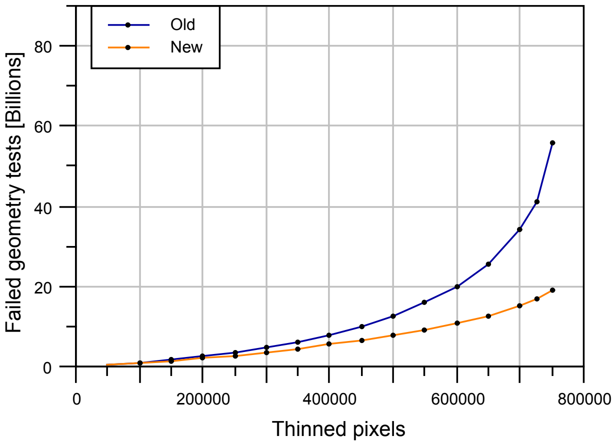

Figure 52: Progression of failed geometry tests in AT6 applied to `City`.

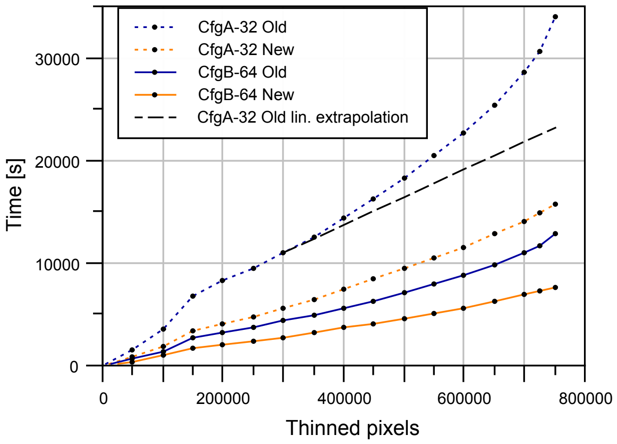

Figure 53: Progression of compression time for AT6 applied to `Highway2`.

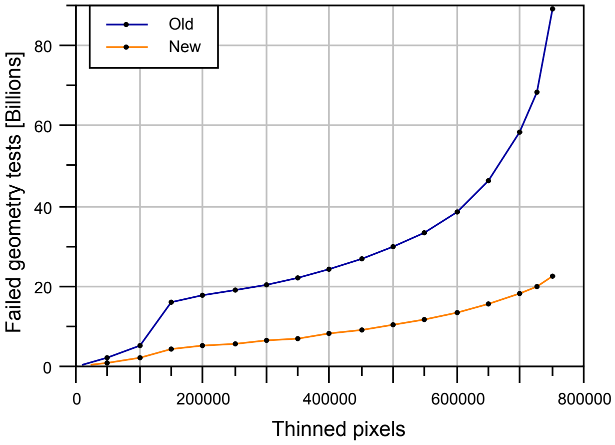

Figure 54: Progression of failed geometry tests in AT6 applied to `Highway2`.

After this complete overview, we additionally investigated the progression of compression times and failed geometry tests during the encoding process of two test sequences. Figures 52 and 54 clearly show that the number of failed geometry tests in the new implementation is significantly lower during the last thinning stages in particular. Additionally, with the help of the linear extrapolation in figures 51 and 53 we see a similar characteristic in the increase of runtime and the increase of the number of failed geometry tests. This clearly indicates, that a major reason for slowdowns of AT6 towards the end of the thinning process are eliminated in the new implementation.

4.3 Conclusion

In this section we described the changes applied to the AT3D implementation in order to improve its runtime. The most significant impact was achieved by introducing a more efficient pixel-to-tetrahedron assignment algorithm and by reducing the number of memory allocations. We proved some theoretical results, which underlined the inefficiency of the old implementation. From these investigations it was unclear, if the number of considered pixels for an assignment would grow measurably, but our numerical results clearly show, that this is not the case. On the contrary, the number of failed geometry tests, in which the position of a pixel with respect to a plane spanned by a tetrahedral face is calculated, was reduced by 72% on average. The numerical results also yielded, that the drastically reduced number of failed geometry tests also affected runtimes noticeably, in particular in the last thinning stage.

As a final result, our work resulted in an overall performance improvement of at least 38.5% and at most 49%. Adaptive thinning performance benefited the most, which is the most important part of the investigated runtimes. Despite all improvements further optimizations are needed, but these are left to future research.

4.4 Outlook

Despite all implemented improvements the overall time consumption to compress a video sequence is still too large. For that reason, further optimization of the implementation is an important issue. During the work on this thesis some more ideas on how to accelerate the algorithm were developed:

- Instead of calculating each tetrahedralization by using the modified flip-algorithm suggested by [Kha11], it would be possible to create a large database of tetrahedralizations that are pre-computed and just match the pixel configuration in each step to the database. This approach is feasible due to the use of an integer grid, thus the number of possible tetrahedralizations is limited, yet large. This would additionally introduce the need for a scaling mechanism, where point configurations are assigned to equivalence classes based on their relative distances and an efficient look-up scheme for the database. Analysis with a profiling application indicates that $50 - 60\%$ of the computational time is currently used by `Tetrahedralization::insertPoint()`, which adds a point to a tetrahedralization, indicating huge potential for further optimizations.

- When the significance of a pixel p is calculated, the resulting temporary tetrahedralization of $\mathcal{C}_p$ is deleted at the end of the calculation. When p becomes the least significant pixel and is removed from the tetrahedralization, the new tetrahedralization of $\mathcal{C}_p$ is calculated again. Saving it after the first calculation and then using it would further increase the performance of AT3D, but at the cost of increased memory requirements.

4.5 Further code improvements

In the development process of this thesis many minor improvements and additions were applied to the code base of the AT3D implementation. Therefore we briefly summarize the new options, which are now available. For usage information and a current list of parameters for AT3D_PAQ, we refer to the appended disc.

- Many bugs in the program code were fixed, yet it is likely, that at some point still a memory corruption happens, leading to a crash in rare cases.
- In the original AT3D implementation it was not possible to continue an adaptive thinning run, e.g. by removing n pixels in the first run and remove m additional pixels starting from the first run. This is now possible with the so-called ***continuation mode***, which needs the original video data and the geometry output of the first run.
- Instead of having to fix the number of pixels to be removed, a target PSNR feature has been implemented. The program then determines when to stop adaptive thinning by estimating the current PSNR in each iteration. Yet, the varying improvements by post-processing may only be estimated, thus it is not exact.

Appendices

A List of used hard- and software

Remark A.0.1 (Hardware)**.** The following configurations were used for computations:

1. Configuration A:
    (a) AMD Phenom II X3 720, 2.8 Ghz, 3 CPU cores (AMD K10 architecture),
    (b) 4 GB DDR2-RAM,
    (c) OCZ Vertex 2 SSD,
    (d) Windows 7 Professional 64-bit.
2. Configuration B:
    (a) Intel Core i5-3570k, 4.3 Ghz, 4 CPU cores (Ivy Bridge architecture),
    (b) 8 GB DDR3-RAM,
    (c) Samsung SSD 840,
    (d) Windows 7 Professional 64-bit.

Remark A.0.2 (Software)**.** In the development process several different versions of the following programs were used, here we will only list the latest versions used to generate the final results:

- Development-IDE: Eclipse IDE for C/C++ Developers, Version: Kepler Release, Build id: 20130614-0229.
- Compiler-Suite: MinGW-w64 with GCC compiler version 4.8.1, parallel installations for
    1. 32-bit target systems (`i686-w64-mingw32`),
    2. 64-bit target systems (`x86_64-w64-mingw32`).
- FFMpeg version N-55066-gc96b3ae, built on Jul 29, 2013.
- YUV conversion: viEWYUV 0.62.
- YUV to PGMV-conversion: AT3D.
- Profiling: Very Sleepy, Version 0.82.

B Test video sequences

Name	Content	Motion	Cam. motion	Texture
City	View down on New York City from a helicopter flying around the Empire State building	-	+	+
Deadline	Man talking in front of bookshelf	-	-	+
Football	Dynamic scene showing an American football play	+	+	o
Galleon	Still view on a complex textured ship anchoring at a pier	-	-	+
Harbor	Sailing boats cross the scene, partially occluded by anchoring boats in the front	+	-	+
Highway	View from hood-mounted camera on a car driving down a highway	+	-	-
Highway	View from hood-mounted camera on a car driving down a highway, crossing below a bridge, with changing contrast afterwards	+	-	o
Ice	People crossing the scene skating on ice	+	-	o
MissAmerica	Woman talking in front of dark solid background	-	-	-
Soccer	Dynamic scene at a soccer practice game, includes very fast camera scrolling	+	+	o
Stefan	Closeup of tennis player in motion running to get a ball and hitting it	+	+	o
Suzie	Blond woman on the phone in front of solid colored background	-	-	-
Tempete	Plant shown from close range and zooming out while leaves fly around	o	o	o
Tennis	Table tennis player shown from close range while serving and camera zooming out	+	+	o
Trevor	Several people shown while getting up at end of a conference in six separate boxes	o	-	o
WashDC	Hand drawing on a map of Washington D.C.	-	-	+
RotGrowBox	Small video of a rotating and growing box with gradient content on black background	+	-	-
RotEll	Centered rotating white ellipse on gradient background	+	-	-
RotGrowEll	Centered rotating and growing white ellipse on gradient background	+	-	-
TranslEll	White ellipse moving from left to right with constant velocity on gradient background	+	-	-
TranslRotEll	White ellipse moving from left to right with constant velocity while rotating on gradient background	+	-	-

Table 38: Test sequence overview - sequences in this table were downloaded from [Der13], except for the bottom five, which were generated by a sequence generator developed in this thesis. Symbols in the rightmost columns indicate sequences containing a high (+), medium (o), or low (-) amount of a characteristic.

