Fast Random Access to Wavelet Compressed Volumetric Data Using Hashing

Flemming Friche Rodler
Rasmus Pagh

BRICS Report Series RS-01-34

ISSN 0909-0878 August 2001

Fast Random Access to Wavelet Compressed Volumetric Data Using Hashing

Flemming Friche Rodler and Rasmus Pagh
BRICS[*]
Department of Computer Science
University of Aarhus, Denmark
{ffr,pagh}@brics.dk

August, 2001

Abstract

We present a new approach to lossy storage of the coefficients of wavelet transformed data. While it is common to store the coefficients of largest magnitude (and let all other coefficients be zero), we allow a slightly different set of coefficients to be stored. This brings into play a recently proposed hashing technique that allows space efficient storage and very efficient retrieval of coefficients. Our approach is applied to compression of volumetric data sets. For the "Visible Man" volume we obtain up to 80% improvement in compression ratio over previously suggested schemes. Further, the time for accessing a random voxel is quite competitive.

1 Introduction

This paper considers the storage of volumetric or volume data, i.e., discrete collections of scalar or vector values sampled over a uniform grid in n-dimensional space. The dimension n is typically three or greater. Such data sets occur naturally in areas such as medical imaging and scientific visualization. Volumes produced by medical scanners such as CT, MR or PET are examples of three-dimensional data. Other examples are the output of physical simulations, and concentric mosaics used in image based rendering [38]. Sampled light fields [23] (or lumigraphs [12]) are examples of 4D volumes also used in image based rendering. Four-dimensional volumes also appear as time varying three-dimensional volumes in, for example, computational fluid dynamics [28, pp. 84-87, pp. 125-143].

The management and processing of such massive data sets present many challenges to developers and researchers. One problem is that these data sets are often too large to keep in internal memory in uncompressed form. For example, the "Visible Man" [39] CT scanned data set takes roughly 1 Gbyte of

[*]Basic Research in Computer Science (www.brics.dk), funded by the Danish National Research Foundation.

storage, and the most detailed anatomical photo data sets are orders of magnitude larger. Thus, we are faced with memory requirements far exceeding the typical memory available on ordinary PCs and workstations. Even taking the rapid development of larger memory and storage capabilities into account. On the other hand it is desirable and becoming increasingly important, for example in interactive visualization, that any part of the data set can be rapidly retrieved. This implicitly assumes that the data can be loaded into memory for efficient processing. A solution to these apparently conflicting goals is to consider compressed representations. Needed are methods allowing the user to load a compressed version of the volume into a small amount of memory and enable him to access and visualize it as if the whole uncompressed volume was present. Such a compression scheme must necessarily allow fast random access to individual voxels, decoded from the compressed volume. Lossless compression schemes only provide very low compression ratios. To do better, lossy techniques have to be considered. Since lossy compression removes information from the data, lossless compression has often been preferred in medical imaging. However, in [33] Perlmutter et al. demonstrated that lossy compression algorithms can be designed such that diagnostic accuracy is preserved. They used an embedded wavelet coding scheme with compression ratios of up to 80:1 for digital mammograms.

When designing lossy volume compression methods, certain properties are desirable and should be considered during design:

1. *Fast decoding for random access.* As described above fast decoding is a necessity for use in real-time or interactive applications. Also, applications often access data in unpredictable ways.

2. *Good visual fidelity at high compression ratios.* This seems obvious but it requires techniques for exploiting data redundancies in all n dimensions.

3. *Scalable or multiresolution decoding* is a desirable property which allows applications to process data at different levels of detail. For example, it could allow a rendering algorithm to render the data in low resolution at interactive frame-rates. When the user is satisfied with the setup, the renderer switches to full resolution.

4. *Selective block-wise decoding.* Even though our motivation is fast random access to individual voxels, some applications access data locally. In such cases it is useful if the compressed data can be decoded block-wise efficiently.

In this paper we present a new coding scheme for wavelet based compression of very large volume data with fast random access. It is based on a new approach to lossy storage of the coefficients of wavelet transformed data. While it is common to store the coefficients of the largest magnitude (and let all other coefficients be zero), we allow a slightly different set of coefficients to be stored. This brings into play a hashing technique recently proposed by the authors [32] in a different context. Our approach is applied to compression of volumetric data sets. For the "Visible Man" volume we obtain up to an 80% improvement in compression ratio over previously suggested schemes. Further, the time for accessing a random voxel is competitive with existing schemes.

The rest of this paper is organized as follows. In Section 3 we present the underlying wavelet theory used throughout the paper. Section 4 gives an overview of methods for storing sparse tables using hashing. Our coding scheme is described in detail in Section 5, and the results of our experiments appear in Section 6.

