Institute for Visualization and Interactive Systems

University of Stuttgart
Universitätsstraße 38
D–70569 Stuttgart

Masterarbeit Nr. 1025002

Progressive Sparse Coding for In Situ Volume Visualization

Grațian Berian

Course of Study:	Informatik
Examiner:	Prof. Dr. Thomas Ertl Associate Prof. Dr. Markus Hadwiger
Supervisors:	Dr. Guido Reina, Dr. Steffen Frey, Dr. Peter Rautek
Commenced:	19. Juli 2016
Completed:	18. Januar 2017
CR-Classification:	I.7.2

Abstract

Nowadays High-Performance Computing (HPC) suffer from an ever-growing gap between computational power, I/O bandwidth and storage capacity. Typical runs of HPC simulations produce Terabytes of data every day. This poses a serious problem when it comes to storing and manipulating such high amount of data.

In this thesis I will present a method for compressing time-dependent volume data using an overcomplete dictionary learned from the input data. The proposed method comprises of two steps. In the first step the dictionary is learned over a number of training examples extracted from the volume that we want to compress. This process is an iterative one and at each step the dictionary is updated to better sparsely represent the training data. The second step expresses each block of the volume as a sparse linear combination of the dictionary atoms that were trained over that volume.

In order to establish the performance of the proposed method different aspects were tested such as: training speed vs sparsifying speed, compression ratio vs reconstruction error, dictionary reusabilty for multiple time steps and how does a dictionary perform when it is used on a different volume than the one it was trained on.

Finally we compare the quality of the reconstructed volume to the original volume and other lossy compression techniques in order to have a visual understanding about the quality of the reconstruction.

Contents

List of Figures

List of Tables

1 Introduction

1.1 History

When we speak about visualization scientific data most people think that this is a field that has its roots in our modern way of representing and analyzing statistical data. In fact visualization can be traced to the earliest forms of maps [Fri06]. One of the earliest maps we have is a 10^{th} century table sketch that plots the inclination of the most prominent stars over the night sky. On the horizontal axes time is divided into 30 intervals and on the vertical axes represents the inclination of the orbit. This idea of coordinates systems comes from the ancient Egyptian for laying down towns and it goes back to 200 BC.

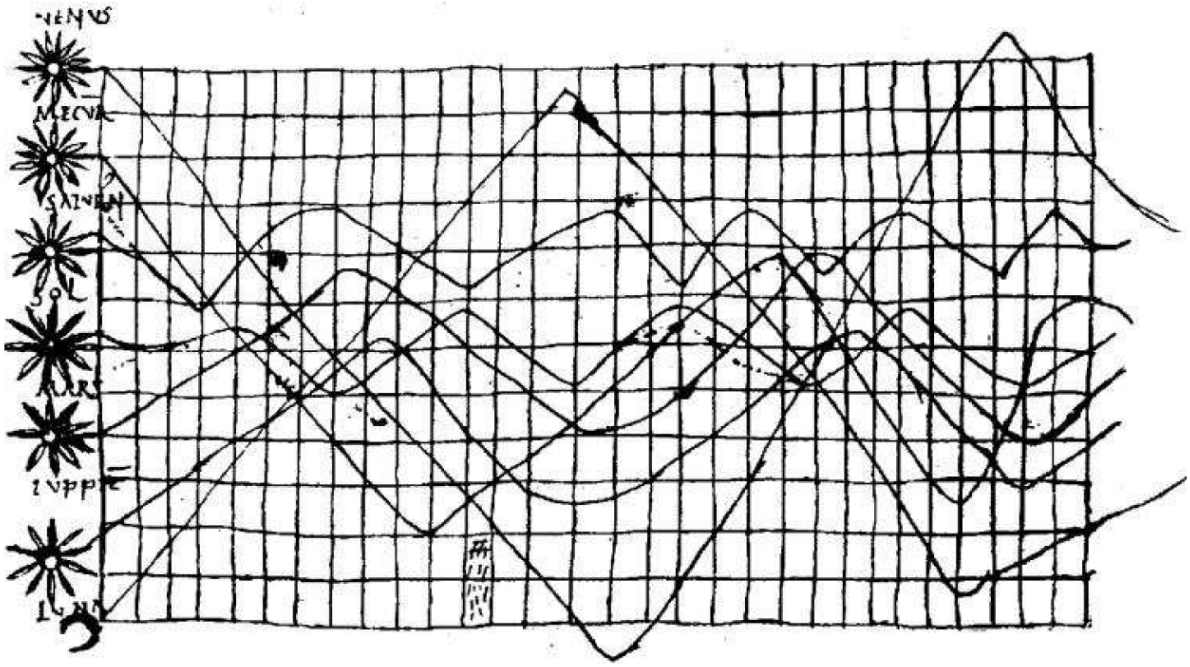

Figure 1.1: Movement of planets over time, by an unknown astronomer.[Fun36]

With time the maps got more refine and included more data and new graphic forms. In cartography new graphic representations like isolines and contours were invented in order to convey more information about the subject in question.

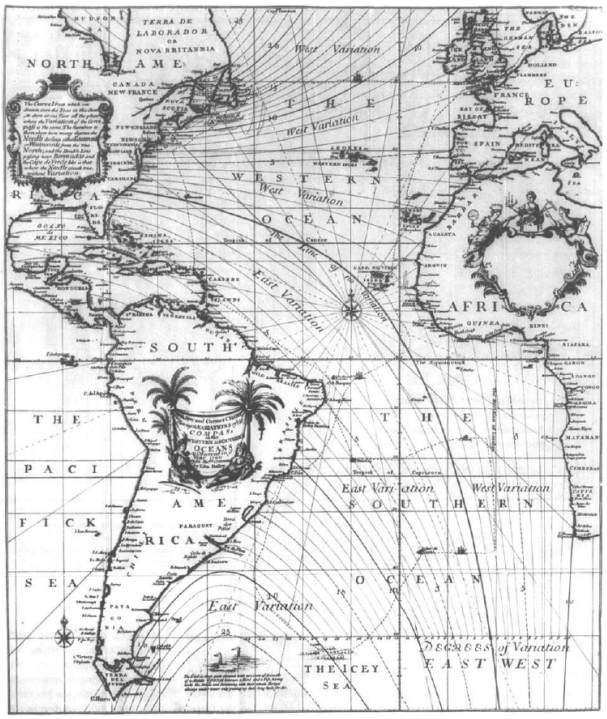

Figure 1.2: A portion of Edmund Halley's New and Correct Sea Chart Shewing the Variations in the Compass in the Western and Southern Ocean, 1701. Source: Halley, E. (1701), image from [Pal]

For example in Fig.1.2 the isolines represent equal magnetic declination. With time thematic mapping has extended to other fields such as economy, politics and medicine, incorporating more and more data and representing it in a visually comprehensible and pleasing way.

A good example of this synthesis of different data into a single graph representation is Fig.1.3. In this figure there are different features plotted of Napoleon's Russian campaign in 1812. The thickness of the of the lines represent the size of the army, branches in the lines represent numbers of soldiers that have separated from the main army at different moments in time. On the horizontal axis there is a time line coupled with temperatures and location. Another information is stored as color and represents the direction in which the army is advancing or retreating.

This is a good example on how to design and convey information in a visually pleasant and engaging way by encoding information within color, thickness position and location.

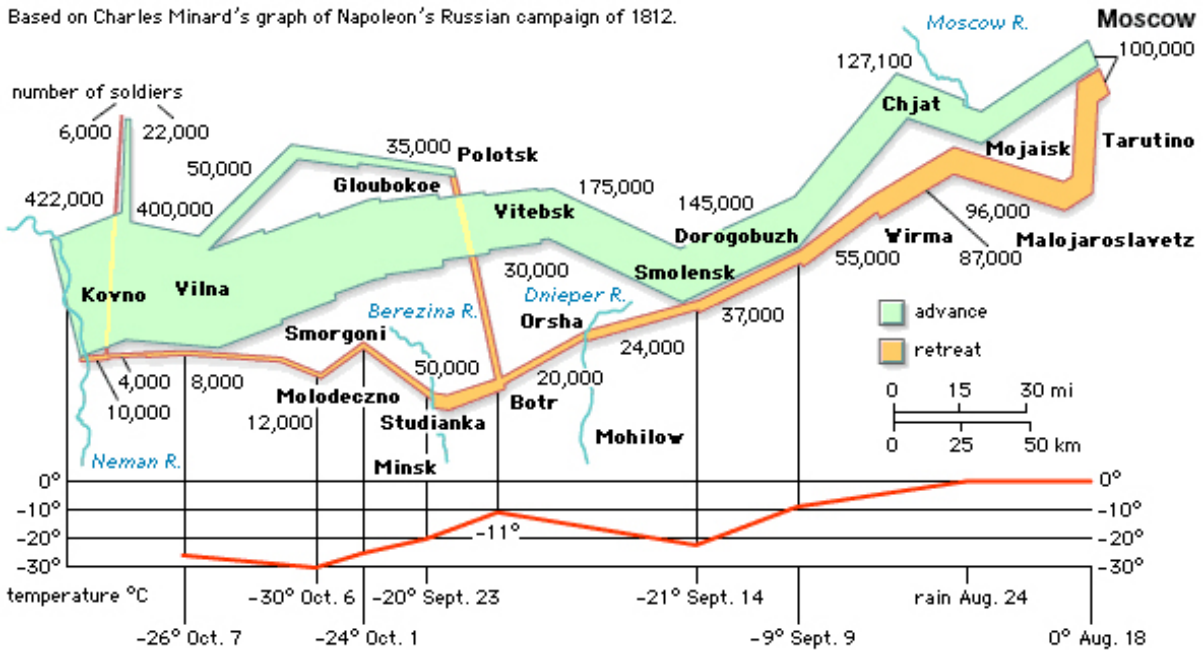

Figure 1.3: Graph of Napoleon's Russian campaign of 1812.

1.2 Present advances and challenges

Visualizing data is an important tool that we have today at our disposal to better understand phenomena and get an insight into complex data. We use visualization in field like chemistry or biology (CT scans) to identify different conditions or potential harmful cells like cancerous cells. In engineering we conduct simulation and then visualize them in order to make cares safer and more reliable or in meteorology to foresee the weather.

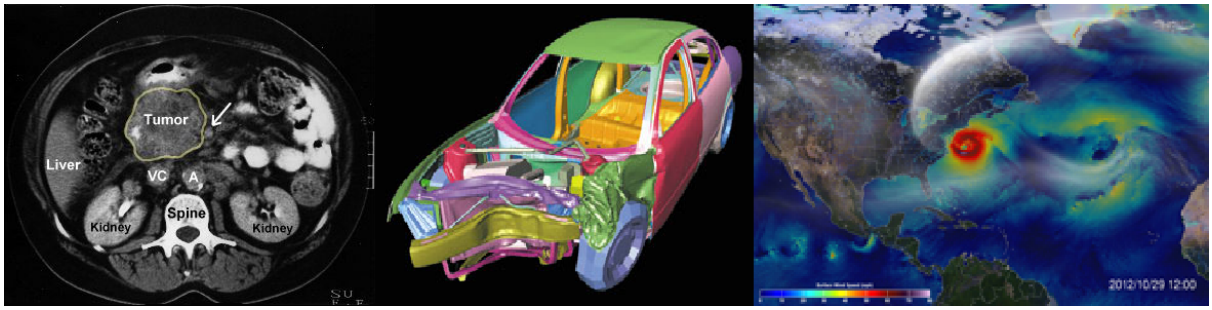

Figure 1.4: Visualization of a CT scan, car crash and a hurricane

As technology advances the computational power that is available is growing almost exponentially so we can achieve more accurate simulations, we can visualize

larger data sets at interactive speeds and achieve photorealism with our reconstructions. At the same time our tools for acquiring data are getting more precise and can scan larger volumes in the same time interval.