Name	Content	Motion	Cam. motion	Texture
SSTB1450	Playfully drawn record player emerging from moving sea	+	-	+
SSTB2710	Chaotic expanding blurry mass	+	-	+
SSTB6355	Space background with several small objects moving downwards and a central element changing in every frame	+	-	+
SSTB7351	Low contrast rotating and shaking earth	+	+	-
SSTB9485	Cartoon scene showing a living room with several people, contains shaking object boundaries	o	o	+
SSTB28053	Butterfly moving towards the right on scrolling background with flowers and a window	+	+	+
SSTB46536	Complex structured stylized person in front of unicolor background	o	-	+
SSTB88250	Woman sitting by a river next to some bushes, rest of scene almost unicolored	-	-	o
SSTB94115	Two persons upper bodies shown while talking in front of slightly structured background	-	-	o
SSTB94910	Three people sitting around a fire	-	o	o
SSTB101980	Sun changing into the moon, illuminated scene fading to darkness	o	-	-
SSTB111130	Comet flying by on a dark space background with a few small stars	o	-	-
SSTB113530	Credits of the movie, many textlines scrolling on mostly black background	-	-	+

Table 39: Test sequences from the movie `Sita sings the blues`, which is available at [Pal08]. Symbols in the rightmost columns indicate sequences containing a high (+), medium (o), or low (-) amount of a characteristic.

Name	qcif	cif	qsif	sif	4sif
Resolution [px×px]	176x144	352x288	176x120	352x240	704x480
Pixels per frame	25344	101376	21120	84480	337920

Table 40: Overview of video resolutions employed in this thesis.

Name-*#Frames_resolution*	Frames from orig. sequence
`City-10_cif`	1-10
`City-10_qcif`	1-10
`City-20_qcif`	1-20
`City-30_qcif`	1-30
`Deadline-30_qcif`	700-729
`Football-30_qcif`	90-119
`Galleon-30_qcif`	1-30
`Harbor-30_qcif`	1-30
`Highway-30_qcif`	1-30
`Highway2-10_qcif`	1250-1259
`Highway2-20_qcif`	1250-1269
`Highway2-30_qcif`	1250-1279
`Ice-30_qcif`	1-30
`Miss America-30_qcif`	1-30
`Soccer-30_qcif`	50-79
`RotGrowBox-120_50x50`	-
`RotEll-30_100x100`	-
`RotGrowEll-30_100x100`	-
`SSTB1450-30_qcif`	1450-1479
`SSTB2710-30_qcif`	2710-2739
`SSTB6355-30_qcif`	6355-6384
`SSTB7351-30_qcif`	7351-7380
`SSTB9485-30_qcif`	9485-9514
`SSTB28053-7_4sif`	28053-28059
`SSTB46536-30_qcif`	46536-46565
`SSTB88250-30_qcif`	88250-88279
`SSTB94115-30_qcif`	94115-94144
`SSTB94910-30_qcif`	94910-94939
`SSTB101980-30_qcif`	101980-102009
`SSTB111130-30_qcif`	111130-111159
`SSTB113530-30_qcif`	113530-113559
`Suzie-10_qcif`	1-10
`Suzie-20_qcif`	1-20
`Suzie-30_qcif`	1-30
`Suzie-90_qcif`	1-90
`Stefan-30_qsif`	165-194
`Tempete-20_cif`	10-29
`Tempete-30_qcif`	10-39
`Tennis-20_sif`	70-89
`Tennis-30_qsif`	60-89
`TranslEll-30_100x100`	-
`TranslRotEll-30_100x100`	-
`Trevor-30_qcif`	10-39
`WashingtonDC-30_qcif`	220-249

Table 41: Overview of test sequences. Note that in the rest of this thesis sequences with 30 frames and `qcif`-resolution are just denoted by their sequence name.

C Detailed compression results

Compression results											
Sequence	$\lvert X_n \rvert$	Improvement/deterioration [%]							Total size @ q4 [kb]		
		Pixel pos.	Val q1	Val q4	Val q16	Total q1	Total q4	Total q16	Old	New	PSNR [dB]
City10_cif	313760	11.17	8.94	10.43	20.49	9.59	10.71	15.82	293.238	261.842	40.45
	213759	10.51	8.30	9.23	18.37	9.03	9.75	14.13	219.603	198.182	37.43
	138759	8.86	7.84	8.22	16.63	8.21	8.50	12.17	156.109	142.833	34.75
	88759	6.70	7.37	7.26	15.04	7.10	6.99	10.00	108.337	100.764	32.63
	58759	5.39	7.22	6.82	13.83	6.44	6.09	8.53	76.714	72.040	31.04
	33759	4.27	7.08	6.45	13.11	5.80	5.28	7.34	47.724	45.205	29.35
	25759	3.92	7.04	6.22	12.42	5.58	4.95	6.78	37.708	35.842	28.61
City10_qcif	93440	8.78	8.59	9.66	19.56	8.64	9.34	14.27	83.041	75.282	42.17
	53440	7.14	7.66	8.03	16.16	7.48	7.65	11.17	55.080	50.864	37.57
	33440	5.58	7.08	7.15	14.20	6.51	6.42	9.12	38.096	35.649	34.7
	18440	4.10	7.23	6.50	12.68	5.92	5.29	7.31	23.398	22.161	31.92
	10440	2.83	7.55	6.49	11.90	5.43	4.53	5.99	14.494	13.837	29.84
	6440	2.00	7.92	6.24	10.72	5.13	3.88	4.86	9.567	9.196	28.46
City20_qcif	156880	9.97	8.18	9.54	19.00	8.72	9.70	14.34	147.058	132.792	40.59
	86880	8.56	7.29	7.88	15.84	7.74	8.19	11.67	94.006	86.311	36.15
	46880	6.48	6.87	6.99	13.82	6.71	6.74	9.30	57.147	53.297	32.86
	26880	4.78	6.90	6.71	12.80	5.96	5.69	7.63	35.914	33.869	30.62
	16880	3.79	6.71	6.21	11.50	5.35	4.88	6.36	24.115	22.939	29.19
	11880	2.87	7.18	6.30	10.91	5.11	4.36	5.45	17.793	17.017	28.28
City30_qcif	250320	10.57	8.26	9.85	19.53	8.94	10.12	14.95	230.062	206.777	41.28
	170320	9.53	7.75	8.96	17.70	8.35	9.20	13.20	173.250	157.313	38.16
	110320	8.39	7.10	7.75	15.64	7.58	8.05	11.38	123.338	113.414	35.35
	70320	6.85	6.80	7.12	14.17	6.82	6.99	9.65	85.560	79.582	33.06
	40320	4.76	6.80	6.86	13.14	5.90	5.75	7.72	53.817	50.720	30.86
	25320	3.51	6.86	6.57	12.54	5.29	4.88	6.51	36.163	34.399	29.42
Deadline_qcif	150320	24.26	18.59	19.63	31.18	20.35	21.40	27.75	162.861	128.008	40.97
	110320	22.07	17.76	18.32	29.60	19.19	19.84	25.71	127.342	102.075	37.88
	70319	18.93	15.78	15.72	26.66	16.93	17.13	22.43	87.941	72.876	34.19
	40319	14.61	13.03	12.55	22.38	13.67	13.54	17.84	55.130	47.668	31.01
	20319	10.60	10.90	10.12	18.07	10.77	10.37	13.43	30.640	27.464	28.06
	10319	7.44	10.73	10.20	17.37	9.17	8.69	10.95	16.879	15.412	25.71
Football_qcif	200320	13.80	8.79	10.95	18.59	10.29	12.01	16.26	209.017	183.916	39.11
	160320	14.39	8.37	10.28	17.23	10.30	11.90	15.80	176.909	155.865	37.13
	120319	15.37	7.83	9.40	15.67	10.42	11.88	15.51	141.537	124.720	34.95
	80317	17.07	6.97	8.03	13.34	10.71	12.04	15.39	102.209	89.899	32.33
	50317	18.26	6.24	6.89	11.20	11.03	12.25	15.26	69.414	60.909	29.75
	30316	16.58	6.23	6.62	10.60	10.63	11.58	14.18	45.160	39.930	27.31
Galleon_qcif	150320	31.70	16.07	17.43	28.08	20.40	22.31	29.69	160.642	124.798	40.81
	120320	28.15	15.00	15.99	26.02	18.86	20.36	27.00	134.652	107.242	38.14
	90319	24.50	13.61	14.15	23.37	17.04	18.10	23.92	106.786	87.457	35.23
	60317	20.23	11.83	12.01	20.10	14.73	15.41	20.17	76.355	64.590	32.02
	40317	16.56	11.06	10.96	18.37	13.11	13.43	17.39	54.369	47.065	29.53
	20316	10.55	10.12	10.03	16.67	10.30	10.28	13.11	30.193	27.089	26.28
Harbor_qcif	250320	12.24	9.71	11.45	20.68	10.40	11.72	16.84	250.975	221.563	39.23
	175320	12.03	9.27	10.59	19.18	10.12	11.14	15.65	191.630	170.290	35.96
	125320	11.51	8.87	9.85	17.95	9.77	10.54	14.58	146.880	131.404	33.44
	85320	10.13	8.38	8.95	16.40	9.03	9.47	12.93	107.153	97.007	31.19
	45319	7.16	7.96	8.05	14.67	7.63	7.62	10.23	62.879	58.088	28.4
	15318	3.52	7.89	7.72	13.04	5.86	5.46	6.95	24.437	23.103	25.06
Highway_qcif	40320	8.14	11.22	11.50	24.15	9.94	9.80	13.88	45.868	41.374	41.28
	15320	7.94	11.86	11.05	22.49	10.05	9.34	12.72	20.586	18.663	38.26
	7320	5.34	13.47	12.13	23.28	9.47	8.18	10.78	10.923	10.029	35.98
	4320	2.96	14.46	12.15	23.71	8.55	6.60	8.85	6.874	6.420	34.49
	2320	1.33	15.72	13.28	22.89	7.99	5.79	6.96	3.958	3.729	32.86
	1320	1.72	17.20	14.09	23.38	8.58	6.10	7.03	2.428	2.280	31.59
	620	2.14	17.98	14.42	18.43	8.87	6.28	5.92	1.273	1.193	29.93