2 Related Work

Research in lossy compression has mainly focussed on lossy compression of still images or time sequences of images such as movies. The aim of these methods is to obtain the best compression ratio while minimizing the distortion in the reconstructed images. Often this limits the random accessibility, the reason being that most compression schemes employ variable-bitrate techniques such as Huffman (used in JPEG [17] and MPEG [18]) and Arithmetic coders [36], or differential encoders such as the Adaptive Differential Pulse Code coder [36]. However, such methods provide fast sequential decoding which is important in, for example, compression of images and video sequences.

However, techniques dealing with the issue of random access in volumetric data have been emerging. In [26, 27] Muraki introduced the idea of using wavelets to efficiently approximate three-dimensional data. The two-dimensional wavelet transform was extended to three-dimensions and it was shown how to compute the wavelet coefficients. By setting small coefficients to zero, Muraki showed that the shape of volumetric objects could be described in a relatively small number of three-dimensional functions. The motivation for the work was to obtain shape descriptions to be used in, for example, object recognition. No results on storage savings were reported. The potential of wavelets to reduce storage is evident, though.

Motivated by the need for faster visualization, a method for both compressing and visualizing three-dimensional data based on vector quantization was given by Ning and Hesselink [29]. The volume is divided into blocks of small size and the voxels in each block are collected into vectors. The vectors are then quantized into a codebook. Rendering by parallel projection is accelerated by preshading the vectors in the codebook and reusing precomputed block projections. Since the accessing of a single voxel is reduced to a simple table lookup in the codebook, fast random access is supported. Compressing two volumes both of size $128 \times 128 \times 128$, a compression factor of 5 was obtained with blocking and contouring artifacts being reported.

Burt and Adelson proposed the Laplacian Pyramid [7] as a compact hierarchical image code. This technique was extended to three dimensions by Ghavamnia and Yang [11] and applied to volumetric data. Voxel values can be accessed randomly by traversing the pyramid structure on the fly. Since there is a high computational overhead connected with the reconstruction, the authors suggest a cache structure to speed up reconstructions during ray casting. They achieve a compression factor of about 10 with the rendered images from the compressed volume being virtually indistinguishable from images rendered from the original data.

Several compression methods, both lossless and lossy, for compressing and transmitting Visible Human images were presented and compared by Thoma and Long [40]. Among the lossy schemes, which as expected outperformed the lossless ones in terms of compression ratio, the scheme based on wavelets performed best. The wavelet method that the authors suggest compresses the images individually and consists of three steps comprised of a wavelet transform followed by vector quantization with Huffman coding of the quantization indices. This makes it a traditional two-dimensional subband coder, and compression factors of 20, 40 and 60 were reported with almost no visible artifacts for a factor of 20. The coder does not allow for fast random access to individual voxels as it was mainly designed for storage and transmission purposes. Also, there is no exploitation of inter-pixel redundancies between adjacent slices.

Motivated by the need for efficient compression of medical data, several methods that exploit correlation in all three dimensions have been proposed in recent years [1, 2, 4, 5, 22, 24, 34, 41]. Three of these methods are lossless methods based on predictive coding [1, 2, 22] and provide compression ratios of about 2:1. A compression ratio of 2:1 does not allow large volumes to be loaded into main memory. The other methods are lossy and utilize a three-dimensional wavelet transform. The most recent of these are due to Bilgin et al. [5]. The method is a straightforward extension of zerotree coding proposed by Shapiro [37] and produces an embedded bit-stream which allows progressive reconstruction of data, i.e., it is possible to decode a lossy version of the data by using any prefix of the bit-stream. If the complete bit-stream is available, lossless reconstruction is possible. In order to improve performance, the zerotree symbols are further compressed using a context-based adaptive arithmetic coder. Compression ratios of up to 80 were reported, still with good fidelity. However, none of the methods support fast random access.

The methods mentioned above all have in common that they support either high compression ratios or fast random access, but not both. Recently, methods dealing with issues of both fast random access and high compression ratios have emerged. In [15, 16] Ihm and Park present an algorithm achieving compression ratios of up to 28 for CT slices of the Visible Human, still with good fidelity of the reconstructed slices for a ratio of about 15:1. The algorithm divides the volume into blocks and decomposes each block using a three-dimensional wavelet decomposition. The wavelet coefficients in each block are then adaptively thresholded, which produces a sparse representation. Each block is coded separately using a data structure that takes the spatial coherency of the wavelet coefficients into consideration. The method also supports selective block-wise decoding.

Recently, Bajaj et al. [3] described an extension of [16] to RGB color volumes. Again the volume is divided into blocks, and the wavelet transform is performed three times on each block, once for each color component. The three corresponding wavelet coefficients are then vector quantized. To speed up voxel reconstruction, they introduce what they call zerobit encoding, which essentially corresponds to using a significance map signaling if certain wavelet coefficients are zero, or if they have to be retrieved. This is similar to the significance map we describe in Chapter 5. Compared to an extension of [16] to color volumes (without zerobit encoding), Bajaj et al. reported a 10% to 15% increase in compression ratio and a speedup in voxel access of a factor of about 2.5. Selective block-wise decoding time is improved by a factor of 4 to 5.5 depending on the compression ratio.