For example electron microscopes can produce scan images that are between 10-40 megapixels per second and have a pixel resolution in the range of 3-5 nm. All of this sums up to almost a Terabyte of data per day, and with the new multi-beam electron microscope the amount of raw data that will be available will increase by 1 or 2 orders of magnitude.

All this amount of data creates two distinct problems. One has to do with the amount of variables that need to be depicted in a visualization and the other has to do with the size that the data set requires to be stored. For the first problem there have been developed techniques for representing multiple attributes of a data set as a symbol(glyph). This has the advantage of carrying all the the information needed for a specific point in the data set but the amount of symbols that can be placed without inducing clutter is lower than the amount of data samples.

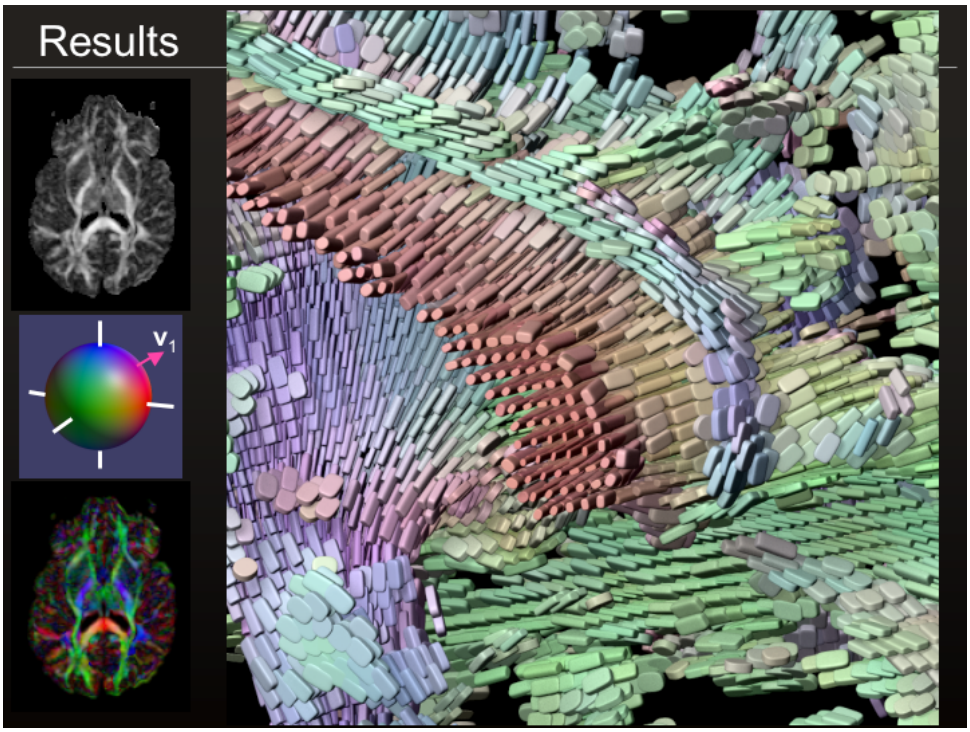

Figure 1.5: Superquadric Tensor Glyphs[Kin04]

In Fig.1.5 we can see a glyph representation of a diffusion tensor field of the human brain. The glyphs encode the tensor field information and the color corresponds to the dominant eigenvector orientation.

The second problem has to do with the gap between local memory of a machine (RAM) and the hard drive storage. There is 1 or 2 orders of magnitude between the RAM capacity and HDD capacity. In order to solve this issue methods like data streaming and out-of-core methods have been developed to enable interactive visualization of large volume data. Also the speed gap at which data is accessed from local memory and permanent memory has grown over the years.

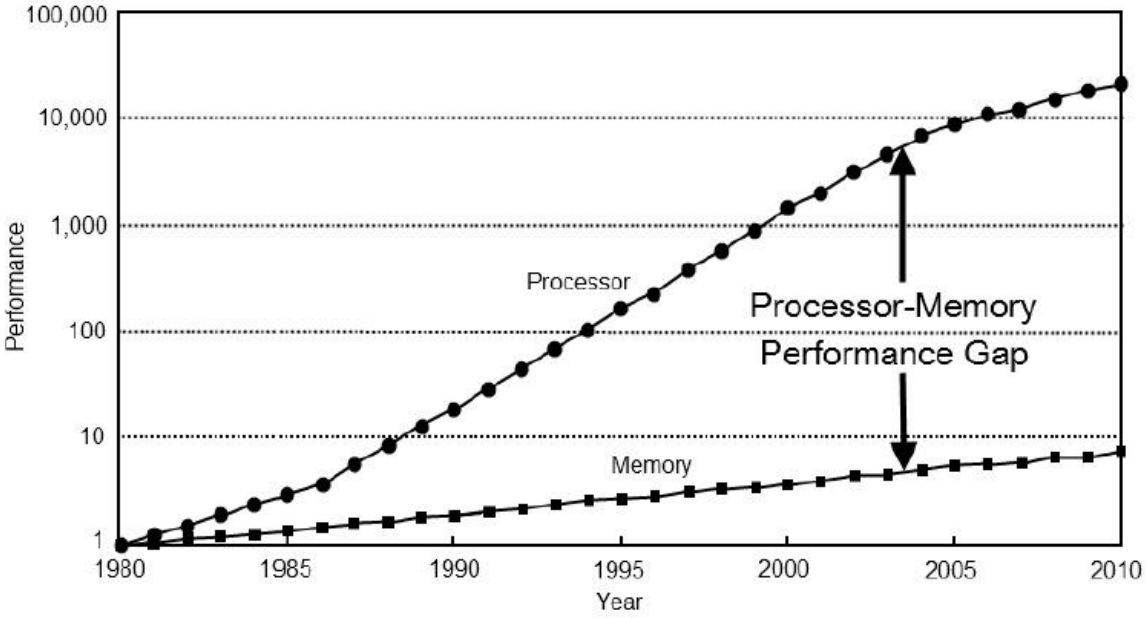

Figure 1.6: CPU/Memory performance[JLH11]

As depicted in 1.6 the performance penalty is huge when we access memory that is outside the cash memory and needs to be fetched from RAM. A similar performance gap is between RAM and HDD access speed, since recently most of permanent data was stored on mechanical disks which are very slowed in terms of latency when compared to RAM.

Ideally we would like to have all the data that is necessary in local memory and do as little I/O operations as possible. In order to achieve this goal there have been developed multiple compression techniques to reduce the data footprint in memory.

1.3 Goals of the Thesis

The focus of this thesis is to develop a method for compressing volumetric data that come from High-Performance Computing(HPC). The goal is to reduce the storage requirements for large-scale time dependent volumes that can be produced by HPC computation runs.

Given the low cost of computation with respect to I/O operations we will concern ourself with the quality of the reconstructed volume and the compression ratio and not with the computational cost. In order to achieve a sparse representation of the data set we will have to train a dictionary that is specific to the volume that will be compressed and then sparsify the volume with respect to the learned dictionary.

The dictionary will start as an overcomplete $DiscreteCosineTransform$(DCT) and the it will be trained over a set of samples that are taken from the volume. This is done in multiple iterations each time with the goal of improving the sparsity of the training data with respect to the previous dictionary. Considering that the resulting overcomplete dictionary will need to be saved in memory the final dictionary will be stored as a sparse representation of the original overcomplete DCT dictionary. By doing so we would just need to multiply the base dictionary with the sparse matrix representation in order to retrieve the learned dictionary when we reconstruct the volume and thus the space needed for storing it is greatly reduced.

After the dictionary is trained on the volume the sparsifying step is next. In this part of the process the volume is decomposed into blocks that have the size of the dictionary atoms. Each block will be reconstructed as a combination of dictionary atoms with a preset tolerance. This step is known as vector quantization. Setting the error tolerance gives us a trade off between compression and fidelity to the original input data.

Finally the sparse dictionary and the information needed for reconstruction are put together and written to the HDD to be later reconstructed and visualized.

2 Related Work

2.1 Sparse Coding

Sparse Coding is a method for finding the representation of an input vector over a basis to represent data efficiently. The set of basis vectors form a dictionary which can be overcomplete, meaning that the number of basis vectors is larger than the dimension of the signal that needs to be represented.

$$\min_x \left\|\mathbf{x}\right\|_0 \text{ subject to } \mathbf{y} = \mathbf{D}\mathbf{x}, \tag{2.1}$$

where $\mathbf{x}$ represents the coefficients of the dictionary atoms that reconstruct the input signal $\mathbf{y}$. The dictionary is a matrix in which the columns are basis vectors and are denoted by $\mathbf{d}_j$. The length of the input signal is equal to the length of the dictionary atoms.

When an approximate solution is sufficient the minimization becomes

$$\min_x \left\|\mathbf{x}\right\|_0 \text{ subject to } \left\|\mathbf{y} - \mathbf{D}\mathbf{x}\right\|_2 \leq \epsilon, \tag{2.2}$$

by setting ϵ we can determine how close we want to reconstruct the input signal to the original one.

The minimization is usually done by a pursuit algorithm. Finding the minimal amount of atoms to reconstruct the input data is an NP hard problem so most algorithms are greedy. The solutions that are generated are feasible but there is no guarantee with respect to optimality.

The next sections will focus on how to choose a dictionary that is suitable for our needs and what minimization techniques we have at our disposal that offer a feasible decomposition.

2.2 Choosing the right Dictionary

There is a wide variety of dictionaries to choose from that have been developed over the years. Just to name some wavelets[Mal99], curvelets[CD99], contourlets[DV05], discrete cosine transform, Fourier transforms and more. The quality of a dictionary is tightly related to how good it can sparsify the input signal. So for different signals different dictionaries perform better.

Wavelet-based dictionaries are very popular in image compression and can approximate an input image with few coefficients. The discrete cosine transform (DCT) has been used with noticeable success in audio and image compression and is one of the most popular base used. Currently DCT is used in different image and video compression standards like JPEG and MPEG.

Furthermore a dictionary can be overcomplete. This implies that there are multiple ways of representing an input signal. Also a dictionary can be generic like DCT or wavelet, meaning it can be described by a mathematical formula and have fast ways of evaluating. Other dictionaries are learned from a set of input signals in order to better fit the data and produce sparser results.

In the next sections we will take a look at some of the most popular dictionaries.

2.2.1 Discrete Cosine Transform

The DCT is comparable to the discrete Fourier transform (DFT) the difference being that DCT works only in the real domain. The input signal is expressed as a linear combination of cosine functions that have different frequency and amplitudes.

Due to boundary conditions there are eight standard DCT variants. The most common one is the type-II DCT [ANR74] and is defined as follows

$$\mathbf{X}_k = \sum_{n=0}^{N-1} x_n cos[\frac{\pi}{N}(n+\frac{1}{2})k] \text{ where k = 0,..., N-1} \tag{2.3}$$

where N is the length of the input signal, x_n is the n^{th} sample point of the signal and $\mathbf{X}_k$ is the K^{th} coefficient of the transform.