Table 42: Compression result comparison between AT3D and AT3D_PAQ. Total results and PSNR are only given for quantization `q4`. For more detailed results see appended disc. Parameters set as described in section 3.4.

Compression results											
Sequence	$\lvert X_n \rvert$	Improvement/deterioration [%]							Total size @ q4 [kb]		PSNR [dB]
		Pixel pos.	Val q1	Val q4	Val q16	Total q1	Total q4	Total q16	Old	New	
Highway2-10_qcif	45440	22.42	12.16	13.03	22.31	15.66	16.98	22.37	49.764	41.316	41.89
	27440	22.41	11.69	12.21	20.14	15.72	16.83	21.42	33.859	28.159	38.39
	20440	21.51	11.86	12.14	19.67	15.66	16.56	20.74	26.683	22.265	36.23
	15440	20.95	11.89	12.00	19.46	15.62	16.37	20.35	21.203	17.732	34.3
	10439	18.10	12.27	12.23	19.82	14.79	15.22	18.76	15.251	12.930	31.77
	7439	15.42	12.50	12.65	19.64	13.81	14.11	16.97	11.396	9.788	29.94
	5439	13.60	12.34	12.45	19.27	12.92	13.08	15.64	8.706	7.567	28.49
Highway2-20_qcif	81880	22.17	13.49	15.04	25.19	16.56	18.14	23.51	89.078	72.920	41.99
	41879	23.30	13.44	14.38	22.78	17.36	18.63	23.09	53.334	43.400	37.76
	31879	22.79	13.15	13.85	22.01	17.14	18.25	22.49	42.652	34.869	35.96
	21879	21.62	13.60	14.13	22.01	17.08	17.96	21.77	31.195	25.592	33.7
	13879	19.27	14.13	14.54	22.92	16.48	17.06	20.60	21.179	17.565	31.26
	9879	17.33	14.09	14.45	22.54	15.62	16.03	19.14	15.750	13.225	29.62
	6879	15.76	14.92	15.31	23.30	15.33	15.56	18.26	11.495	9.706	28.13
Highway2-30_qcif	120320	21.41	14.23	16.18	27.25	16.79	18.47	23.96	129.759	105.796	42.35
	60320	22.28	14.25	15.75	24.70	17.47	18.89	23.26	76.275	61.868	38.44
	35320	20.89	14.61	15.91	24.25	17.32	18.45	22.16	49.012	39.969	35.1
	20320	19.07	15.61	17.07	25.72	17.21	18.15	21.42	30.647	25.084	31.96
	12320	16.59	16.89	18.25	27.44	16.74	17.31	20.15	19.763	16.343	29.53
	7320	13.06	17.75	18.64	27.97	15.34	15.32	17.57	12.419	10.516	27.48
Ice_qcif	65320	30.73	12.25	12.73	23.53	19.66	21.41	27.79	81.030	63.684	40.93
	40320	26.63	12.14	11.91	22.38	18.45	19.53	25.03	53.827	43.312	37.3
	25320	23.42	12.61	12.23	22.60	17.65	18.38	23.14	36.009	29.389	34
	17320	19.72	13.20	12.64	23.31	16.38	16.69	20.90	25.860	21.544	31.36
	11320	14.93	13.66	12.90	23.67	14.30	14.10	17.66	17.878	15.358	28.86
	8320	11.51	13.61	12.53	23.83	12.52	11.92	15.26	13.702	12.069	27.36
MsAmerica_qcif	20020	4.91	14.25	15.56	26.90	10.14	9.87	12.30	24.198	21.809	43.57
	8020	2.49	12.68	10.97	20.41	7.76	6.09	7.95	11.261	10.575	40.93
	4020	1.41	12.60	9.65	17.15	6.91	4.71	5.93	6.260	5.965	38.78
	2020	0.37	13.44	8.95	15.43	6.50	3.66	4.50	3.497	3.369	36.54
	1220	-0.49	13.78	8.90	14.00	5.97	2.94	3.22	2.248	2.182	34.96
	620	0.12	16.05	10.21	13.31	6.96	3.57	3.30	1.259	1.214	33
RotGrowBox-120_50x50	50000	51.69	52.84	46.26	50.48	52.15	49.70	51.30	37.767	18.996	43.09
	35000	41.01	58.00	49.35	51.14	47.65	43.98	44.12	29.970	16.789	41.61
	25000	40.41	57.17	50.52	52.72	47.74	44.24	44.22	24.835	13.849	40.28
	17000	52.57	60.46	54.89	56.98	56.06	53.47	53.95	19.213	8.940	39.17
	10000	45.33	57.73	55.15	58.18	50.76	48.96	49.13	12.546	6.404	31.07
	6000	33.51	42.06	45.14	54.62	37.53	37.90	39.48	8.553	5.311	26.33
RotEll_100x100	10000	34.86	34.67	29.77	38.27	34.77	32.73	35.87	12.810	8.617	48.05
	5000	51.09	26.00	20.28	27.21	38.77	38.25	43.77	6.949	4.291	48.24
	2000	38.29	27.50	23.16	31.11	33.32	32.60	36.36	3.095	2.086	42.85
	1000	26.18	28.39	22.49	34.18	27.14	24.85	28.05	1.682	1.264	39.21
	500	17.17	27.97	21.56	31.25	21.84	18.69	20.79	0.963	0.783	33.61
	250	12.82	27.48	22.40	31.75	19.09	16.21	17.82	0.543	0.455	29.7
RotGrowEll_100x100	10000	51.05	25.90	22.06	30.30	38.62	38.69	44.41	13.733	8.420	48.27
	5000	40.54	27.79	23.67	31.61	34.63	33.96	37.99	7.442	4.915	43.16
	2000	24.99	31.20	25.42	36.18	27.66	25.14	27.73	3.306	2.475	35.33
	1000	16.17	27.54	23.86	34.05	21.08	18.92	20.76	1.876	1.521	29.95
	500	10.15	24.80	22.07	27.92	16.36	14.33	14.78	1.047	0.897	26.33
	250	7.18	25.56	22.44	29.20	14.86	12.56	13.06	0.581	0.508	23.66
Soccer_qcif	230320	19.13	12.06	13.83	25.78	14.15	15.80	22.54	225.602	189.968	43.39
	150320	20.39	12.27	13.60	24.92	14.96	16.36	22.58	166.387	139.174	39.78
	90320	23.34	12.01	12.58	23.15	16.15	17.32	23.26	111.334	92.051	35.59
	55320	24.16	12.51	12.63	23.34	17.13	18.06	23.81	74.187	60.789	31.67
	35320	23.07	12.68	12.59	23.16	17.07	17.80	23.11	50.879	41.825	28.43
	25320	21.46	12.48	12.21	22.30	16.42	16.95	21.79	38.344	31.845	26.38
SSTB 1450_qcif	80320	13.00	19.20	23.64	38.40	16.47	17.88	21.43	84.928	69.741	41.81
	55320	8.51	18.48	22.22	37.09	13.82	14.44	17.39	62.783	53.719	39.47
	35320	4.70	16.46	18.56	32.69	10.68	10.37	12.94	43.447	38.940	36.9
	20320	2.50	13.94	13.80	27.30	8.14	7.01	9.63	27.881	25.927	33.71
	10320	1.73	12.72	11.49	21.41	6.99	5.54	7.25	16.017	15.130	30.35
	5320	0.74	12.38	11.37	18.79	6.10	4.72	5.53	9.052	8.625	28.15
SSTB 2710_qcif	400320	8.06	6.87	7.93	14.15	7.11	7.97	12.04	363.877	334.893	41.08
	350320	8.00	6.70	7.65	13.66	6.98	7.75	11.54	332.442	306.693	39.02
	300319	8.12	6.51	7.31	13.00	6.90	7.55	11.05	298.295	275.764	36.87
	250317	8.36	6.42	7.12	12.56	6.93	7.52	10.77	261.114	241.477	34.7
	200314	8.54	6.37	6.94	12.22	6.99	7.50	10.55	220.606	204.065	32.46
	150310	8.72	6.35	6.75	11.88	7.09	7.50	10.36	176.058	162.859	30.07
SSTB 6355_qcif	100320	23.69	12.78	13.88	20.09	16.86	18.34	22.11	110.872	90.542	40.73
	70320	20.41	12.21	13.15	19.29	15.47	16.62	19.94	83.463	69.592	38.22
	45320	17.75	12.09	12.68	19.58	14.49	15.25	18.47	58.137	49.274	35.15
	30320	16.39	12.19	12.61	19.77	14.07	14.61	17.65	41.359	35.317	32.68
	20320	16.00	12.74	12.58	20.21	14.26	14.45	17.50	29.443	25.188	30.7
	12320	15.31	13.31	12.27	20.45	14.29	14.00	17.03	19.171	16.487	28.73