A different approach was taken by Rodler in [35]. Instead of using a 3D wavelet transform, the method borrows good ideas from video coding. The volume is viewed as a "time sequence" of slices that is motion compensated using a simple prediction strategy. Finally, each motion compensated slice is transformed using a 2D wavelet transform. The transform coefficients are thresholded and then coded using a nested data structure. While the method offers significant coding gains over the results in [16], it is slightly slower. This method also supports selective block-wise decoding.

Not focusing directly on fast random access, Grosso et al. describe in [13] a wavelet based compression method for volumetric data suited for volume rendering. Each subband of wavelet transformed data is independently runlength encoded. The problem with runlength encoding is that it does not allow for efficient retrieval of a single coefficients. In order to accelerate rendering speed, a method that avoids starting from the beginning of the runlength sequence is suggested. For a triple (i, j, k) a lookup table indexed by (i, j) is generated. Each entry in the table points to a runlength encoding of the coefficients corresponding to k. If the renderer accesses the data in the k-direction, efficient rendering can be achieved. However, it is obvious that fast random access is not supported.

In [19] Kim and Shin present a similar approach for fast volume rendering. The main difference is the way coefficients are arranged into runs. They divide the volume into blocks that are coded separately. Each block is decomposed in a three-dimensional wavelet basis. The thresholded coefficients are then ordered according to the reconstruction order and run-length encoded. Compression ratios of up to 40:1 were reported with good visual fidelity at ratios up to about 30:1. Using the Shear Warp rendering algorithm of Lacroute and Levoy [21], they demonstrate that their algorithm using the compressed volume is only roughly two times slower, than using uncompressed data. Because of the run-length encoding, the algorithm does also not support fast random access to individual voxels. Selective block-wise decoding is supported, though no results were reported.

The four methods [3, 16, 19, 35] described above all make use of the Haar wavelet, which is convenient because of its simplicity. Since the mentioned methods all divide the data into blocks it is not clear, because of boundary conditions, how they generalize to wavelets with more vanishing moments, i.e., using wavelets with longer filters.

We conclude this section by noting that many of the results on volumetric compression in the literature are difficult to compare to one another since there is little consensus as to which data sets to use in presenting results. However, in volumetric compression with fast random access it seems that the CT images of the Visible Man is becoming a standard benchmark.

3 Wavelets

Wavelets have been applied successfully in many areas of computer graphics, such as pattern recognition, segmentation, compression, illumination models, visualization of volumes, and most state-of-the-art compression schemes. The ability of wavelets to efficiently approximate non-linear and non-stationary signals with only a few nonzero coefficients, leading to a compact representation, makes them very useful for compression purposes.

In this section we introduce the Haar wavelet. For a general description of wavelet theory we refer to [25]. The Haar wavelet decomposition of a discrete signal $A^0 = a^0[0]a^0[1]\ldots a^0[N-1]$ of length N is computed as follows. By taking averages of the $a^0[i]$ we obtain a new average signal $A^1 = a^1[0]a^1[1]\ldots a^1[N/2-1]$, with $a^1[i] = \frac{a^0[2i]+a^0[2i+1]}{\sqrt{2}}$. A^1 is half the length of A^0. Since A^1 is computed as an averaging and down-sampling over A^0, we can consider A^1 a representation of A^0 at a coarser resolution. By going from A^0 to A^1, fine details are lost. In order to recover A^0 from A^1 we must keep information about the lost details. The difference sequence $D^1 = d^1[0]d^1[1]\ldots d^1[N/2-1]$, with $d^1[i] = \frac{a^0[2i]-a^0[2i+1]}{\sqrt{2}}$ represents exactly this information. From A^1 and D^1 it is easy to see how A^0 can be recovered. The process of averaging can be applied recursively on the average signal A^i to obtain A^{i+1} and D^{i+1}. Since the length of A^{i+1} is half the length of A^i, there are at most $\log_2 N$ recursion steps (we assume N to be a power of 2 for convenience), and we end up with sequences $A^n, D^n, D^{n-1}, \ldots, D^1$. $A^n = a^n[0]$ is called the average coefficient, and the D^i are called the detail or wavelet coefficients. For compression purposes it is often not necessary to perform all $\log_2 N$ recursion levels; 3 to 5 levels are usually sufficient.

For many functions the wavelet transform is able to concentrate most of the energy in a small number of wavelet coefficients, with the rest of the coefficients being zero or close to zero. In the case of the Haar wavelet this can be seen from the fact that the wavelet coefficients correspond to sample-to-sample differences. In smooth regions these differences will be of small magnitude. This, together with the recursive averaging (smoothing), often leads to many small wavelet (detail) coefficients. By thresholding, i.e., setting all the small valued coefficients to zero, a very sparse representation can be obtained and exploited for compression purposes. For an orthonormal wavelet transform, thresholding of the coefficients corresponds to a globally optimal approximation in terms of the introduced *mean square error* (MSE) given by:

$$MSE = \frac{1}{N}\sum_k (x_k - \hat{x}_k)^2 = \frac{1}{N}\sum_k (y_k - \hat{y}_k)^2, \tag{1}$$

where the x_k and $\hat{x}_k$ are the original and reconstructed signals, and the y_k and $\hat{y}_k$ are the transform coefficients before and after thresholding. As a consequence, the exact error that occurs due to thresholding can be explicitly given.