In this way we can express a 1D signal as a sum of cosine functions, but the DCT is not limited just to 1D input data. There are multidimensional variants of the DCT that can express input data of higher dimensions. Those transforms are simply an separable product of DCTs along each dimension.

$$\mathbf{X}_{k,l} = \sum_{n_1=0}^{N_1-1} \sum_{n_2=0}^{N_2-1} x_{n_1,n_2} cos[\frac{\pi}{N_1}(n_1 + \frac{1}{2})k_1] cos[\frac{\pi}{N_2}(n_2 + \frac{1}{2})k_2] \tag{2.4}$$

Applying 2.4 on a 2D input data, like an image, will produce a matrix of coefficients that will reconstruct the initial image when applied to the inverse transform.

An example of two dimensional DCT basis of length eight can be seen in 2.1. Every entry of the dictionary is an 8x8 matrix that represents the dictionary atom. In total there are 64 dictionary atoms. This is used for compression in JPEG.

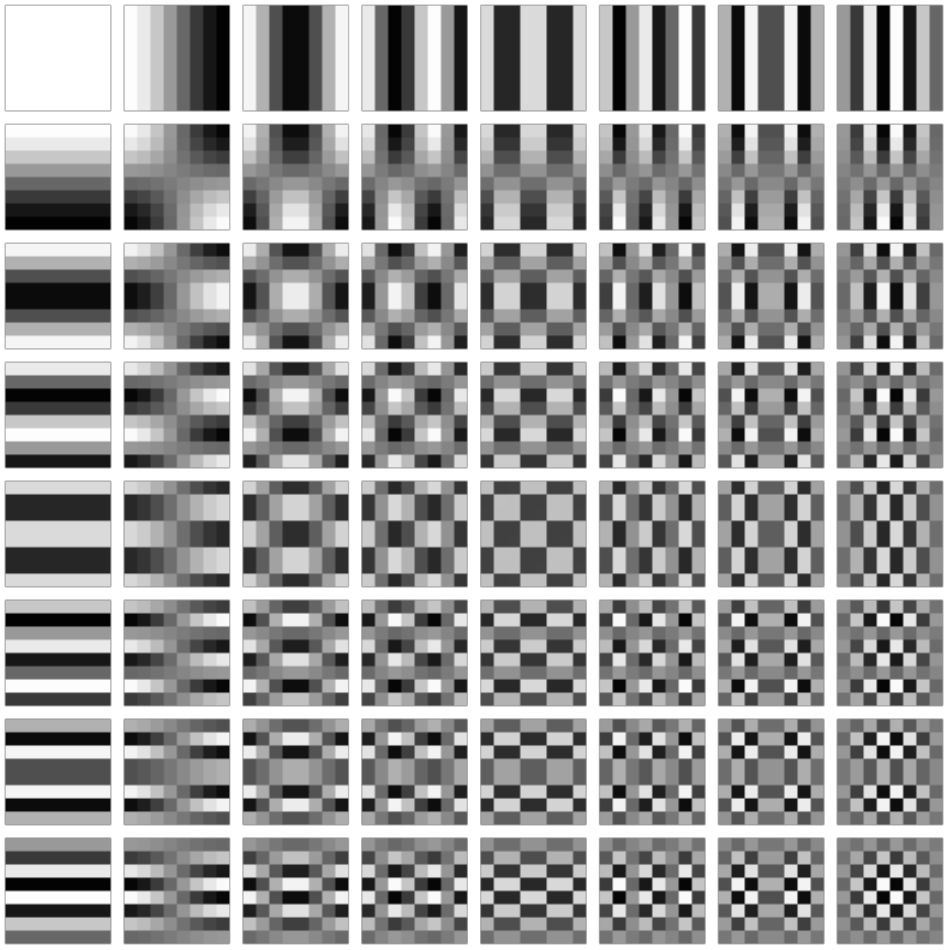

Figure 2.1: Two dimensional DCT dictionary[Wika]

We can see that DCT is separable, when we traverse the dictionary atoms from left to right the frequency increases in the X direction and when we iterate over the atoms from the top to the bottom the frequency increases in the Y axes.

By splitting an image into blocks of 8x8 pixels and then expressing each block in terns of the DCT dictionary atoms we get a different representation of the image.

Figure 2.2: DCT of an image[Lea]

In Fig2.2 we have on the left the image on which we will apply the discrete cosine transform. On the right we can see the representation of each 8x8 block as an 8x8block of coefficients corresponding to the DCT atoms. The image is still discernible due to the fact that the first atom of the dictionary corresponds to the DC component of the signal so the top left coefficient of each block represents the average color in the block that it encodes.

After we have this representation of the image we can perform lossy compression by eliminating coefficients from each block that correspond to high frequency signals. The high frequency coefficients are responsible for the fine details in the image.

Figure 2.3: Image compressed with different ratios using JPEG

As we can see in Fig2.3 the quality of the image is slowly decaying from left to right. On the left there is the original image of size 37Kb, the next image contains only half of the DCT coefficients and has 18Kb. The last two images only keep 25% and 10% of the DCT coefficients and have 9.8Kb and 4.2Kb respectively.

2.2.2 Discrete Wavelet Transform

In the previous section we had a look on how to represent signals with cosine functions. There is a limitation when we want to extract meaning from a local region of the input signal when we look at one DCT coefficient because the basis functions have infinite support. Now we will see another way of representing signals that gives us besides the frequency information also local information about the structures.

Wavelet transforms analyze data at different scales and different location. This is done by adopting a prototype wavelet function, also known as mother wavelet, and then shifting and scaling it. Wavelet functions have finite support as opposed to the Fourier or Cosine transform, so the information that we get from these functions is localized.

High frequency wavelets are used to get temporal information and are a contracted version of the mother wavelet. Frequency analysis is achieved by using low frequency wavelets which are dilations of the prototype wavelet. Using this approach has proven very useful and efficient when analyzing data that has sharp discontinuities. Furthermore if the mother wavelet is suited for your dataset then the results will be sparse which makes wavelet decomposition an excellent tool for compression.

The first and most simple wavelet appeared in the appendix of Alfred Haar's thesis in 1909. The mother wavelet is a step function with mean 0 defined as:

$$\psi(x) = \begin{cases} 1 & \text{for } 0 \leq x \leq \frac{1}{2}, \\ -1 & \text{for } \frac{1}{2} \leq x \leq 1, \\ 0 & else \end{cases} \tag{2.5}$$

with its scaled and shifted versions

$$\psi_{j,k}(x) = \frac{1}{2^{j/2}}\psi(\frac{x}{2^j} - k) \tag{2.6}$$

Here the shift is controlled by k and the scale is specified by j. The wavelet function $\psi_{j,k}$ has width 2^j and range $[-\frac{1}{2^{j/2}}, \frac{1}{2^{j/2}}]$, starting at $k2^j$ and ending at $(k+1)2^j$ as shown in 2.4.

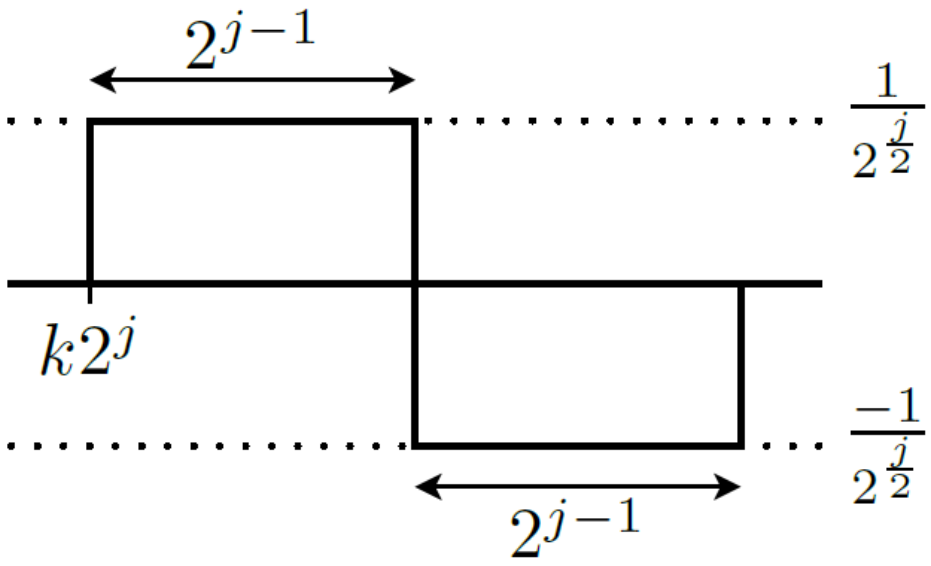

Figure 2.4: Haar wavelet $\psi_{j,k}$. Author: M. Mainberger (2008)

The factor $\frac{1}{2^{j/2}}$ ensures that $\psi_{j,k}$ has norm 1,

$$\|\psi_{j,k}\| = \sqrt{\langle \psi_{j,k}, \psi_{j,k} \rangle} = 1, \tag{2.7}$$

if j and k are integer numbers, the Haar wavelets are orthonormal:

$$\langle \psi_{j,k}, \psi_{n,m} \rangle = \begin{cases} 1 & for (j,k) = (n,m) \\ 0 & else. \end{cases} \tag{2.8}$$

Similar to the first DCT dictionary atom that represents the DC component of a signal the Haar transform has a scaling function:

$$\Phi = \begin{cases} 1 & \text{for } 0 \leq x \leq 1 \\ 0 & else \end{cases} \tag{2.9}$$

that can also be shifted and scaled

$$\Phi_{j,k} = \frac{1}{2^{j/2}} \Phi(\frac{x}{2^j} - k) \tag{2.10}$$

To represent any discrete signal of length 8 we need 8 orthonormal functions, one scaling function and 7 wavelets, that form a basis as seen in Fig.2.5

Comparable to the case of multidimensional DCT the Haar transform is also separable. For example if we want to decompose a 2D signal we first do a decomposition

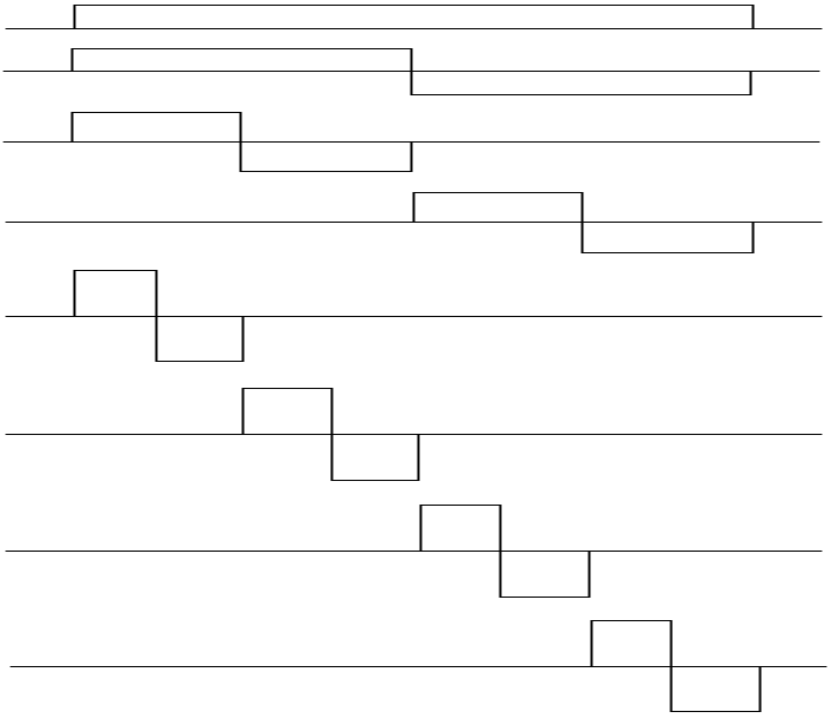

Figure 2.5: Scaling function and Haar wavelets needed to represent discrete signals of length 8. Author: S. Zimmer (2002)

in the x direction, then in the y direction. After this step we perform a decomposition only in the low-frequency parts(scaling coefficients). This procedure is repeated until a single pixel is reached.