Table 43: Compression result comparison between AT3D and AT3D_PAQ. Total results and PSNR are only given for quantization `q4`. For more detailed results see appended disc.

Compression results											
Sequence	$\|X_n\|$	Improvement/deterioration [%]							Total size @ q4 [kb]		
		Pixel pos.	Val q1	Val q4	Val q16	Total q1	Total q4	Total q16	Old	New	PSNR [dB]
SSTB 7351_qcif	15320	17.90	22.45	24.28	33.60	19.96	20.16	21.63	20.885	16.675	41.88
	25320	22.98	23.35	18.51	31.78	23.16	15.86	18.47	11.320	9.525	38.87
	15319	17.84	22.60	17.68	31.18	20.00	14.75	17.35	7.252	6.182	37.22
	2320	12.24	20.11	15.36	28.33	15.54	13.27	15.86	4.251	3.687	35.41
	1520	13.93	19.55	13.21	25.77	16.28	13.70	16.59	2.957	2.552	34.28
	920	14.12	19.49	13.72	23.83	16.34	13.99	16.36	1.916	1.648	33.04
SSTB 9485_qcif	160320	9.93	13.38	14.94	27.45	12.30	12.98	18.46	160.034	139.254	41.8
	110320	7.65	12.30	13.26	24.91	10.71	10.90	15.58	119.434	106.413	39.06
	75320	5.63	11.36	11.76	22.44	9.26	9.03	12.96	87.555	79.653	36.59
	50320	4.27	10.23	10.10	19.78	7.89	7.34	10.65	62.608	58.011	34.36
	30320	3.06	9.82	9.09	18.08	6.98	6.07	8.87	40.669	38.200	32.06
	15320	1.71	9.37	8.03	15.64	5.88	4.64	6.66	22.642	21.591	29.51
SSTB 28053-6_4sif	320440	23.14	24.71	24.70	30.40	24.14	24.00	26.38	346.897	263.639	41.61
	250440	20.51	24.40	24.48	30.27	22.91	22.63	24.71	284.590	220.183	39.17
	190440	18.36	24.21	24.32	30.41	21.87	21.45	23.34	227.616	178.793	36.86
	140440	16.45	23.65	24.14	30.66	20.64	20.30	22.09	177.182	141.214	34.49
	100440	14.62	22.48	23.11	30.53	19.05	18.72	20.67	134.030	108.946	32.09
	60439	11.66	20.49	20.94	29.67	16.40	15.90	18.10	87.504	73.587	28.98
SSTB 46536_qcif	90320	41.90	19.89	16.33	23.86	26.44	25.77	32.51	94.499	70.142	42.53
	60320	31.88	16.62	14.10	21.92	21.76	21.41	27.11	68.787	54.061	37.04
	40320	25.17	15.53	13.57	21.67	19.11	18.76	23.62	49.513	40.225	33.65
	25320	19.65	14.92	13.38	22.03	16.84	16.42	20.62	33.602	28.086	31.12
	10320	11.55	14.46	12.58	22.13	13.10	12.02	15.25	15.605	13.729	28.26
	5320	7.34	14.05	11.78	20.75	10.68	9.18	11.52	8.706	7.907	27.01
SSTB 88250_qcif	100320	29.23	15.53	17.17	28.82	19.86	21.93	29.03	102.133	79.740	42.9
	65320	22.47	13.65	14.41	24.89	16.71	17.84	23.58	72.711	59.740	39.06
	45320	18.49	12.54	12.45	22.37	14.76	15.18	20.16	53.966	45.773	36.41
	30320	15.06	11.61	11.08	20.21	12.99	12.99	17.15	38.628	33.611	34.13
	20320	12.24	11.07	9.87	18.78	11.56	11.07	14.77	27.588	24.535	32.32
	10319	8.50	10.63	8.95	16.09	9.66	8.71	11.19	15.444	14.099	30.12
	6319	5.86	9.80	7.54	13.42	7.89	6.59	8.33	10.074	9.410	28.92
SSTB 94115_qcif	100320	37.11	28.89	27.04	40.00	31.67	31.24	38.46	105.852	72.780	43.22
	70320	35.04	26.01	23.45	35.90	29.13	28.35	35.44	78.771	56.438	41.23
	40319	30.15	25.81	19.66	30.96	27.39	24.29	30.51	48.922	37.041	37.95
	20319	22.45	20.03	14.36	25.81	21.01	18.28	23.82	27.135	22.175	34.38
	10319	13.54	14.79	11.28	21.47	14.23	12.47	16.43	15.232	13.332	31.07
	5319	8.28	13.34	10.55	19.97	10.89	9.28	12.22	8.663	7.859	28.18
SSTB 94910_qcif	100320	23.40	14.20	15.95	27.06	17.20	18.95	25.15	106.787	86.551	41.92
	70320	18.81	13.15	14.36	24.46	15.14	16.27	21.37	80.734	67.597	38.96
	40319	13.30	11.44	11.79	20.50	12.17	12.50	16.30	51.040	44.660	35.41
	20319	8.76	10.87	10.65	18.51	9.95	9.67	12.43	28.676	25.902	32.02
	10319	5.60	9.95	9.14	16.03	7.88	7.17	9.14	16.047	14.896	29.24
	5319	3.12	10.37	9.67	15.34	6.74	5.86	7.00	8.989	8.462	27.13
SSTB 101980_qcif	10320	19.87	11.16	9.32	20.44	15.22	15.17	20.06	14.361	12.183	41.57
	5320	11.98	10.42	7.28	16.87	11.20	10.04	13.43	8.090	7.278	38.88
	2820	6.97	11.05	6.75	14.35	8.91	6.86	9.02	4.693	4.371	36.43
	1620	4.71	11.57	5.61	12.52	7.83	5.04	6.73	2.878	2.733	34.3
	920	2.25	12.93	7.42	12.01	6.92	4.00	4.61	1.776	1.705	32.37
	520	2.71	13.85	10.03	13.10	7.35	5.15	5.17	1.107	1.050	30.37
SSTB 113530_qcif	15320	19.03	14.93	14.41	21.99	17.14	17.33	19.85	20.433	16.892	39.69
	8320	21.15	14.73	13.22	21.02	18.32	18.38	21.13	12.124	9.896	38.03
	4320	16.00	17.23	14.53	24.08	16.51	15.53	17.94	6.761	5.711	36.07
	2320	14.58	18.65	15.04	23.17	16.17	14.74	16.47	3.975	3.389	34.49
	620	20.09	18.21	8.76	12.56	19.39	16.96	18.63	1.285	1.067	32.87
	370	17.92	19.53	13.45	13.43	18.51	16.59	16.96	0.820	0.684	32.44
Stefan_qsif	233600	16.74	7.73	8.84	15.30	9.76	11.08	15.85	223.305	198.555	41.52
	153600	13.13	7.43	8.29	14.27	9.01	9.94	13.76	164.380	148.042	35.44
	123600	11.90	7.35	8.12	13.88	8.71	9.50	12.95	138.994	125.787	33.04
	98600	10.80	7.25	7.92	13.54	8.39	9.04	12.20	116.279	105.766	31.04
	78600	9.95	7.14	7.76	13.21	8.10	8.66	11.54	96.861	88.477	29.32
	63600	9.12	7.05	7.59	12.92	7.79	8.24	10.91	81.362	74.658	27.97
Suzie-10_qcif	18440	9.10	10.97	11.06	23.14	10.21	10.07	14.16	20.717	18.630	41.54
	8440	6.36	10.66	9.78	18.86	8.75	7.94	10.54	10.893	10.028	38.69
	3440	4.62	11.67	9.60	17.22	8.30	6.77	8.58	5.108	4.762	35.85
	1940	4.52	11.24	8.67	13.38	7.94	6.26	7.23	3.098	2.904	34.12
	1140	3.59	12.50	9.32	11.27	8.05	5.95	5.91	1.948	1.832	32.48
	540	2.69	12.46	9.09	10.08	7.47	5.29	4.93	1.001	0.948	30.22
Suzie-20_qcif	40880	7.03	11.12	12.04	25.41	9.48	9.56	13.76	44.933	40.637	41.84
	15880	3.87	9.82	9.55	19.95	7.11	6.45	9.14	20.729	19.392	38.45
	5880	1.25	10.95	9.82	17.42	6.16	4.84	6.15	8.970	8.536	35.46
	2880	-0.38	11.48	9.50	15.85	5.38	3.58	4.34	4.866	4.692	33.43
	880	-0.85	12.06	9.15	12.95	5.13	2.98	3.05	1.712	1.661	30.24
	480	0.80	12.23	9.38	10.46	5.97	4.01	3.48	0.997	0.957	28.45

Table 44: Compression result comparison between AT3D and AT3D_PAQ. Total results and PSNR are only given for quantization `q4`. For more detailed results see appended disc.