The wavelet decomposition can be extended to 3D by performing the averaging and differing first along the x-axis, then along the y-axis, and finally along the z-axis. Figure 1 shows a three-level 3D wavelet decomposition. Each block is called a subband. The block marked a corresponds to the average (low pass) coefficients, while the other blocks correspond to the detail (high pass) coefficients at the seven different subbands. In order to reconstruct a voxel we need one coefficient from each subband. To compute one level of the Haar wavelet transform we use the following formulae

$$\begin{aligned}
a &= (c_1 + c_2 + c_3 + c_4 + c_5 + c_6 + c_7 + c_8)/2\sqrt{2} \qquad (2)\\
d^1 &= (c_1 + c_2 + c_3 + c_4 - c_5 - c_6 - c_7 - c_8)/2\sqrt{2}\\
d^2 &= (c_1 + c_2 - c_3 - c_4 + c_5 + c_6 - c_7 - c_8)/2\sqrt{2}\\
d^3 &= (c_1 + c_2 - c_3 - c_4 - c_5 - c_6 + c_7 + c_8)/2\sqrt{2}\\
d^4 &= (c_1 - c_2 + c_3 - c_4 + c_5 - c_6 + c_7 - c_8)/2\sqrt{2}\\
d^5 &= (c_1 - c_2 + c_3 - c_4 - c_5 + c_6 - c_7 + c_8)/2\sqrt{2}\\
d^6 &= (c_1 - c_2 - c_3 + c_4 + c_5 - c_6 - c_7 + c_8)/2\sqrt{2}\\
d^7 &= (c_1 - c_2 - c_3 + c_4 - c_5 + c_6 + c_7 - c_8)/2\sqrt{2}\ ,
\end{aligned}$$

where $c_1, \ldots, c_8$ are the eight coefficients in a $2 \times 2 \times 2$ subregion of the volume.

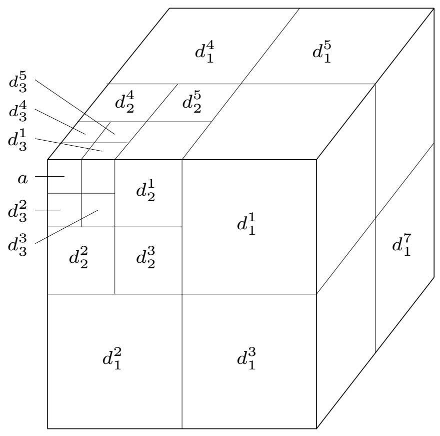


Figure 1: Subbands of a three-level three-dimensional wavelet transform. The subbands are numbered d_j^i, where j denotes the resolution level while i denotes the subband number within the level.

4 Storing Sparse Tables Using Hashing

In computer science one often needs to store a table that is only partially utilized, i.e., where most table cells are "empty". For sparsely used tables it is often not acceptable to allocate an array of full size, and compressed representations have to be considered. *Hashing* was first described by Dumey [9] as a method for space-efficient storage of sparse tables, allowing fast random access to original table elements. Since its conception as a heuristic, a good theoretical understanding of hashing techniques has been obtained, with breakthroughs in [8, 10], and sparse tables implemented in this way form an important part of many data structures.

The basic idea of hashing is to apply a function h, the *hash function*, to table indices. More specifically, one searches for the content of cell i in cell $h(i)$ of a "hash table" much smaller than the original sparse table. Hash table cells that store a non-empty cell from the sparse table also contain information indicating the index of the cell, typically the index itself. A good hash function will scatter the indices of non-empty cells such that few pairs of indices hash to the same value. Many techniques for dealing with collisions have been examined, as surveyed in [20, Section 6.4]. Some of the most space efficient of these are the so-called *open addressing* schemes, where a sequence of hash table cells $h_1(i), h_2(i), \dots, h_k(i)$ are probed until the content of cell i or an empty cell is found. Tables are filled in a greedy manner, by putting non-empty cells one by one into the first available hash table cell in the probe sequence. Though open addressing schemes are time efficient on average and widely used, the space overhead is too large for our applications. For example, an open addressing hash table for a sparse table with n non-empty cells uses at least $n \log n$ bits more than an optimal encoding. Also, $O(\log n)$ levels of hash functions are needed, i.e., it is expected that some indices take logarithmic time to retrieve.