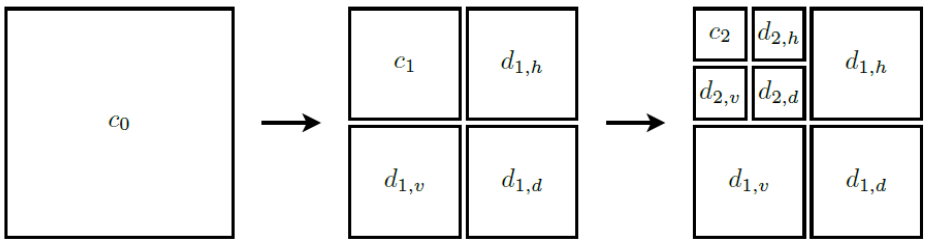

Figure 2.6: Distribution of the coefficients of the first two decomposition steps. Author: M. Mainberger (2008).

After we achieve this decomposition of the data compression is achieved like in the case of DCT by supressing coefficients that are small in magnitude. This reduces the memory needed for storing without introducing severe visual degradation.

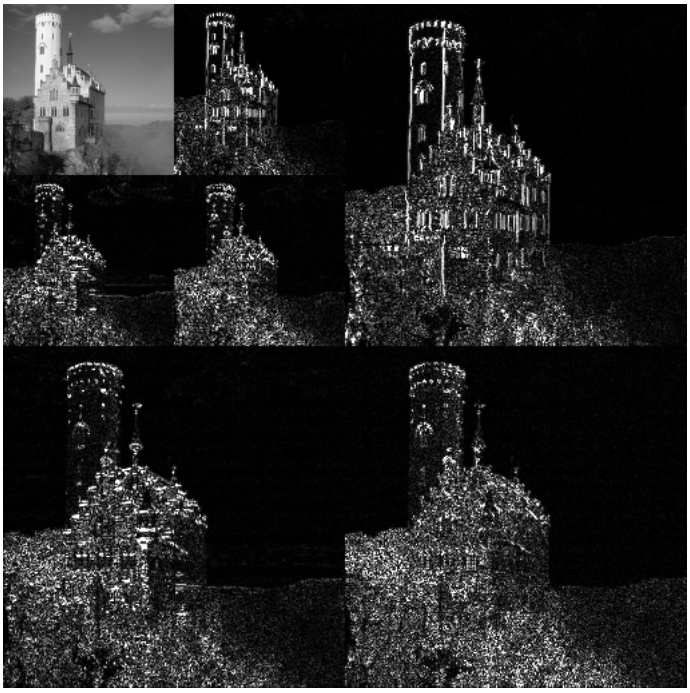

Figure 2.7: Example of 2D discrete wavelet transform.[Wikb]

Other types of wavelets have been developed over time like the Daubechies wavelets[Dau88] which describes a whole family of functions.

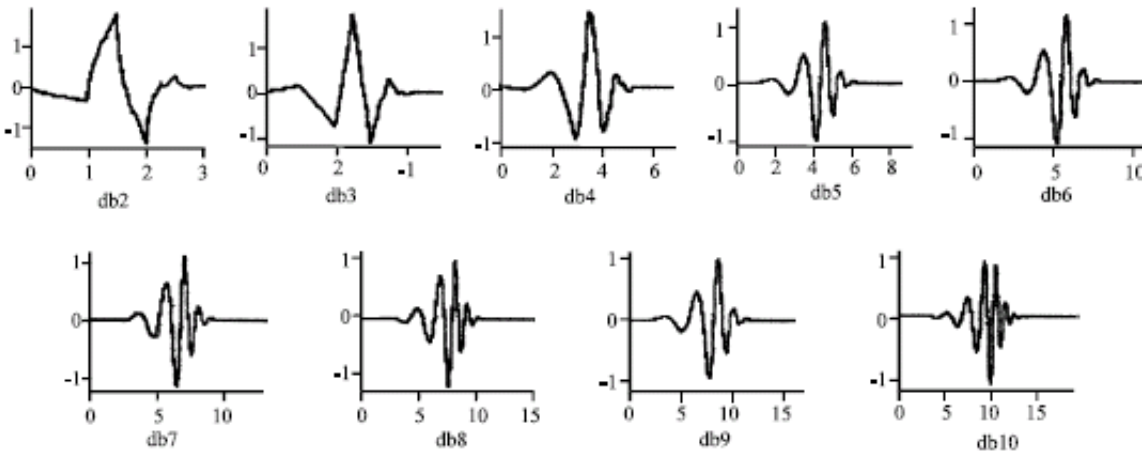

Figure 2.8: Daubechies mother wavelet functions of different order

2.2.3 Overcompleteness

All the bases that we have seen by now were complete. This means that every possible data input can be represented as a unique linear combination of the dictionary atoms.

$$\mathbf{Dx} = \mathbf{y} \tag{2.11}$$

If we think of the 3D space R^3 the dictionary(D) atoms are $e_1 = \{1, 0, 0\}$, $e_2 = \{0, 1, 0\}$ and $e_3 = \{0, 0, 1\}$. The number of basis vectors is equal to the dimension of the input data. Using just these three we can represent any point in 3D space in a unique way.

Now if we add more atoms to this already complete dictionary the dictionary becomes overcomplete. The number of basis vectors is larger than the dimension of the input data. This means that now the input signals can be represented in more than one way(infinitely many), due to the fact that the atoms are not linearly independent.

Having more representations of data gives us the possibility to choose the one that best fits us. In terms of data compression we would like to see the input data expressed with as few dictionary atoms as possible.

2.3 The Sparsity Problem

Taking a look back at equation 2.11 we know the dictionary D and the input data y, the only unknown is the vector x which stores the coefficient of each atom in order to reconstruct y. Our goal is to find x with the most amount of 0's or the lowest amount of non zero elements.

Formally speaking we have to minimize the x vector with respect to the l^0 norm such that equation 2.11 holds.

$$\min_x \left\|\mathbf{x}\right\|_0 \text{ subject to } \mathbf{y = Dx} \tag{2.12}$$

Finding the sparsest representation is a NP-hard problem as shown by G. Davis et. al. [DMA97]. Because of this impediment greedy algorithms have been developed in order to obtain an approximate solution in reasonable time. In the next sections we will take a look at two pursuit algorithms used to solve the sparsity problem.

2.3.1 Matching Pursuit (MP)

Matching Pursuit is a step-wise greedy algorithm that tries at each step to choose the best fitting dictionary atom to the input signal. This is done by computing the inner product with each dictionary atom and then choosing the atom that gives the highest result. For this to work the atoms have to be normalized.

$$i_k = argmax_w | < r_{k-1}, d_w > | \tag{2.13}$$

The next step is to update the signal by removing the weighted value of the selected atom and then go back to the first step. This cycle is repeated until we have a good approximation of the input data or for a certain number of steps. The residual is calculated at each step by subtracting the contribution of the best fitting atom from the previous residual

$$r_k = r_{k-1} - i_k d_{ik}, \tag{2.14}$$

where r_k is the residual at step k, i_k is the coefficient of the atom d_{ik} chosen at step k.

Some of the matching pursuit advantages are its simplicity and control over the desired outcome either by setting the maximum number of iterations or by providing a threshold for the residual. As a disadvantage we have to search over all the dictionary atoms in each iteration. This introduces computational complexity.Because this is a greedy algorithm we also have no guarantee that we will get the optimal solution.

2.3.2 Basis Pursuit (BP)

Another popular solver for the sparsity problem is Basis Pursuit in which we replace the l^0 norm in equation 2.12 with an l^1 norm. Because of this change the intractable expression becomes now solvable by means of linear optimization. We can formulate the optimization term as:

$$\min_x \{ \left\| \mathbf{y} \text{ - } \mathbf{Dx} \right\|^2 + \lambda \left\| \mathbf{x} \right\|_1 \} \tag{2.15}$$

The first term of the minimization represents the constraint and the second term the objective function. The parameter λ is a penalty term that allows us to choose between the importance of sparsity of the result and the error of the reconstruction.

For this kind of problems there are general solvers such as interior point methods or for larger problems other methods like LASSO(least absolute shrinkage and selection operator).

When compared to the matching pursuit algorithm the basis pursuit has soft thresholding. The solution is globally optimal as opposed to the greedy optimization performing at each step in the MP algorithm. As a disadvantage the BP needs to choose the penalty parameter λ appropriate in order to force N-sparsity while in MP we get it simply by stopping after N steps.

3 Methods Used

3.1 Overview

The proposed method for sparsifying time dependent volume data sets uses an overcomplete dictionary that is trained over patches extracted from the data that will be compressed. The dictionary is trained over a set of examples using the K-SVD algorithm proposed by Michal Aharon et. al.[AEB06].

Training is done in an iterative manner, updating each time the dictionary atoms in order to get a better sparse representation of the training examples. Dictionaries that are obtained using machine learning algorithms like Principal Component Analysis (PCA), Method of Optimal Directions (MOD), K-SVD and others give better results than analytical dictionaries. However these unstructured dictionaries are more costly to apply. Also the size of the dictionary is limited by complexity constraints since the approximation methods like Matching Pursuit have to compare the signal at each step with all the dictionary atoms.

After the learning step is done the volume is broken down into blocks and expressed in terms of the dictionary atoms using Orthogonal Matching Pursuit in order to obtain a sparse representation. All the coefficients are stored as column vectors and form a sparse matrix that will be saved for reconstructing the volume.

Due to the fact that the dictionary is unstructured we need to store it in order to reconstruct the volume after we obtain its representation over the dictionary. In order to reduce the size needed to save the overcomplete dictionary we compute each atom as a sparse representation over a generic base dictionary, thus storing just the sparse matrix used to reconstruct the dictionary.

In the next section we will take a closer look on how the K-SVD algorithm works and how we express the final dictionary as a sparse representation over a generic one.

3.2 K-SVD Algorithm

3.2.1 K - Means

The first part of the algorithm is a clustering method called K - Means. There is a close relation between sparse representation and clustering. K - Means is a method of vector quantization(VQ) in which we have a number of input signals (vectors) $\mathbf{Y} = \{ \mathbf{y}_i\}_{i=1}^N$ and we want to find K codewords that best fit the data. Here the number of input signals is larger than the number of codewords $(N \gg K)$. The codewords are column vectors and form the codebook matrix $\mathbf{C} = [\mathbf{c}_1, \mathbf{c}_2, ..., \mathbf{c}_K]$. In the case of vector quantization each signal is associated to the closest codeword (under l^2 norm distance).