Compression results

Sequence	$\lvert X_n \rvert$	Improvement/deterioration [%]							Total size @ q4 [kb]		
		Pixel pos.	Val q1	Val q4	Val q16	Total q1	Total q4	Total q16	Old	New	PSNR [dB]
Suzie-30_qcif	70320	8.22	11.22	12.87	26.94	10.02	10.58	15.06	75.126	67.178	42.53
	25320	4.45	10.22	10.38	21.81	7.58	7.13	10.10	32.805	30.466	38.77
	13320	2.38	10.30	9.98	19.82	6.45	5.61	7.72	19.180	18.104	36.81
	6320	0.20	10.44	9.46	17.52	5.19	3.90	5.18	10.129	9.734	34.75
	1820	-1.40	10.96	7.82	11.76	4.24	2.05	2.14	3.417	3.347	31.52
	1020	-1.54	11.25	8.00	10.08	4.18	1.96	1.58	2.045	2.005	29.89
Suzie-90_qcif	150960	13.95	9.99	11.60	23.62	11.70	12.83	17.33	181.486	158.207	40.75
	120959	13.08	9.84	11.28	22.67	11.28	12.24	16.35	151.701	133.137	39.85
	90957	11.81	9.64	10.85	21.64	10.63	11.37	15.06	120.025	106.378	38.78
	60954	10.07	9.44	10.27	20.33	9.74	10.16	13.35	86.042	77.300	37.39
	30950	7.09	9.10	9.29	18.18	8.08	7.99	10.42	48.320	44.457	35.26
	10950	2.47	8.97	7.84	15.22	5.45	4.51	6.04	19.668	18.781	32.36
Tempete-20_cif	457520	9.98	10.40	12.45	21.21	10.28	11.55	15.86	473.417	418.737	40.18
	382520	9.02	9.95	11.71	20.11	9.66	10.68	14.62	412.498	368.439	38.66
	257518	7.57	9.33	10.67	18.44	8.73	9.38	12.71	301.661	273.366	35.63
	207518	6.98	8.87	9.93	17.32	8.19	8.65	11.70	253.087	231.196	34.22
	157517	6.38	8.52	9.29	16.34	7.71	7.97	10.74	201.607	185.531	32.62
	107517	5.59	8.10	8.48	15.19	7.09	7.09	9.57	146.338	135.956	30.76
	67517	4.68	7.72	7.72	14.03	6.40	6.17	8.31	98.334	92.262	28.82
Tempete_qcif	200320	13.74	10.17	12.01	20.82	11.21	12.64	17.41	201.107	175.686	41.2
	150320	10.93	9.65	11.16	19.40	10.06	11.07	15.10	161.732	143.832	38.36
	110320	8.83	9.08	10.18	17.92	8.99	9.62	13.09	126.724	114.539	35.78
	80320	7.33	8.49	9.04	16.35	8.07	8.28	11.34	98.061	89.941	33.62
	55320	5.74	8.09	8.27	14.92	7.16	7.08	9.60	72.056	66.956	31.53
	30319	3.87	7.74	7.11	13.34	6.08	5.47	7.54	43.354	40.984	28.89
Tennis_qsif	203600	18.18	11.73	13.87	25.76	13.62	15.48	21.98	187.071	158.121	43.91
	123600	23.66	10.47	11.36	21.41	14.83	16.38	22.60	131.692	110.116	40.65
	73600	27.62	9.66	9.61	18.32	16.09	17.41	23.38	88.252	72.884	37.28
	43600	24.28	9.79	9.67	17.72	15.37	16.38	21.43	57.581	48.148	33.9
	23600	19.76	10.12	9.58	17.68	14.18	14.62	18.92	34.245	29.238	30.47
	13600	15.53	10.89	10.12	18.01	12.99	12.97	16.47	21.398	18.623	27.98
Tennis-20_sif	399600	18.83	11.89	14.14	24.24	14.06	15.95	21.53	417.883	351.221	39.02
	324600	21.36	11.75	13.60	23.05	14.88	16.71	22.18	358.625	298.715	37.5
	249599	25.24	11.50	12.91	21.60	16.18	18.00	23.50	293.366	240.549	35.88
	199598	28.44	11.57	12.68	20.85	17.49	19.34	24.85	246.087	198.501	34.6
	149598	31.58	11.69	12.45	20.11	18.93	20.76	26.26	194.981	154.500	33
	99597	33.99	13.01	13.75	21.27	21.06	22.92	28.27	139.092	107.216	30.91
	64597	32.18	14.85	15.45	23.48	21.93	23.44	28.46	96.283	73.717	28.33
	44597	28.95	15.79	16.31	24.63	21.45	22.62	27.20	69.934	54.115	26.32
TranslEll_100x100	10000	33.72	37.77	32.39	45.51	35.62	33.21	36.72	12.699	8.482	47.97
	5000	48.19	29.81	26.57	35.67	39.64	39.76	44.76	7.254	4.370	48.18
	2000	55.48	23.55	22.59	31.06	40.69	42.63	48.45	3.401	1.951	47.62
	1000	39.25	22.85	21.36	29.17	32.04	32.66	36.51	1.828	1.231	44.15
	500	25.68	24.01	22.46	31.75	25.00	24.62	27.16	0.983	0.741	42.22
	250	14.79	24.56	17.72	26.85	18.55	15.68	17.55	0.523	0.441	39.47
TranslRotEll_100x100	10000	36.61	34.35	30.52	41.08	35.53	34.16	37.87	13.152	8.659	48.03
	5000	54.44	22.95	19.56	26.79	39.93	40.84	46.64	7.434	4.398	48.1
	2000	47.93	23.34	22.01	30.16	37.33	38.52	43.15	3.367	2.070	44.19
	1000	34.91	25.29	23.45	30.60	30.87	30.97	33.83	1.834	1.266	42.13
	500	23.37	29.14	28.97	37.21	25.69	25.18	26.71	0.997	0.746	39.81
	250	16.71	34.80	33.15	38.84	23.92	21.98	22.09	0.555	0.433	34.98
Trevor_qcif	55320	9.96	11.13	12.15	22.12	10.67	11.11	14.85	67.280	59.808	38.86
	45320	8.69	10.68	11.41	20.69	9.87	10.08	13.38	57.046	51.298	37.92
	35319	7.33	9.66	9.94	18.33	8.67	8.62	11.48	46.205	42.221	36.8
	27819	5.97	9.55	9.68	17.58	7.98	7.75	10.21	37.777	34.850	35.75
	20318	4.79	9.08	8.99	15.80	7.13	6.74	8.66	28.993	27.039	34.46
	15318	3.88	8.65	8.11	14.42	6.42	5.79	7.47	22.776	21.458	33.37
	10317	2.63	8.22	7.29	12.81	5.51	4.65	5.95	16.180	15.428	31.86
	5315	0.78	8.28	7.10	11.47	4.45	3.37	4.07	9.104	8.797	29.42
WashDC_qcif	150320	19.60	13.61	15.45	27.92	15.54	17.12	23.55	154.089	127.711	40.92
	120320	16.98	12.93	14.24	26.26	14.32	15.39	21.21	129.333	109.424	38.88
	100320	15.55	12.48	13.45	25.05	13.57	14.37	19.74	111.706	95.656	37.24
	80320	13.98	11.91	12.47	23.79	12.68	13.16	18.14	93.203	80.938	35.46
	60320	11.91	11.29	11.30	22.17	11.53	11.59	16.06	73.702	65.162	33.45
	40320	9.49	10.52	9.94	20.34	10.09	9.71	13.62	52.622	47.511	31.18
	25320	7.18	9.66	8.32	17.91	8.55	7.72	11.00	35.390	32.659	29.26
	15320	5.13	9.13	7.38	16.24	7.25	6.13	8.84	22.807	21.408	27.72

Table 45: Compression result comparison between AT3D and AT3D_PAQ. Total results and PSNR are only given for quantization `q4`. For more detailed results see appended disc.