Recently, very compact representations of static sparse tables have been proposed [6, 14, 30]. For a uniform distribution of non-empty table cells, these data structures use almost information theoretical minimal space. However, they are not practical for high-performance applications, due to a rather large, though constant, overhead in retrieving table elements. They are also somewhat intricate and difficult to implement. However, we will make use of the central technique described by Pagh in [30], called *quotienting.* Suppose that the sparse table has length p and the hash table length r. The basic idea is to start with a hash function $\phi : \{0, \dots, p-1\} \to \{0, \dots, p-1\}$ that is a *permutation*, and use $h(x) = \phi(x) \bmod r$ to map to the hash table indices. Assume that no pair of non-empty indices of the sparse table collide under h, so that the content of non-empty sparse table cell i can be stored in hash table cell $h(i)$. Then we need not store the index i, but only the integer quotient $\phi(i)$ div r to uniquely identify i, since i can be reconstructed from the quotient and $h(i)$, which is given by the position in the table. This uses $\lceil \log(p/r) \rceil$ bits, thereby saving about $\log(r)$ bits over the $\lceil \log(p) \rceil$ bits needed to store the key itself.

Pagh [31] recently introduced a variant of open addressing in which two hash tables T_1 and T_2 are used with two hash functions h_1 and h_2. If cell i in the sparse table is non-empty, it can be found in either hash table cell $h_1(i)$ of T_1 or cell $h_2(i)$ of T_2. Thus, only two table accesses are needed to look up an entry in the sparse table. A family of hash functions was given, from which suitable h_1 and h_2 can always be chosen. However, for a vast majority of problem instances it suffices to use somewhat simpler hash functions, for example of the form:

$$i \mapsto ((a\,i + b) \bmod p) \bmod r, \tag{3}$$

where p is a prime larger than any index, $0 < a < p$, $0 \le b < p$, and r is the desired range, i.e., the size of a hash table.

A drawback of the two table hashing scheme is that, in order to work, it needs hash tables of total size *twice* the number of non-empty sparse table cells, and is therefore quite redundant. This problem was addressed by Pagh and Rodler in [32], introducing a variant of the above scheme in which the hash tables are practically fully utilized. In fact, using quotienting the resulting space usage is *below* the information theoretical minimum for storing a (uniformly distributed) set of $2r$ elements. The price paid to achieve this is some loss of control concerning which sparse table indices are included. Roughly speaking, the scheme tries to store the most "important" indices, but will sometimes include less important indices to fill the table. Suppose we want to store $2r$ wavelet coefficients. The results of [32] show that, typically, all r coefficients of largest magnitude are stored, while about 85% of the $2r$ coefficients of largest magnitude are stored.

5 Coding Method

In designing our coding method we have to consider the trade-off between compression ratio, distortion, and speed of retrieval. As a consequence of its simple basis function, the Haar wavelet decomposition is simple to compute. In order to reconstruct a single data value, only one coefficient from each subband must be retrieved. Higher order wavelets, while promising better compression ratios, require the retrieval of significantly more coefficients. Since we focus on fast retrieval of voxels, we choose to use the Haar wavelet. We note, however, that our method easily generalizes to other wavelets. As we shall see in Section 6, the Haar wavelet provides good compression results.

We have chosen to use 3 decomposition levels. This provides a good trade-off between how sparse a representation the wavelet transform produces, and the number of coefficients needed to reconstruct a single voxel. We thus have one level of average coefficients and three levels of wavelet coefficients, each level consisting of seven subbands, as depicted in Figure 1. In order to reconstruct a voxel, we need 22 coefficients: 1 average coefficient, and 7 wavelet coefficients from each of the three detail levels. Most wavelet based compression schemes exploit the fact, mentioned in Section 3, that a very sparse representation can be obtained by thresholding the wavelet representation. After thresholding, the challenge is to efficiently store positions and values of the non-zero coefficients. This corresponds to storing the set W containing the n most important wavelet coefficients. Using the hashing scheme described at the end of Section 4, we will take a slightly different approach. We noted in Section 4 that the most space efficient methods for storing sparse tables have a rather high overhead for lookups, but that an index set W' "slightly different" from a given set W can be stored in a way allowing much more efficient lookups. By not storing exactly the n coefficients of largest magnitude, we get a larger MSE, but we will see that the increase is quite acceptable compared to the gain in compression ratio.

Since the sparseness of the wavelet coefficients change between subband levels, we will use a different set of hash tables for each level. For the moment, we will in fact pretend that each subband is stored as a separate sparse table. We later "merge" the hash tables of subbands at the same level. The main reason for choosing a three-level (instead of a two-level) wavelet transform is that the average coefficients take up a fraction of just 2^{-9} of the wavelet transformed volume, and can therefore with negligible overhead be stored in uncompressed form.