If we think about the sparsity problem we can write $\mathbf{y}_i = \mathbf{C}\mathbf{x}_i$, where $\mathbf{x}_i = \mathbf{e}_j$ is a vector from the trivial basis with one at position j and zero everywhere else. This is an extreme version of sparse coding where only one atom is allowed for the reconstruction of the original signal. The index j is chosen such that

$$\forall_{k \neq j} \|\mathbf{y}_i - \mathbf{C}\mathbf{e}_j\|_2^2 \leqslant \|\mathbf{y}_i - \mathbf{C}\mathbf{e}_k\|_2^2 \tag{3.1}$$

After every input signal is assigned to a codeword we can compute the error for each signal y_i as $e_i^2 = \|\mathbf{y}_i - \mathbf{C}\mathbf{x}_i\|_2^2$ and the overall MSE is

$$E = \sum_{i=1}^{K} e_i = \|\mathbf{Y} - \mathbf{C}\mathbf{X}\|_F^2 \tag{3.2}$$

We are interested in finding the best possible codebook C such that the overall error is minimized. To achieve this goal K - Means works in an iterative manner and has two steps. The first step is the one described above where every input signal y_i is assigned a codeword and the second step updates the codewords in order to reduce the overall error term.

$$\min_{\mathbf{C},\mathbf{X}} \{\|\mathbf{Y} - \mathbf{C}\mathbf{X}\|_F^2\} \text{ subject to } \forall i, \mathbf{x}_i = \mathbf{e}_k \text{ for some k} \tag{3.3}$$

For the second step we fix $\mathbf{X}$ and update the codebook $C^{(J-1)}$, where $C^{(J-1)}$ represents the codebook at iteration step j of the algorithm. Each codeword c_k is updated by averaging over all input signals y_i that have been assigned the codeword c_k.

$$c_k^{(J)} = \frac{1}{|R_k|} \sum_{i \epsilon R_k^{(J-1)}} y_i \tag{3.4}$$

where R_k is the set of all indices of the input signals that were assigned to c_k.

At each iteration the overall error is reduced or stays the same. Since the error is lower bounded at zero and in each step of the algorithm we have a monotonic decrease of the MSE we are guaranteed convergence to at least a local minimum.

3.2.2 Generalized K - Means

We have seen how the K - Means algorithm works in a two-step iterative manner for vector quantization (VQ). The sparse representation problem can be seen as a generalization of VQ in which the input signal can be obtained by a linear combination of codewords, which we will call dictionary atoms from now on and the dictionary is equivalent to the codebook. Now the coefficient vector can have more than one non-zero element and for this case we express the minimization to search for the best dictionary **D** and sparsest coefficient vector **X** in order to represent the input signals **Y**.

$$\min_{\mathbf{D},\mathbf{X}}\{\|\mathbf{Y}-\mathbf{DX}\|_F^2\} \text{ subject to } \forall i, \|\mathbf{x}_i\|_0 \leq T_0 \tag{3.5}$$

There are two goals that need to be optimized **D** and **X**. As in the case of K - means we will consider first that the dictionary is fixed and in this stage we will try to find the sparsest representation of **Y** over **D** which optimizes **X**. This is the sparse coding stage and can be expressed as

$$\|\mathbf{Y}-\mathbf{DX}\|_F^2 = \sum_{i=1}^{N} \|y_i - Dx_i\|_2^2 \tag{3.6}$$

this can be decoupled into N distinct problems

$$\min_{x_i} \|y_i - Dx_i\|_2^2 \text{ subject to } \|x_i\|_0 \leq T_0 \tag{3.7}$$

We have seen that this set of problems can be solved by different pursuit algorithms in Chapter 2.

After obtaining a sparse representation of **Y** over **D** we can proceed to the next step which will update the dictionary atoms one atom at a time. In this stage we consider **X** to be fixed as well as all atoms of **D** except one d_k which represents the k^{th} column of the dictionary. Here x_T^j represents the coefficients that correspond to the dictionary atom d_k and not the k^{th} column vector of **X**.

$$\|\mathbf{Y} - \mathbf{DX}\|_F^2 = \left\|\mathbf{Y} - \sum_{j=1}^{K} d_j x_T^j\right\|_F^2 \tag{3.8}$$

we can isolate the term which contains the dictionary atom that we are interested in

$$\left\|(\mathbf{Y} - \sum_{j\neq k} d_j x_T^j) - d_k x_T^k\right\|_F^2 = \left\|E_k - d_k x_T^k\right\|_F^2 \tag{3.9}$$

Here E_k represents the error matrix for all N examples where the contribution of atom d_k has been removed. We could apply here the SVD to find an alternative for d_k and x_T^k. SVD will find the closest rank-1 matrix that approximates the error matrix E_k and thus minimizing the error. Although this would minimize the error term it is a mistake to apply here the SVD low rank matrix approximation because we can not enforce the sparsity of x_T^k and it would probably be filed with non-zero values.

In order to avoid this problem we define ω_k which is the group of indices of the examples y_i which use the atom d_k in their reconstruction.

$$\omega_k = \{i | 1 \leq i \leq K, X_T^k(i) \neq 0\}. \tag{3.10}$$

Define Ω_k which is a $N \times |\omega_k|$ matrix that has ones at $(\omega_k(i), i)$ positions and zero elsewhere. When multiplying $\mathbf{x}_R^k = \mathbf{x}_T^k \Omega_k$ only the non-zero entries of $\mathbf{x}_T^k$ remain and the length of $\mathbf{x}_R^k$ becomes equal to $|\omega_k|$. Also we can select only the examples that use the dictionary atom d_k, $Y_k^R = Y\Omega_k$ this matrix has size $|n \times \omega_k|$. Similarly we can select only the error columns that correspond to d_k by multiplying Ω_k with the error matrix $E_k^R = E_k \Omega_k$.

Now if we go back to eq. 3.9 and minimize with respect to d_k and x_T^k forcing the solution $\tilde{x}_T^k$ to have the same support as the original x_T^k we get the following minimization

$$\left\|E_k - d_k x_T^k\right\|_F^2 = \left\|E_k^R - d_k x_R^k\right\|_F^2 \tag{3.11}$$

applying SVD on the restricted error matrix $E_k^R = U\Delta V$ will give us the solution for $\tilde{d}_k$ as the first column of **U** and for $\tilde{x}_R^k$ the first column of **V** multiplied by $\Delta(1,1)$. The resulting solutions have the following properties: $\tilde{d}_k$ is normalized due to the fact that the **U** matrix is orthonormal and the support of $\tilde{x}_R^k$ stays the same or shrinks due to possible nulling of terms.

These two steps, sparse coding and dictionary update are repeated until we obtain the desired error or after a certain number of steps if we get stuck in a local minimum. Due to the fact that the error is lower bounded by zero and in every update

step the error decreases or stays the same convergence is assured. The convergence relies on the success of the sparse coding stage which for small enough T_0 algorithms like matching pursuit or base pursuit perform well. An important aspect of the K-SVD algorithm is the fact that with every dictionary atom update the coefficients are also updated. This leads to a convergence than requires on average 4 times less iterations and provides a better solution than updating just the atoms and keeping the coefficients fixed. A disadvantage of this approach is that parallel update of the dictionary atoms is prohibited.

3.3 Learning Sparse Dictionaries

We have seen how a dictionary can be trained over several examples to better sparsify the data. In order to reconstruct the sparsified data we need the coefficients of the dictionary atoms that comprise each input signal and the learned dictionary. Due to the fact that the dictionary is overcomplete and is learned from examples there is no efficient way of generating it from a mathematical formula like the cosine or wavelet dictionaries are. This means that we need to also store the dictionary along with the reconstruction coefficients. To reduce the amount of memory needed to store the dictionary we can learn a sparse representation of it over a generic dictionary as shown in [RZE10].

$$\mathbf{D} = \Phi\ \mathbf{A} \tag{3.12}$$

where Φ is a generic dictionary and **A** is a sparse matrix.

Similar to the K-SVD algorithm we have to solve the following minimization problem

$$\begin{aligned} &\min_{A,\Gamma} \|X - \Phi A\Gamma\|_F^2 \\ &\text{Subject to } \begin{cases} \forall i\ \|\gamma_i\|_0^0 \leq t \\ \forall j\ \|a_j\|_0^0 \leq p, \|\Phi a_j\|_2 = 1 \end{cases} \end{aligned} \tag{3.13}$$

what differs from the original K-SVD algorithm is that the dictionary atom is constrained to $d = \Phi a$ with $\|a\|_0^0 \leq p$. The first step of sparse coding the signal is similar to the original algorithm but the dictionary atom update step becomes

$$\begin{aligned} &\{a, g\} := \underset{\mathbf{a,\ g}}{\text{Argmin}} \left\|E - \Phi a g^T\right\|_F^2 \\ &\text{Subject to } \|a\|_0^0 \leq p, \|\Phi a\|_2 = 1 \end{aligned} \tag{3.14}$$

To solve this minimization problem an alternating minimization over a and g is applied in order to get the sparse dictionary representation over the base dictionary Φ. An in-depth method of solving the minimization problem can be found in [RZE10].

After we obtain the matrix **A** we no longer need to store the whole dictionary but only **A** that reconstructs the learned dictionary **D** from Φ. At this point we have all the tools that are needed to understand the flow of the algorithm.

3.4 Algorithm

The algorithm is implemented in **MATLAB** and has three sections: **Setup**, **Dictionary learning**, **Sparsification of the volume**. In the first section all the parameters that set how accurate we want the learning and sparsification are defined and also the size and the type of the generic dictionary are fixed. In the second section the dictionary learning process determines **A** and stores it. In the last part of the algorithm the volume is divided into n^3 size blocks and sparsified using the dictionary learned previously.

3.4.1 Setup

The first step in the setup process is to fetch the volume data from external sources and store it in an array. After the data is loaded a matrix, which has the size that the volume has in each dimension in voxels, will be filled. I also normalize the range of the data to fit into [0,255] interval.

After the data is loaded and reshaped the dictionary size is chosen. I have chosen to set the dictionary atom to be of size 8^3 for two reasons. The first one has to do with performance due to the fact that each block of the volume has to be compared to the dictionary atoms at every step. For example if we chose a dictionary atom of size 8^3 and the dictionary to be two times overcomplete we end up with 1024 dictionary atoms, if we chose the atom to be 16^3 we get 8192 atoms. At each step of the dictionary learning phase we would need to update the dictionary atoms and sparsify the training examples which is an expensive process.

The second argument for choosing a smaller dictionary size is the training data that needs to be provided. In the case of the dictionary with the 8^3 atom size which has 1024 atoms we provide 80.000 training examples. If the atom size would be 16^3 we would have to provide proximately 660.000 training blocks from the input data which would make the dictionary learning running time impracticable and for smaller volumes there are not that many blocks in the volume.

Due to this considerations I have chosen a two times overcomplete dictionary with the dictionary atom size 8^3, 8 in each direction. As for the base dictionary Φ I have chosen the DCT dictionary.

The second part of the setup is reserved for setting the error σ for the reconstruction, the sparsity of each trained atom, number of training samples, number of iterations for the dictionary training and other parameters.

3.4.2 Dictionary training

At this stage the original volume is broken into 8^3 blocks. Depending on the number of training examples that were set blocks are chosen from the volume equidistantly and serve as input for the K-SVD algorithm. Before passing the blocks to the K-SVD algorithm the DC component is calculated and removed because we are training the dictionary over the DCT dictionary and this saves memory.