D Detailed H.264 results

Compression results											
Sequence	PSNR [dB]	Total size [kb] / Bits per pixel								AT3D_PAQ vs. H.264 [%]	
		AT3D		AT3D_PAQ		H.264 Regular		H.264 Best		Regular	Best
`City-10`	42.60	83.025	2.621	75.438	2.381	28.146	0.888	25.335	0.800	-168.02	-197.76
	38.30	55.086	1.739	51.070	1.612	14.601	0.461	12.694	0.401	-249.77	-302.32
	35.50	38.016	1.200	35.774	1.129	9.154	0.289	8.286	0.262	-290.80	-331.74
	32.90	23.424	0.739	22.238	0.702	6.190	0.195	5.450	0.172	-259.26	-308.04
	30.80	14.490	0.457	13.862	0.438	4.301	0.136	3.892	0.123	-222.30	-256.17
	29.50	9.570	0.302	9.214	0.291	3.487	0.110	3.120	0.098	-164.24	-195.32
`RotGrowEll_ 100x100`	48.40	13.735	0.366	8.421	0.225	10.643	0.284	8.480	0.226	20.88	0.70
	44.20	7.468	0.199	4.897	0.131	8.027	0.214	6.442	0.172	38.99	23.98
	38.10	3.297	0.088	2.414	0.064	5.145	0.137	4.003	0.107	53.08	39.70
	33.30	1.844	0.049	1.456	0.039	3.293	0.088	2.800	0.075	55.78	48.00
	30.20	1.027	0.027	0.858	0.023	2.570	0.069	2.272	0.061	66.61	62.24
	27.20	0.568	0.015	0.493	0.013	1.997	0.053	1.926	0.051	75.31	74.40
`SSTB 6355`	41.20	110.803	1.166	90.837	0.956	84.042	0.884	67.384	0.709	-8.09	-34.80
	39.10	83.147	0.875	69.656	0.733	66.444	0.699	53.254	0.560	-4.83	-30.80
	36.40	57.785	0.608	49.217	0.518	47.125	0.496	36.984	0.389	-4.44	-33.08
	34.10	41.033	0.432	35.411	0.373	33.469	0.352	25.229	0.265	-5.80	-40.36
	32.10	29.233	0.308	25.145	0.265	23.281	0.245	16.894	0.178	-8.01	-48.84
	30.00	19.057	0.201	16.506	0.174	14.463	0.152	9.919	0.104	-14.13	-66.41
`SSTB 101980`	42.9	14.379	0.151	12.270	0.129	10.193	0.107	7.189	0.076	-20.38	-70.68
	40.4	8.128	0.086	7.324	0.077	7.904	0.083	5.648	0.059	7.34	-29.67
	38.1	4.657	0.049	4.375	0.046	6.264	0.066	4.560	0.048	30.16	4.06
	36	2.871	0.030	2.760	0.029	5.056	0.053	3.710	0.039	45.41	25.61
	34	1.765	0.019	1.698	0.018	4.202	0.044	3.093	0.033	59.59	45.10
	32	1.093	0.012	1.040	0.011	3.435	0.036	2.600	0.027	69.72	60.00
`Suzie-10`	42.20	20.657	0.652	18.717	0.591	9.590	0.303	7.949	0.251	-95.17	-135.46
	39.60	10.877	0.343	10.083	0.318	5.966	0.188	4.795	0.151	-69.01	-110.28
	36.80	5.089	0.161	4.775	0.151	3.524	0.111	3.010	0.095	-35.50	-58.64
	35.20	3.079	0.097	2.915	0.092	2.653	0.084	2.321	0.073	-9.88	-25.59
	33.80	1.930	0.061	1.824	0.058	2.150	0.068	1.889	0.060	15.16	3.44
	31.40	1.002	0.032	0.947	0.030	1.591	0.050	1.402	0.044	40.48	32.45

Table 46: Detailed H.264 results of full comparison.

Compression results						
Sequence	PSNR [dB]	Total size [kb] / Bits per pixel				Improvement/ deterioration [%]
		AT3D_PAQ		H.264 Regular		
`MissAmerica`	43.57	21.809	0.229	12.217	0.129	-78.51
	40.93	10.575	0.111	6.198	0.065	-70.62
	38.78	5.965	0.063	3.921	0.041	-52.13
	36.54	3.369	0.035	2.727	0.029	-23.54
	34.96	2.182	0.023	2.263	0.024	3.58
	33.00	1.214	0.013	1.961	0.021	38.09
`SSTB 111130`	39.69	20.433	0.215	14.532	0.153	-40.61
	38.03	12.124	0.128	10.097	0.106	-20.08
	36.07	6.761	0.071	6.595	0.069	-2.52
	34.49	3.975	0.042	4.438	0.047	10.43
	33.79	2.756	0.029	3.487	0.037	20.96
	33.00	1.214	0.013	1.961	0.021	38.09
`SSTB 88250`	42.90	79.740	0.839	15.605	0.164	-410.99
	39.06	59.740	0.629	9.310	0.098	-541.68
	36.41	45.773	0.482	6.823	0.072	-570.86
	34.13	33.611	0.354	5.295	0.056	-534.77
	32.32	24.535	0.258	4.367	0.046	-461.83
	30.12	14.099	0.148	3.052	0.032	-361.96
	28.92	9.410	0.099	2.706	0.028	-247.75
`Tempete-20_cif`	40.88	418.737	1.652	237.960	0.939	-75.97
	39.12	368.439	1.454	186.115	0.734	-97.96
	37.24	313.144	1.236	139.933	0.552	-123.78
	35.85	273.366	1.079	112.152	0.443	-143.75
	34.37	231.196	0.912	84.683	0.334	-173.01
	32.72	185.531	0.732	59.639	0.235	-211.09
	30.82	135.956	0.536	39.336	0.155	-245.63
	28.86	92.262	0.364	25.241	0.100	-265.52
`TranslRotEll_100x100`	48.10	4.398	0.117	5.137	0.137	14.39
	44.19	2.070	0.055	3.880	0.103	46.65
	42.13	1.266	0.034	3.347	0.089	62.18
	39.81	0.746	0.020	2.862	0.076	73.93
	34.98	0.433	0.012	1.965	0.052	77.96
`WashDC`	40.90	127.711	1.344	26.825	0.282	-376.09
	38.90	109.424	1.151	20.710	0.218	-428.36
	37.20	95.656	1.006	16.286	0.171	-487.35
	35.50	80.938	0.852	12.945	0.136	-525.25
	33.50	65.162	0.686	9.817	0.103	-563.77
	31.20	47.511	0.500	7.041	0.074	-574.78
	29.30	32.659	0.344	5.157	0.054	-533.29

Table 47: Detailed H.264 results of short comparison.

E Detailed runtime results

	CfgA-32 adaptive thinning									
	Heap construction time				AT6 time			Failed geometry tests		
Sequence	Old [s]	New [s]	Gain [%]	Thinned px [M]	Old [s]	New [s]	Gain [%]	Old [Bil]	New [Bil]	Reduction [%]
City-10	237	130	45.15	0.25	7926	3984	49.74	15.84	5.41	65.87
City-20	520	280	46.15	0.50	17708	9083	48.71	35.96	12.21	66.06
City	780	432	44.62	0.75	28240	13841	50.99	55.78	19.12	65.72
Football	774	439	43.28	0.75	32489	14814	54.40	76.60	20.34	74.67
Galleon	767	432	43.68	0.75	27737	13573	51.07	63.15	19.26	69.50
Highway2-10	236	129	45.34	0.25	10357	4557	56.00	29.53	6.17	79.10
Highway2-20	506	289	42.89	0.50	24998	10582	57.67	82.25	14.81	82.00
Highway2	768	435	43.36	0.75	34013	15796	53.56	89.15	22.58	74.67
MissAmerica	766	438	42.82	0.75	28407	14005	50.70	64.10	19.41	69.72
Soccer	769	440	42.78	0.75	30597	14133	53.81	70.14	19.65	71.98
	Average gain		**44.01**		Average gain		**52.66**	Average gain		**71.81**

Table 48: Adaptive thinning runtime and efficiency comparison of old and new implementation.

	CfgA-32 exchange								
		Exchanged pixels		Exchange time			Failed geometry tests		
Sequence	$\|X_n\|$	Old	New	Old [s]	New [s]	Gain [%]	Old [Bil]	New [Bil]	Reduction [%]
City-10	3440	4304	4116	4216	2489	40.96	48.26	13.63	71.75
City-20	6880	8213	7933	10875	6603	39.28	118.79	36.88	68.96
City	10320	12114	11897	17423	10750	38.30	191.70	60.16	68.62
Football	10320	9844	9527	25043	13934	44.36	258.68	70.57	72.72
Galleon	10320	9015	8591	17423	10397	40.33	189.26	51.96	72.55
Highway2-10	3440	3026	3022	7969	4102	48.53	88.62	17.62	80.11
Highway2-20	6880	5173	5474	18998	9713	48.87	198.15	43.92	77.83
Highway2	10320	8620	8342	28044	13935	50.31	294.45	68.45	76.76
MissAmerica	10320	7585	7722	16510	9333	43.47	163.97	46.76	71.48
Soccer	10320	9383	9218	22825	13756	39.73	228,00	60.14	73.62
		Average gain				**43.41**	Average gain		**73.44**

Table 49: Exchange runtime and efficiency comparison of old and new implementation. $|X_n|$ indicates the amount of significant pixels in the grid.