To use the sparse table data structure, we map (x, y, z) positions of a subband B to indices in the range $0, \ldots, |B| - 1$ in a straightforward way. Denote by $i_{x,y,z}$ the sparse table index corresponding to (x, y, z). We use pairwise independent pseudo-random permutations of the following form:

$$\phi(x, y, z) = (a\, i_{x,y,z} + b) \bmod p \ , \tag{4}$$

where p is the smallest prime greater than $|B|$, and a, b are chosen at random such that $0 < a < p$, and $0 \leq b < p$. For each sparse table, we have two such permutations ϕ_1 and ϕ_2, and take as our hash functions $h_1(x, y, z) = \phi_1(x, y, z) \bmod r$ and $h_2(x, y, z) = \phi_2(x, y, z) \bmod r$. To store the coefficient with index (x, y, z) in entry $h_1(x, y, z)$ of table one, we need only specify the quotient $\phi_1(x, y, z)$ div r and the uniformly quantized value of the coefficient (the same holds for hash table two). We choose the number of quantization levels to be a value such that the distortion introduced by quantization constitutes a minor part of the total distortion. This desired value is then adjusted slightly, such that the quantization index and the quotient fit into an integer number of bits. This defines the size of each entry in the hash tables. In case of an empty cell, we store the quotient $((a\,(p - 1) + b) \bmod p)$ div r, which does not match the quotient of any index (x, y, z).

For performance reasons we do not store the sparse tables exactly as described above. In particular, we store all subbands at each level in a single sparse table (using two hash tables), of correspondingly larger size, in such a way that wavelet coefficients that are needed for the same voxel reconstruction are more or less adjacent in the hash tables. This significantly improves the time for retrieving the coefficients, because of better cache usage. The merging of the tables is illustrated in Figure 2. At each subband level, we number the seven subbands $0, \ldots, 6$. The way in which wavelet coefficients are accessed at each subband level during a voxel reconstruction is very simple: The same index within each subband is needed. When hashing index (x, y, z) of the subband with number s we use the functions:

$$h'_i(x, y, z, s) = (\phi_i(x, y, z) + s) \bmod r, \tag{5}$$

for $i = 1, 2$. This gives corresponding indices in different subbands, at the same subband level, consecutive hash function values (modulo r). There was no significant degradation in MSE as a result of making hash function values correlated in this way. To be able to distinguish coefficients in different subbands, we pack the subband number together with the quotient.

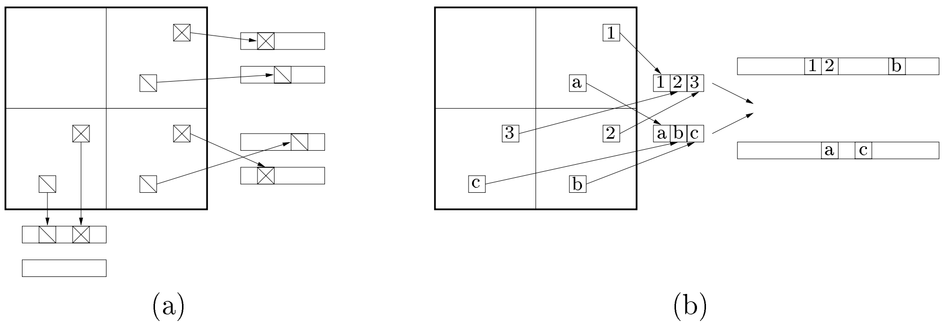


Figure 2: How tables are merged. (a) No merging – a separate sparse table is used for each subband. (b) All coefficient of a level is stored in one sparse table. Note that coefficients at the same spatial position get consecutive hash values.

The above describes how we store the wavelet coefficients, but not how to determine the size of the hash tables at different subband levels. This is non-trivial, as the best ratio between tables at different levels depends on the volume and the desired distortion. We use a simple heuristic, namely to perform global thresholding and use the number of non-zero coefficients at each level as the total size of the hash tables at that level. As we "nearly" store the coefficients remaining after thresholding, the MSE will be similar to the MSE introduced by the thresholding.

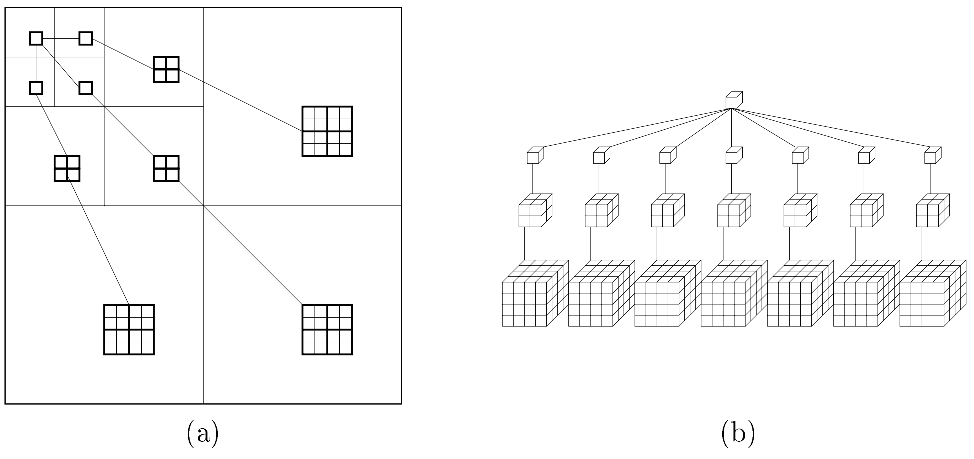


Figure 3: Wavelet coefficients corresponding to the same spatial location. (a) Illustrated in two dimensions. (b) In three dimensions we obtain an octree like structure. Note that, except for the top voxel, all voxels point to eight children. For clarity this is not shown.