At each step of the K-SVD dictionary update the mean number of atoms needed per block is displayed. The update algorithm will run for a number of iterations or can be stopped when the average number of atoms required per training block no longer decreases from one iteration to another. As output the learning algorithm provides the sparse **A** matrix, Γ and the general error. The sparse matrix **A** is the one that we are interested in and it will be saved along with the coefficients for reconstruction.

After obtaining the sparse matrix **A** from the learning algorithm we have the final representation of the dictionary **D**. Due to the fact that we are working on time dependent volume data sets we can use the **A** matrix from one time step as a starting point for the dictionary learning algorithm for the next time step thus reducing the number of iterations needed for convergence. This can be done every time we get to a new frame of the time dependent volume or after the compression performance degrades over a preset threshold.

3.4.3 Volume Sparsification

Sparsification is a straight forward operation in which each block of the volume is expressed as a linear combination of the learned dictionary atoms. We can choose in which manner we would like to break up the volume into blocks of the atom size. If we set the distance between blocks to be equal to the dimension of the dictionary atom then no block will overlap and we will end up with the lowest amount of coefficients for the whole volume. On the other hand if we set the distance between blocks to be less than the atom size we will get overlapping blocks which will offer us a greater fidelity of the

original volume but we will take a penalty in terms of number of coefficients needed to reconstruct the volume.

For sparsifying the volume I have used Orthogonal Matching Pursuit(OMP) which is very similar to MP with the mention that instead of projecting the residual on the best fitting atom the input signal is projected on atoms that were chosen by that point and then the residual is obtained. Another observation is that this process of sparsifying the volume which is broken down into blocks is embarrassingly parallel due to the fact that in order to obtain the sparse representation we only need access to the dictionary and the block in question. There is no interaction between two OMP runs so adding more processing units will decrease the time needed to sparsify the volume in a linear manner.

After all the blocks have been sparsified the algorithm provides the sparse Γ matrix which will be stored along with $\mathbf{A}$ and the DC component of every block. Having all this information stored we can reconstruct the volume blocks by multiplying each column of Γ with the dictionary and adding the corresponding DC component. This is also a very easily parallelizable problem.

4 Results

In this chapter we will analyze the results that were obtained over two time-dependent volume data sets. The first data set is a physics simulation capturing the core-collapse of a supernova [BM07]. The simulation has 60 time steps in which it captures the first second of a supernova explosion. The volume is a scalar field of size $432 \times 432 \times 432$. The second data set is a Turbulent Combustion Simulation [YCS07] where different aspects of turbulent flames are simulated. In this data set multiple scalar fields are provided conveying different aspects of the simulation like temperature, mixing rates and species concentration. Each time frame of the simulation has five scalar fields carrying different information about the simulation and has size $480 \times 720 \times 120$.

On this data sets I have performed different tests to evaluate the performance of the algorithm in terms of compression ratio, error, time needed to train the dictionary and to sparsify the volume, reusability of the dictionary over different time steps and viability of the dictionary in other volumes unrelated to the one that it was trained.

4.1 Compression Vs Quality

The first thing that we will take a look at is the trade-off between compression and quality of reconstruction. This is a typical decision that needs to be taken in lossy compression algorithms. Usually we can choose one at the expense of the other. If we want a high rate of compression the quality of the reconstruction will take a hit as we have seen with the DCT and Wavelet transform, on the other hand if we want a very good reconstructed volume we need to store more coefficients and thus the compression ratio will drop. Depending on the desired outcome and specific needs a trade-off can be found.

The following results are from the supernova data set, frame 1295, which is 314928KB on disk in its original form. For this test I have varied the amount of error that we allow for each reconstructed block by changing the values of σ. I have recorded the resulting size of the compressed volume, the peak signal to noise ratio(PSNR), number of non zero(NNZ) entries in the Γ matrix as well as the average number of atoms required per block in order to reconstruct the volume.

Table 4.1: Compression Vs Quality on supernova data set frame 1295

Sigma	NNZ(Gamma)	Avg dict atom	PSNR	Size(Kb)	Size ratio
1	1378983	9	56.7	13796	1:22
2	615639	3.9	51	6855	1:45
3	334934	2	47.8	4378	1:71
4	181261	1	45.8	3017	1:104
5	105078	0.58	44.4	2311	1:136

As we can see in Table 4.1 as we increase the error that we tolerate the compression ratio grows from 22, when σ is equal to one, to 136 when σ is equal to five. At the same time the PSNR drops from 56.7dB to 44.4dB. We can also observe how the amount of coefficients needed for reconstruction decrease when we allow a higher error which is what we would expect. The most eloquent indicator of the final size of the sparsified data is the average number of dictionary atoms needed per block followed by the number of non-zero entries of the Γ matrix.

4.2 Training and Sparsifying Time

Another important metric is the time it takes to train and then sparsify the volume. The tests were done on the same frame of the supernova data set on a machine with two Intel Xeon processors each having 12 cores and 24 threads.

Table 4.2: Dictionary and Sparsifying times

Sigma	Dictionary time(sec)	Sparse time(sec)	Sparse time parallel(sec)
1	1005	1195	28
2	275	548	13.7
3	126	315	11
4	74	200	8.3
5	54	148	6

In Table 4.2 we can see the time it takes to train a dictionary and sparsify the volume. We can see here that the time it takes to train the dictionary is very high for low error due to the fact that in each step we need more coefficients to get the learned dictionary from the base dictionary in that error limit. Also this training time can not be avoided since in the update stage every dictionary atom update causes also an update

in the coefficients matrix so in order to have a coherent dictionary atoms need to be updated sequentially.

On the other hand the sparsification time is highly parallel and here we can reduce the time needed by adding more processing units. We can see that for low error where there are a lot of coefficients to be determined the improvement we get from 48 threads is $\times 40$, which is almost linear. As the error increases and the number of coefficients drops the gain from adding more processing units drops a bit due to the fact that operations like dictionary copying now take a larger percentage of the whole sparsification process.

4.3 Dictionary Reusability

An interesting question in the context of time-dependent volumes is the reusability of a dictionary learned in the previous time step. The next set of tests will give us an insight on the feasibility of using one dictionary across multiple time frames. In order to determine this we are interested in how the compression ratio and quality of the reconstruction evolves across different time steps when we don't learn the dictionary at each time step.

In Table 4.3 we have listed on the second column all the PSNR values for dictionaries learned at each frame and the third column shows the PSNR values for the same frames but this time the dictionary used for sparsifying the volume was trained in the first frame and then reused in all the next time steps.

This is not surprising since the sparsification process does not stop until the residual is under the desired threshold thus resulting in a similar PSNR even though the dictionary was not the best fit for the current frame.

Now we would like to analyze what happens to the compression ratio when we reuse the same dictionary over multiple time frames. We would expect, since it is not learned on the current frame of the volume, to take a penalty in terms of compression.

The next set of tests were done on the second data set of the turbulent flames in which we compared different update strategies to see how the dictionary update will affect the compression. For this test we propose three strategies to compare. The first is to update the dictionary every time frame of the volume, second learn the dictionary on the first frame and then use it for all other frames and third update the dictionary every ten frames.

In Fig 4.1 we can see how the memory footprint of the reconstructed volume changes over different time frames. In the first graph we see how the compression of the

Table 4.3: Quality of reconstructed volume when reusing dictionary

Frame	Relearned Dict PSNR(dB)	Reused Dict PSNR(dB)
1	50.93	50.93
2	50.84	50.84
3	50.78	50.78
4	50.8	50.78
5	50.83	50.81
6	50.81	50.81
7	50.81	50.79
8	50.77	50.75
9	50.66	50.65
10	50.59	50.58
11	50.50	50.50
12	50.41	50.41
13	50.33	50.33
14	50.26	50.27
15	50.21	50.21
16	50.13	50.14
17	50.12	50.12
18	50.08	50.08
19	49.98	49.97
20	49.87	49.87
21	49.78	49.78
22	49.7	49.7
23	49.62	49.62
24	49.56	49.55
25	49.51	49.5
26	49.51	49.49
27	49.54	49.52
28	49.56	49.54
29	49.57	49.55
30	49.6	49.57

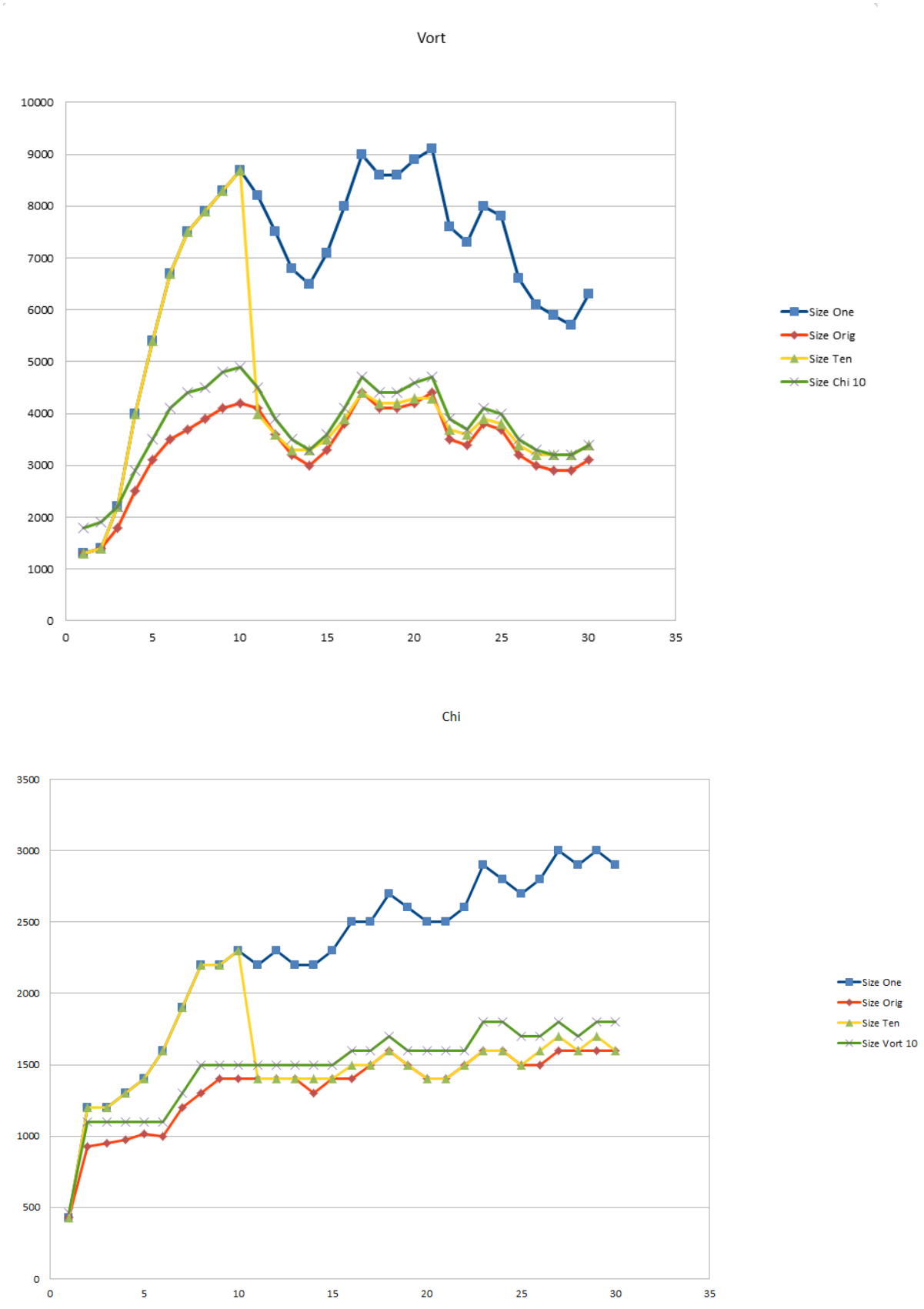

Figure 4.1: Compression results for different dictionary update strategies.

vorticity of the flow is compressed and in the second we can see how the chi component of the flow is compressed.