CfgB-64 adaptive thinning										
Sequence	Heap construction time			AT6 time				Failed geometry tests		
	Old [s]	New [s]	Gain [%]	Thinned px [M]	Old [s]	New [s]	Gain [%]	Old [Bil]	New [Bil]	Reduction [%]
City-10	92	69	25.00	0.25	3081	1956	36.51	15.62	5.40	65.34
City-20	201	148	26.37	0.50	6977	4413	36.75	35.56	12.23	65.61
City	302	227	24.83	0.75	10853	6891	36.51	55.78	19.10	65.76
Football	304	228	25.00	0.75	12019	7197	40.12	76.40	20.29	73.44
Galleon	302	228	24.50	0.75	10660	6704	37.11	63.42	19.23	69.68
Highway2-10	92	68	26.09	0.25	3914	2202	43.74	28.27	6.16	78.22
Highway2-20	198	147	25.76	0.50	9001	4928	45.25	76.50	14.82	80.63
Highway2	303	227	25.08	0.75	12886	7584	41.15	90.76	22.47	73.44
MissAmerica	307	228	25.73	0.75	10922	6761	38.10	63.94	19.46	69.57
Soccer	307	228	25.73	0.75	11434	6848	40.11	70.05	19.66	71.94
City-10_cif	373	277	25.74	1.00	12556	7907	37.03	64.69	21.82	66.26
Suzie-90*	966	725	24.95	2.25	34552	21230	38.56	195.18	61.03	68.73
Tempete-20_cif	802	598	25.44	1.975	27361	17474	36.14	133.55	46.89	64.89
Tennis-20_sif	667	495	25.79	1.65	27643	15654	43.47	187.41	46.11	75.40
SSTB28053-6_4sif	816	604	25.98	2.30	40582	21513	46.99	310.01	55.69	82.04
Average gain			**25.47**	Average gain			**39.83**	Average gain		**71.52**

Table 50: Adaptive thinning runtime and efficiency comparison of old and new implementation.

*: Due to a crash this sequence was encoded with randomized point insertion, which increases absolute compression times, but relative results remain valid.

CfgB-64 exchange									
Sequence		Exchanged pixels		Exchange time			Failed geometry tests		
	$\lvert X_n \rvert$	Old	New	Old [s]	New [s]	Gain [%]	Old [Bil]	New [Bil]	Reduction [%]
City-10	3440	4096	3979	1301	835	35.82	41.77	12.83	69.29
City-20	6880	8207	8134	3577	2377	33.55	118.02	39.00	66.95
City	10320	12327	12100	6019	3868	35.74	190.56	64.04	66.39
Football	10320	9484	9828	7271	5220	28.21	230.31	73.50	68.09
Galleon	10320	8962	8685	5768	3665	36.46	175.94	50.59	71.25
Highway2-10	3440	2971	2917	2868	1480	48.40	82.34	18.41	77.64
Highway2-20	6880	5616	5636	6787	3908	42.42	226.59	45.20	80.05
Highway2	10320	8398	8314	8550	5375	37.13	295.54	71.89	75.66
MissAmerica	10320	7610	7517	5285	3200	39.45	166.56	44.78	73.11
Soccer	10320	9396	9185	7557	4627	38.77	234.04	55.94	76.1
City-10_cif	13760	15120	14805	5341	3455	35.31	185.70	55.34	70.20
Suzie-90	30960	28260	28914	18898	12578	33.44	592.23	194.04	67.24
Tempete-20_cif	52520	41458	40536	18863	12489	33.79	472.24	166.48	64.75
Tennis-20_sif	39600	19797	19809	22370	14723	34.18	675.28	170.35	74.77
SSTB-28053-6_4sif	65440	44822	44372	40056	23498	41.34	1170.43	239.40	79.55
Average gain						**36.93**	Average gain		**72.07**

Table 51: Exchange runtime and efficiency comparison of old and new implementation. $|X_n|$ indicates the amount of significant pixels in the grid.

References

[Ahm74] N. Ahmed, T. Natarajan, K.R. Rao. ***Discrete Cosine Transform.*** IEEE Transactions on Computers C-23 (1): 90-93, 1974.

[Ber13] W. Bergmans. ***Data compression benchmarks, http://www.maximumcompression.com/index.html***. Accessed 2013.

[Bla02] C. Blatter. ***Wavelets: a Primer.*** A K Peters Ltd., 2002.

[Che05] Z. Chen. ***Finite element methods and their applications.*** Springer, 2005.

[Cov91] T. M. Cover, J.A. Thomas, J. Kieffer. ***Elements of Information Theory.*** Wiley, New York, 1991.

[deB08] M. de Berg, O. Cheong, M. van Kreveld, M. Overmars. ***Computational Geometry: Algorithms and Applications.*** Springer, 3rd edition, 2008.

[Dem06] L. Demaret,A. Iske. ***Adaptive Image Approximation by Linear Splines over Locally Optimal Delaunay Triangulations.*** IEEE Signal Processing Letters 13(5), 2006, 281-284.

[Dem11] L. Demaret, A. Iske, W. Khachabi. ***Sparse Representation of Video Data by Adaptive Tetrahedralizations***. 2011.

[Der13] ***Test video sequences, http://media.xiph.org/video/derf/***, viewed 2013.

[Dev02] O. Devillers, S. Pion, M. Teillaud. ***Walking in a triangulation.*** International Journal on Foundations of Computer Science, 13:181-199, 2002.

[Enc13] ***Encoding forum, http://encode.ru/forum.php***, viewed 2013.

[Fed92] M. Feder, N. Merhav, M. Gutman. ***Universal prediction of individual sequences.*** Information Theory, IEEE Transactions on 38.4: 1258-1270, 1992.

[GCC13] GCC-Team. ***Options that control optimization, http://gcc.gnu.org/onlinedocs/gcc/Optimize-Options.html***. 2013.

[Hin10] M. Hinnenthal. ***Erweiterung der kontextuellen Kompressionsmethoden mit Hilfe von Delaunay Triangulierungen***. Diploma thesis at TU Muenchen, 2010.

[ITU11] International Telecommunications Union. ***ITU-R Recommendation BT.601, http://www.itu.int/rec/R-REC-BT.601-7-201103-I/en***. 2011.

[ITU02] International Telecommunications Union. ***ITU-R Recommendation BT.709, http://www.itu.int/rec/R-REC-BT.709/en***. 2002.

[Kha11] W. Khachabi. ***Introducing a novel approach in compression of digital video sequences.*** Preprint, Uni Hamburg, 2011.

[Kno12] B. Knoll, N. de Freitas. ***A machine learning perspective on predictive coding with PAQ8.*** Data Compression Conference, pages 377-386, 2012.

[Mah02] M. V. Mahoney, ***The PAQ1 data compression program, http://mattmahoney.net/dc/paq1.pdf.*** Draft, 2002.

[Mah05] M. V. Mahoney. ***Adaptive weighing of context models for lossless data compression.*** Florida Tech., Melbourne, USA, Tech. Rep, 2005.

[Mah13] M. V. Mahoney. ***Data compression benchmarks. http://mattmahoney.net/dc/#benchmarks***. Accessed 2013.

[Mat12] C. Mattern. ***Mixing Strategies in Data Compression.*** Proc. of the 22nd Data Compression Conference (DCC), pp. 337-346, 2012.

[Mat13] C. Mattern. ***Linear and Geometric Mixtures - Analysis.*** DCC: 301-310, 2013.

[Pal08] N. Paley. ***Sita sings the blues, 480p uncompressed***. http://media.xiph.org/video/derf/y4m/sita_sings_the_blues_480p24.y4m.xz, 2008, viewed 2013. Author's page available at http://www.sitasingstheblues.com.

[Pur04] Puri, Atul, X. Chen, A. Luthra. ***Video coding using the H. 264/MPEG-4 AVC compression standard.*** Signal processing: Image communication 19.9, 793-849, 2004.

[Roj96] R. Rojas. ***Neutral Networks: A Systematic Introduction.*** Springer, 1996.

[Say00] K. Sayood. ***Introduction to Data Compression.*** Academic Press, 2nd Edition, San Diego, CA, 2000.

[Sou11] R. Soukal, et al. ***Hybrid Walking Point Location Algorithm.***, ADVCOMP 2011, The Fifth International Conference on Advanced Engineering Computing and Applications in Sciences, 2011.

[Sul05] Sullivan, Gary J., and Thomas Wiegand. ***Video compression-from concepts to the H. 264/AVC standard.*** Proceedings of the IEEE 93.1, 18-31, 2005.

[Wan03] Z. Wang, H. R. Sheikh, A. C. Bovic. ***Objective Video Quality Assessment.*** Chapter 41 in The Handbook of Video Databases: Design and Applications, B. Furht and O. Marqure, ed., CRC Press, pp. 1041-1078, September 2003.