While the scheme described this far is reasonably fast, a significant speedup can be obtained by exploiting the fundamental property of the stored wavelet coefficients: A very large fraction of coefficients is zero. Thus, in many cases only zero coefficients are found in the detail levels. To avoid such expensive lookups we store, together with each average coefficient, a “significance map” that can be consulted to avoid some lookups of detail coefficients that are zero. In particular, we divide subband levels 1 and 2 into $2 \times 2 \times 2$ blocks. If all seven blocks at the same spatial location in the different subbands of a level only have zero coefficients, we set a bit in the significance map of the corresponding average coefficient. This uses 8 bits for level 1 (at level 1 there are 8 blocks of size $2 \times 2 \times 2$ that correspond to the same spatial location as the average coefficient), and a single bit for level 2. Figure 3(a) shows, in two dimensions, the spatial relationship between the blocks at different subband levels. Finally, we set a bit to indicate whether all seven corresponding wavelet coefficients at level 3 are zero. In total, we use 10 bits for the significance map per average coefficient. Since the average coefficients only constitute a fraction of 2^{-9} of all wavelet coefficients this is negligible. The complete data structure used is illustrated in Figure 4. The illustration is in two dimensions for simplicity.

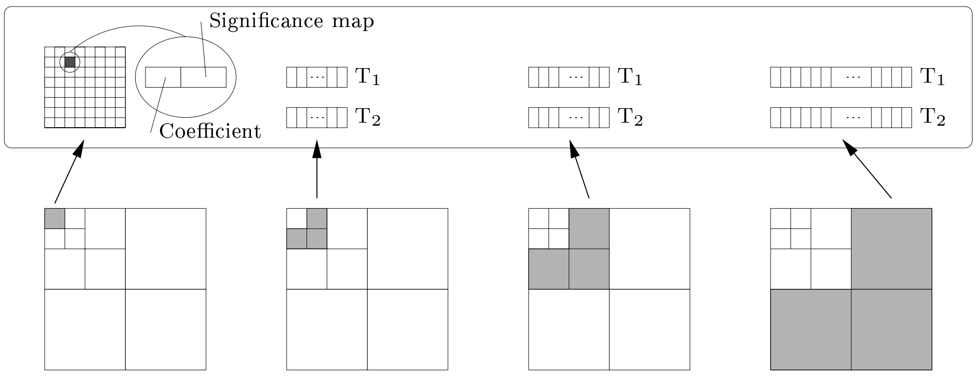


Figure 4: How the coefficients at different levels are stored in the data structure.

When reconstructing a voxel we first look up the average coefficient and the significance map. For each subband level, where the significance map does not declare all coefficients to be zero, we must then look up the seven coefficients in the sparse table of that level. Let r denote the size of the hash tables used for the sparse table, and let ϕ_1 and ϕ_2 denote the pseudo-random permutations used. If the position within the subbands, at a given subband level, is (x, y, z), we evaluate $\phi_1(x, y, z)$ and $\phi_2(x, y, z)$. Then, starting at position $\phi_1(x, y, z) \bmod r$ of hash table one and position $\phi_2(x, y, z) \bmod r$ of hash table two, we extract two sequences of seven consecutive quotients and subband numbers. If a quotient in the sequence from table i equals $\phi_i(x, y, z)$ div r, for some i, and the subband number equals the position in the sequence, the coefficient in that subband is nonzero, and its value is calculated from the quantized value in the table. Otherwise the coefficient is zero. When all the necessary wavelet coefficients have been extracted, the inverse Haar wavelet transform is applied.

The reconstruction formulas for one level are

$$\begin{aligned}
c_1 &= (a + d^1 + d^2 + d^3 + d^4 + d^5 + d^6 + d^7)/2\sqrt{2} \\
c_2 &= (a + d^1 + d^2 + d^3 - d^4 - d^5 - d^6 - d^7)/2\sqrt{2} \\
c_3 &= (a + d^1 - d^2 - d^3 + d^4 + d^5 - d^6 - d^7)/2\sqrt{2} \\
c_4 &= (a + d^1 - d^2 - d^3 - d^4 - d^5 + d^6 + d^7)/2\sqrt{2} \\
c_5 &= (a - d^1 + d^2 - d^3 + d^4 - d^5 + d^6 - d^7)/2\sqrt{2} \\
c_6 &= (a - d^1 + d^2 - d^3 - d^4 + d^5 - d^6 + d^7)/2\sqrt{2} \\
c_7 &= (a - d^1 - d^2 + d^3 + d^4 - d^5 - d^6 + d^7)/2\sqrt{2} \\
c_8 &= (a - d^1 - d^2 + d^3 - d^4 + d^5 + d^6 - d^7)/2\sqrt{2} \ .
\end{aligned} \tag{6}$$

Depending on the voxel that is to be reconstructed we must traverse an octree like structure from top to bottom, as illustrated in Figure 3(b), using one of the eight formulas at each node in the octree.