The red line in both graphs represent the size in KB of the sparse coded volume. We can see that in the first ten frames the size needed for representing the volume increases due to the fact that at first the flames begin to burn and the flames gradually fill the volume. After that a repeating pattern of the burn appears and the space needed to sparsely code the volume oscillates around a value.

The second strategy is to learn the dictionary on the first time frame and then reuse it for the upcoming frames without update. The blue line shows the outcome of this strategy. Due to the fact that we learned the dictionary at the first time step when not much is happening the dictionary trains mostly on empty samples and thus we get a worse compression. Nevertheless we can still observe the trend that we saw in the first strategy where we have an increase in size in the first ten frames and then oscillation.

The yellow line represents the third strategy in which the dictionary is relearned every ten frames. We can see that on the first ten frames we obtain the same outcome as the previous strategy since both have as support a dictionary learned in the first frame. After frame ten the dictionary is updated and it will be used for the next ten frames. As we can see the result is much closer to the result obtained by learning the dictionary on each frame. This is a very important result since it means that we can reuse the dictionary from previous frames when the volume does not drastically change with low penalty in terms of compression performance. By doing so we spread out the high cost of training the dictionary, which can not be done in parallel, over multiple frames thus improving the performance.

The last series of tests are represented by the green line in the graph. For this set of tests I have used the dictionary trained on the 10^{th} frame of the vorticity volume for the whole time steps of the chi series and vice versa. I have chosen the 10^{th} time step because from that point forwards the simulation is entering an oscillation phase. As we can see the green line follows more closely the red line as oppose to the blue line which is also a one time learned dictionary. Even more interesting is the fact that the dictionary was trained on another volume than the one it was used for sparse coding.

Another scenario in which we tested how the trained dictionaries perform is when we use them on volumes that are unrelated to the volume that they were trained on. In Fig.4.2 we have used dictionaries trained at different time steps of the turbulent combustion volume, on the vorticity scalar field, and used them in the supernova volume. The compression results were compared with the original results obtained by training dictionaries at every frame of the supernova data set.

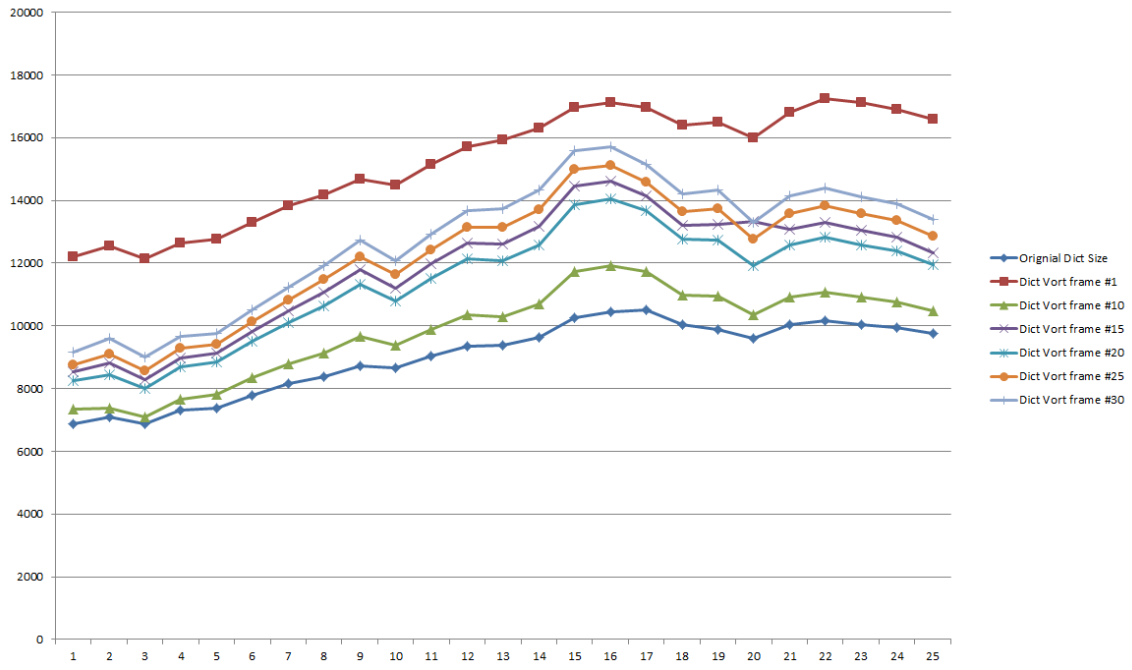

Figure 4.2: Compression results when reusing dictionaries on other volumes.

We can see the compression performance of the dictionaries trained on the supernova volume in dark blue. In terms of compression those are the most efficient. The other lines that are plotted in the graph belong to dictionaries that were learned from different frames of the turbulent combustion data set. The best dictionary out of those is the one learned on time-step 10 and is depicted with a green line. The worst dictionary in terms of compression is the one learned on the first time-step of the combustion simulation, where not much is happening.

The size of the compressed volume varies from 10% to 40% more than the original depending on the dictionary that was chosen. Another interesting correlation seems to be between the size of the reconstruction and how well the dictionary fits the new data set. The first frame of the turbulent combustion has the lowest reconstructed size and is the worst performing dictionary in the supernova data set and the 10^th frame has the largest reconstructed size and is the best fitting dictionary.

As we can see from this graph dictionaries trained on one data set can be used on other data sets with varying degrees of success but are worse than dictionaries trained on the same volume and used for multiple time-steps.

4.4 Visualizing the results

Now we will take a look on how the reconstructed volume compares to the original one another lossy compression ZFP [LI06]. ZFP is an open source library for floating-point data compression. In the next panels we will see a reconstruction of the supernova data set with K-SVD and ZFP. The original volume weighs 315MB the K-SVD sparsified volume weighs 7MB and the ZFP 10MB.

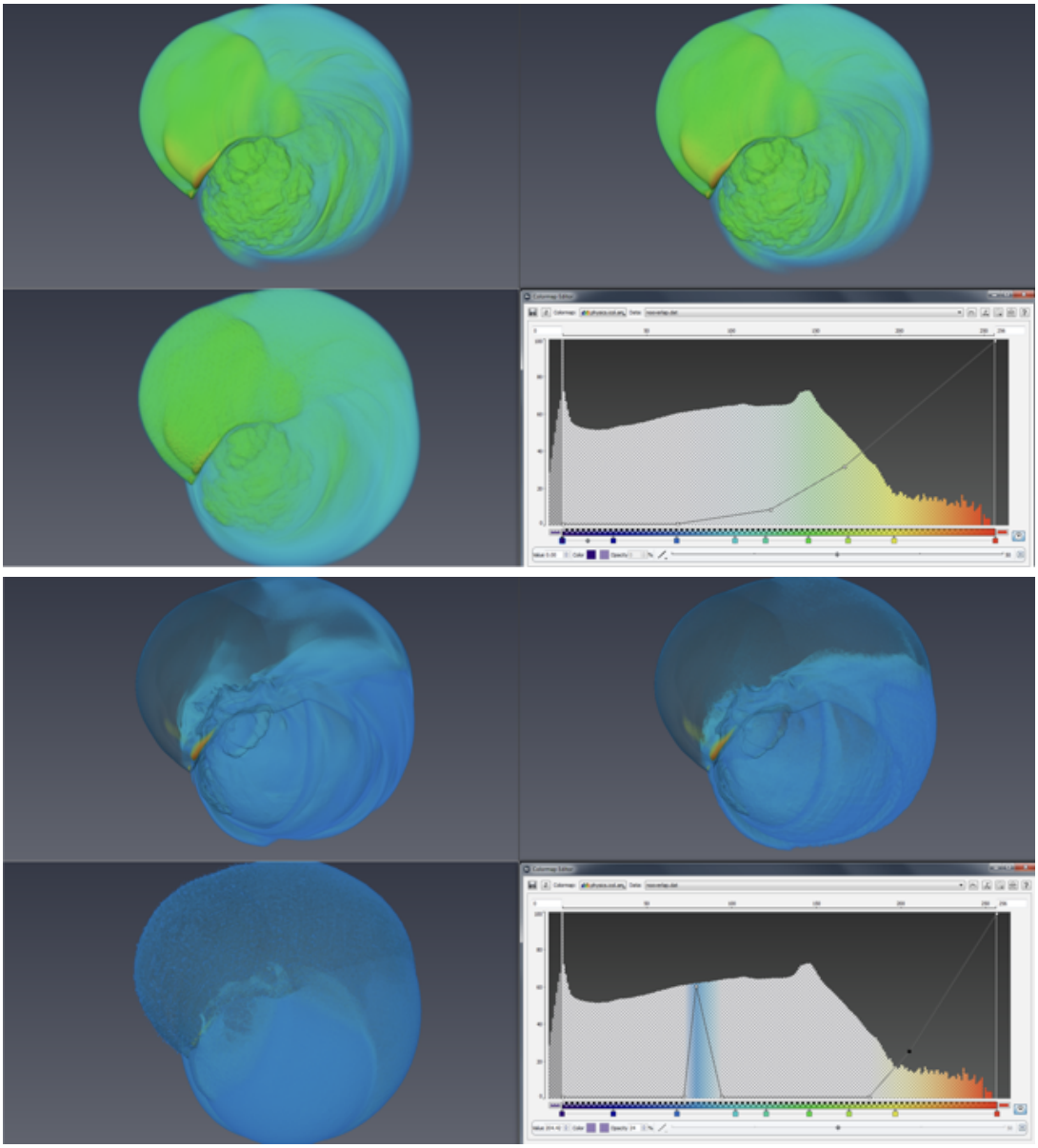

Figure 4.3: Visual comparison of compression results.

In each panel of Fig.4.3 and Fig.4.4 the top left image is the uncompressed volume, followed by the K-SVD compression on the right and ZFP compression on the bottom left. The transfer function that was used can be seen in the bottom right part of each panel.