[Wel98] F. Weller. ***On the Total Correctness of Lawson's Oriented Walk Algorithm.*** The 10th International Canadian Conference on Computational Geometry, Montral, Quebec, 1998.

[Zin03] M. Zinkevich. ***Online convex programming and generalized infinitesimal gradient ascent.*** In ICML, pages 928-936, 2003.

List of Figures

1 Overview of the H264 encoding process. 15
2 Arithmetic coding example. 31
3 Exemplary Delaunay tetrahedralization. 39
4 Schematic display of a restricted 2d context box. 49
5 Schematic display of context box projection. 50
6 Schematic high-level description of the architecture of PAQ8. 56
7 Schematic high-level description of r models in a model class. 60
8 Schematic display of the NN-mixer. 65
9 Schematic display of the original software architecture in PAQ. 69
10 Schematic display of the merged software architecture in AT3D_PAQ. . . . 70
11 Visualization of position-specific context box model. 74
12 Visualization of cascaded SSE-stage of pixel position modeling. 74
13 First frames of `Suzie`, `MissAmerica`, and `SSTB94115`. 81
14 Typical distribution of significant pixels among frames for low-entropy sequences. 82
15 Pixel position encoding improvement of AT3D_PAQ vs. AT3D on `Suzie`. . 82
16 Significant pixel distribution in `SSTB94115` with 5319 significant pixels. . . . 82
17 Pixel position encoding improvement of AT3D_PAQ vs. AT3D on `SSTB94115`. 82
18 First frame and significant pixels in frames 0, 9 and 19 in `TranslEll_100x100`. 83
19 Significant pixel distribution in `TranslEll_100x100` with 2000 sign. pixels. 83
20 Pixel position encoding improvement of AT3D_PAQ vs. AT3D at `TranslEll_100x100`. 83
21 Comparison of bits per pixel vs. PSNR of old and new implementation at `q4`. 86
22 Improvement of AT3D_PAQ vs. AT3D on `Highway2`. 86
23 Comparison of bits per pixel vs. PSNR of old and new implementation at `q4`. 87
24 Improvement of AT3D_PAQ vs. AT3D on `SSTB1450`. 87
25 Comparison of bits per pixel vs. PSNR of old and new implementation at `q4`. 88
26 Improvement of AT3D_PAQ vs. AT3D on `RotGrowBox-120_50x50`. 88
27 Comparison of bits per pixel vs. PSNR of old and new implementation at `q4`. 89
28 Improvement of AT3D_PAQ vs. AT3D on `Tennis-20_sif`. 89
29 bpp vs. PSNR of AT3D, AT3D_PAQ vs. H.264 on `Suzie-10`. 92
30 Improvement/deterioration comparison of AT3D_PAQ and H.264 at `Suzie-10`. 92
31 bpp vs. PSNR of AT3D, AT3D_PAQ vs. H.264 on `City-10`. 93

32 Improvement/deterioration comparison of AT3D_PAQ and H.264 at `City-10`. . . . 93

33 bpp vs. PSNR of AT3D, AT3D_PAQ vs. H.264 on `SSTB6355`. . . . 94

34 Improvement/deterioration comparison of AT3D_PAQ and H.264 at `SSTB6355`. . . . 94

35 bpp vs. PSNR of AT3D, AT3D_PAQ vs. H.264 on `SSTB101980`. . . . 95

36 Improvement/deterioration comparison of AT3D_PAQ and H.264 at `SSTB101980`. . . . 95

37 bpp vs. PSNR of AT3D, AT3D_PAQ vs. H.264 on `RotGrowEll__100x100`. 96

38 Improvement/deterioration comparison of AT3D_PAQ and H.264 at `RotGrowEll__100x100`. . . . 96

39 bpp vs. PSNR of AT3D_PAQ vs. H.264 on `WashDC`. . . . 97

40 bpp vs. PSNR of AT3D_PAQ vs. H.264 on `SSTB88250`. . . . 97

41 bpp vs. PSNR of AT3D_PAQ vs. H.264 on `MissAmerica`. . . . 97

42 bpp vs. PSNR of AT3D_PAQ vs. H.264 on `SSTB111130`. . . . 97

43 bpp vs. PSNR of AT3D_PAQ vs. H.264 on `Tempete-20__cif`. . . . 97

44 bpp vs. PSNR of AT3D_PAQ vs. H.264 on `TranslRotEll__100x100`. . . . 97

45 Minimal bounding box $\mathfrak{B}_{T_j}$ of a single tetrahedron T_j. . . . 101

46 Visualization of proof idea of lemma 4.1.5. . . . 103

47 Minimal bounding box $\mathfrak{B}_{\mathbb{T}}$ of a set $\mathbb{T}$ of two tetrahedra. . . . 111

48 Schematic representation of old pixel scan implementation. . . . 112

49 Schematic representation of new pixel scan implementation. . . . 112

50 Schematic display of a ring buffer. . . . 118

51 Progression of compression time for AT6 applied to `City`. . . . 126

52 Progression of failed geometry tests in AT6 applied to `City`. . . . 126

53 Progression of compression time for AT6 applied to `Highway2`. . . . 126

54 Progression of failed geometry tests in AT6 applied to `Highway2`. . . . 126

List of Tables

1 Overview of AT3D stages. . . . 36
2 Stationarity investigation: Matching n-grams in `Suzie-90`. . . . 54
3 Stationarity investigation: Matching n-grams in `Football`. . . . 54
4 Compression results: Overall average grayscale value encoding sorted by encoding stage. . . . 80
5 Compression results: Overall average grayscale value encoding sorted by quantization setting. . . . 81
6 Significant pixels in `Suzie` and `SST94115`. . . . 82
7 Pixel position coding cost visualization: `Suzie`. . . . 84
8 Pixel position coding cost visualization: `RotEll__100x100`. . . . 84
9 Pixel position coding cost visualization: `Football`. . . . 84
10 Compression result comparison: AT3D vs. AT3D_PAQ, `Highway2`. . . . 86
11 Frame comparison: `Highway2`. . . . 86
12 Compression result comparison: AT3D vs. AT3D_PAQ, `SSTB1450`. . . . 87
13 Frames and significant pixels: `SSTB1450`. . . . 87
14 Compression result comparison: AT3D vs. AT3D_PAQ, `RotGrowBox-120`. . . . 88
15 Frame comparison: `RotGrowBox-120__50x50`. . . . 88
16 Compression result comparison: AT3D vs. AT3D_PAQ, `Tennis-20__sif`. . . . 89
17 Frames and significant pixels: `Tennis-20__sif`. . . . 89
18 Compression time comparison: AT3D vs. AT3D_PAQ. . . . 91
19 Compression result comparison: AT3D_PAQ vs. H.264, `Suzie-10`. . . . 92
20 Frame comparison: `Suzie-10`. . . . 92
21 Compression result comparison: AT3D_PAQ vs. H.264, `City-10`. . . . 93
22 Frame comparison: `City-10`. . . . 93
23 Compression result comparison: AT3D_PAQ vs. H.264, `SSTB6355`. . . . 94
24 Frame comparison: `SSTB6355`. . . . 94
25 Compression result comparison: AT3D_PAQ vs. H.264, `SSTB101980`. . . . 95
26 Frame comparison: `SSTB101980`. . . . 95
27 Compression result comparison: AT3D_PAQ vs. H.264, `RotGrowEll`. . . . 96
28 Frame comparison: `RotGrowEll__100x100`. . . . 96
29 Run-time comparison: Standard vs. custom queue implementation. . . . 119
30 Run-time comparison: Configuration A, Ba-32 vs. i686-32. . . . 123
31 Run-time comparison: Configuration A, Ba-64 vs. i686-64. . . . 123
32 Run-time comparison: Configuration B, Co-32 vs. i686-32. . . . 123
33 Run-time comparison: Configuration B, Co-64 vs. i686-64. . . . 123

34 Relative run-time comparison of different compiler-optimizations and environments on configuration A. 123
35 Relative run-time comparison of different compiler-optimizations and environments on configuration B. 123
36 Run-time results: Average best-case improvements. 125
37 Run-time results: Average worst-case improvements. 125
38 Overview: Test sequences from natural and artifical sequences. 130
39 Overview: Test sequences from `Sita sings the blues`. 131
40 Overview of video resolutions employed in this thesis. 131
41 Overview of test sequences 132
42 Detailed compression result comparison of AT3D and AT3D_PAQ: A-H. . . 133
43 Detailed compression result comparison of AT3D and AT3D_PAQ: H-SSTB. 134
44 Detailed compression result comparison of AT3D and AT3D_PAQ: SSTB-Su. 135
45 Detailed compression result comparison of AT3D and AT3D_PAQ: Su-Z. . 136
46 Detailed H.264 results of full comparison. 137
47 Detailed H.264 results of short comparison. 138
48 Configuration A: Adaptive thinning runtime and efficiency comparison of old and new implementation. 139
49 Configuration A: Exchange runtime and efficiency comparison of old and new implementation. 139
50 Configuration B: Adaptive thinning runtime and efficiency comparison of old and new implementation. 140
51 Configuration B: Exchange runtime and efficiency comparison of old and new implementation. 140