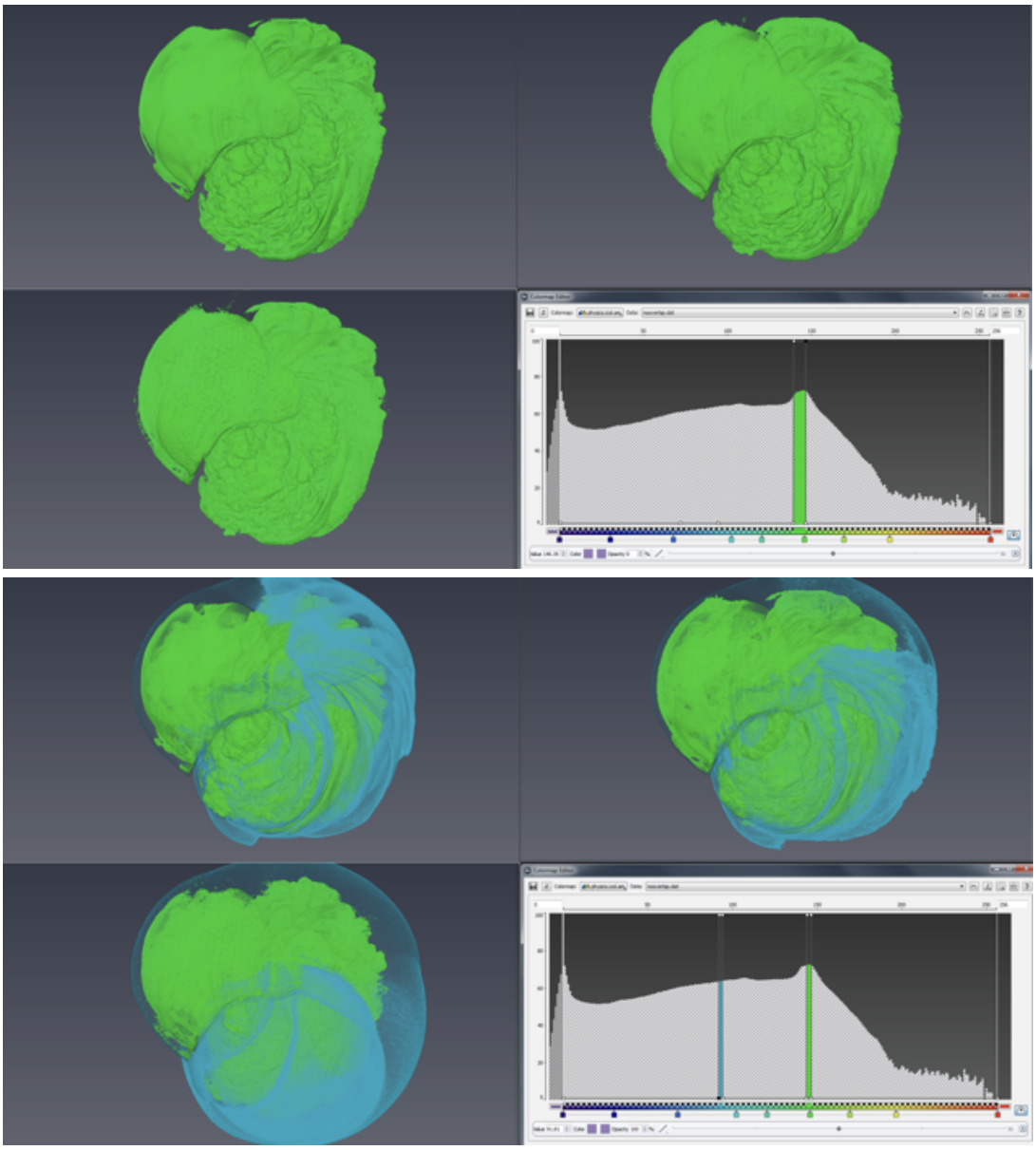

Figure 4.4: Visual comparison of compression results.

We can see from Fig.4.3 and Fig.4.4 that the K-SVD reconstruction preserves more details than the ZFP compression. In the second panel of Fig.4.3 that both methods present block artifacts but in our reconstruction the artifacts are less accented and

also the fine details are better preserved. The red spot that appears on the left of the supernova is better preserved in our reconstruction while lost in the ZFP.

5 Conclusions

5.1 Summary

Simulations obtained by HPC runs produce high amounts of data which pose a problem when it comes to storing the data for later analysis and visualization. Compression is used in order to reduce the size of memory needed to store this simulations. Due to the fact that computation is getting more and more cheap when compared to I/O operations we can afford to spend more on computing a good compression in order to reduce the I/O operations needed for storing the results.

In the second chapter we have reviewed some classical methods in which compression can be achieved. DCT and Wavelet transform are two classic techniques that use a dictionary as support and express blocks of the input data as linear combination of the dictionary atoms. These dictionaries can be overcomplete which leads to a better sparsification of the data but at the same time introduces a NP hard problem.

The problem arises from the fact that in the context of an overcomplete dictionary there is no unique representation of the input signal and a choice has to be made with the goal of obtaining the sparsest representation possible. For this problem two greedy algorithms were presented, Matching Pursuit and Basis Pursuit, which produce a sparse representation of the input signal within a reconstruction tolerated error.

In chapter three we have seen how a dictionary can be trained over a set of examples from the volume we want to sparsify using the K-SVD algorithm. K-SVD is computationally more complex than classical compression techniques, trading speed for a better sparsification of the data. Since computation is cheap we can afford such a trade-off with the goal of obtaining a more sparse result. In order to further reduce the size in memory needed for storing, the learned dictionary can be expressed as a sparse representation over a generic dictionary. By doing so we will store just the sparse representation over the base dictionary and at reconstruction compute the learned dictionary.

Testing the proposed method on time-dependent data sets gave us a perspective on the performance we can expect from it. In terms of speed there are two different components that give us the final performance of the method. The training stage is a

lengthier process and can not be run in parallel due to the way that dictionary atom updates also change sparse representation of the training signals. The second component is the sparsifying stage which is less demanding and highly parallel. From tests we have seen that adding more processing units to the task increases performance in a linear manner. We have also seen the quality of reconstruction and compression ratio we can expect depending on the different errors we tolerate in our reconstructed volume.

Reusing the dictionary trained at one time step for the next steps proved to be a viable strategy thus reducing the average time needed for the dictionary training phase. Also using one dictionary on different scalar fields of the same volume proved to be a viable option. Using dictionaries on completely different data sets turned out to have varying degrees of success depending on what frame the dictionary was trained on.

Finally we compared our reconstructed volume to another lossy compression method. We have seen that our reconstruction has less block artifacts and maintains more high frequency details than the other method.

5.2 Future work

There are a few directions that I would like to pursuit in order to improve and continue the current work. Testing more data sets to observe the behavior of the method in more situations will give us a better understanding about the reusability of dictionaries in other data sets.

Devise new metrics that can tell us beforehand how a dictionary will fit another data set and how similar two dictionaries are to one another. These metrics would help us to identify the best dictionary from one data set that can be reused in another data set and thus reducing the trial and error process of finding a good dictionary from another data set.

Another improvement that can be brought is to reduce the sparsification time by migrating the OMP work from CPU to GPU. This will take full advantage of the parallel nature of this problem and will make the compression of larger volumes a much faster process. Furthermore in order to assure an even load we could do a dictionary update step at each frame instead of relearning the whole dictionary every ten frames.

By scaling the dictionary atoms and then reconstructing the volume with original coefficients we could have a representation of the volume at different resolutions. This is particularly interesting since we would have for each block of the volume a different scaling function given by the reconstruction coefficients as oppose to classical rescaling algorithms. This could be used as a starting point for simulations at a larger scale.

Bibliography

[AEB06] M. Aharon, M. Eladand, A. Bruckstein. “K-SVD: An Algorithm for Designing Overcomplete Dictionaries for Sparse Representation.” In: *IEEE TRANSACTIONS ON SIGNAL PROCESSING, VOL. 54, NO. 11* (2006) (cit. on p. 29).

[ANR74] N. Ahmed, T. Natarajan, K. R. Rao. “Discrete Cosine Transform.” In: *IEEE Transactions on Computers, C–23 (1): 90–93* (1974) (cit. on p. 18).

[BM07] J. M. Blondin, A. Mezzacappa. “Pulsar spins from an instability in the accretion shock of supernovae.” In: *Nature, 445:58-60* (2007) (cit. on p. 37).

[CD99] E. J. Cande‘s, D. L. Donoho. “Curvelets – a surprisingly effective nonadaptive representation for objects with edges.” In: (1999) (cit. on p. 18).

[Dau88] I. Daubechies. “Orthonormal Bases of Compactly Supported Wavelets.” In: *Commun. Pure Appl. Math.* (1988) (cit. on p. 24).

[DMA97] G. Davis, S. Mallat, M. Avellaneda. “Adaptive greedy approximations. Journal of Constructive Approximation.” In: (1997), 13:57–98 (cit. on p. 25).

[DV05] M. N. Do, M. Vetterli. “The contourlet transform: an efficient directional multiresolution image representation.” In: *IEEE Trans. Image Process., vol. 14, no. 12, pp. 2091–2106* (2005) (cit. on p. 18).

[Fri06] M. Friendly. “A Brief History of Data Visualization.” In: *Handbook of Computational Statistics: Data Visualization*. Ed. by C. Chen, W. Härdle, A Unwin. Vol. III. (In press). Heidelberg: Springer-Verlag, 2006, pp. 1–7 (cit. on p. 11).

[Fun36] H. G. Funkhouser. “A note on a tenth century graph.” In: *In proceedings of SIGGRAPH* (1936), pp. 260–262 (cit. on p. 11).

[JLH11] D. A. P. John L. Hennessy. *Computer Architecture, Fifth Edition: A Quantitative Approach*. Morgan Kaufmann Publishers Inc. San Francisco, CA, USA, 2011. ISBN: 012383872X 9780123838728 (cit. on p. 15).

[Kin04] G. Kindlmann. *Superquadric Tensor Glyphs*. 2004. URL: http://www.cs.utah.edu/~gk/papers/vissym04/web/slide026.html (cit. on p. 14).

[Lea] *DCT encoding exampel*. http://obsessive-coffee-disorder.com/image-encoding (cit. on p. 20).

[LI06] P. Lindstrom, M. Isenburg. “Fast and Efficient Compression of Floating-Point Data.” In: *IEEE Transactions on Visualization and Computer Graphics, 12(5):1245-1250* (2006) (cit. on p. 44).

[Mal99] S. Mallat. “A wavelet tour of signal processing.” In: *Academic Press* (1999) (cit. on p. 18).

[Pal] G. Palasky. “Gilles Palsky, Des chiffres et des cartes. Naissance et développement de la cartographie quantitative au XIXe siècle. Paris: Comit´e des Travaux Historiques et Scientifiques.” In: () (cit. on p. 12).

[RZE10] R. Rubinstein, M. Zibulevsky, M. Elad. “Double Sparsity: Learning Sparse Dictionaries for Sparse Signal Approximation.” In: *IEEE TRANSACTIONS ON SIGNAL PROCESSING, VOL. 58, Issue 3* (2010) (cit. on pp. 33, 34).

[Wika] *Discrete cosine transform*. https://en.wikipedia.org/wiki/Discrete_cosine_transform (cit. on p. 19).

[Wikb] *Discrete wavelet transform*. https://en.wikipedia.org/wiki/Discrete_wavelet_transform (cit. on p. 24).

[YCS07] C. S. Yoo, J. H. Chen, R. Sankaran. “Direct numerical simulation of a turbulent lifted hydrogen/air jet flame in heated coflow. In Computational Combustion.” In: *ECCOMAS Thematic Conference* (2007) (cit. on p. 37).

All links were last followed on January 18, 2017.

Declaration

I hereby declare that the work presented in this thesis is entirely my own and that I did not use any other sources and references than the listed ones. I have marked all direct or indirect statements from other sources contained therein as quotations. Neither this work nor significant parts of it were part of another examination procedure. I have not published this work in whole or in part before. The electronic copy is consistent with all submitted copies.

---

place, date, signature