As discussed in Section 1, applications do not always access data totally at random. To enhance decoding efficiency when some locality in the access patterns exists, our decoding algorithm also supports selective block-wise decoding. When reconstructing a single voxel, seven wavelet coefficients must be retrieved for each subband level. These wavelet coefficients can be reused to compute neighboring voxels. In our selective block-wise decoding, we retrieve all 512 wavelet coefficients necessary to compute all voxels in a 8×8 block. To do this we must decode one average coefficient and all its associated detail coefficients as illustrated in Figure 3(b). From (6) it appears that 56 additions/subtractions must be performed to reconstruct eight voxels. However, by reusing computations this can be reduced to 24 operations.

6 Experiments

In this section we present results based on two data sets. The first data set is a $512 \times 512 \times 512$ CT data set of the upper body of the Visible Man. Each voxel is stored in 2 bytes, in which 12 bits are used, and the whole volume takes up 256 Mbytes. This data set is the same as in [15, 35] and was kindly provided to us by Ihm and Park. The second data set is a CT scan of an engine block, resampled to size $256 \times 256 \times 112$ to make all dimensions a multiple of 8. Each voxel is stored in 1 byte and the volume uses about 7 Mbytes of space. The original Engine Block data set was obtained from [43].

The exact set of wavelet coefficients that is stored during compression depends somewhat on the particular parameters of the pseudo-random permutations in Equation (4). For each compression ratio, we have compressed the volume three times, and for each subband level selected the permutation yielding the best results. Table 1 shows the results of compressing the Visible Man data set three times at a compression ratio of 44 : 1. The highlighted values in the table show the best MSE for each subband. Similar results were obtained for other compression ratios and also for the Engine Block data set. As mentioned in the previous section we uniformly quantize the wavelet coefficients.

Subband level	Experiment 1	2	3	Best
3	166.59	**162.49**	163.02	162.49
2	54.10	**53.23**	53.98	53.23
1	24.29	15.23	**13.57**	13.57
Total MSE	244.98	230.95	230.57	229.29
Total PSNR	48.84	48.61	48.62	48.64

Table 1: MSE for the three subband levels when using three different pseudo-random permutations.

The number of quantization levels is chosen such that the quantization induced distortion is a minor part of the total distortion. The number of bits used for coefficients at subband level 1 varied between 4.0 and 7.5 bits depending on the compression ratio. The exact number was chosen such that the hash quotient and the quantization level would fit into an integer number of bits. For the other subband levels we increase the disired number of bits used by 1.5 bits per level. This is necessary because of the $2/\sqrt{2}$ scaling of the wavelet coefficients in the wavelet transform (2).

6.1 Compression Ratio and Distortion

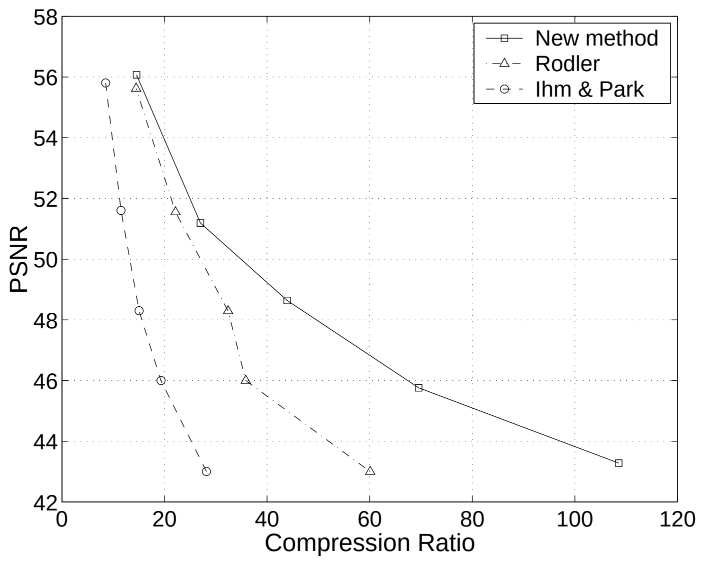


Figure 5: Rate-distortion curve for the Visible Man data set.

It is important that our compression scheme provides high compression ratios at low distortion, while maintaining fast random access. In our first experiment we compressed the two test volumes at various ratios. The results are summarized in Tables 2 and 3. Figures 5 and 6 show the *peak signal to*

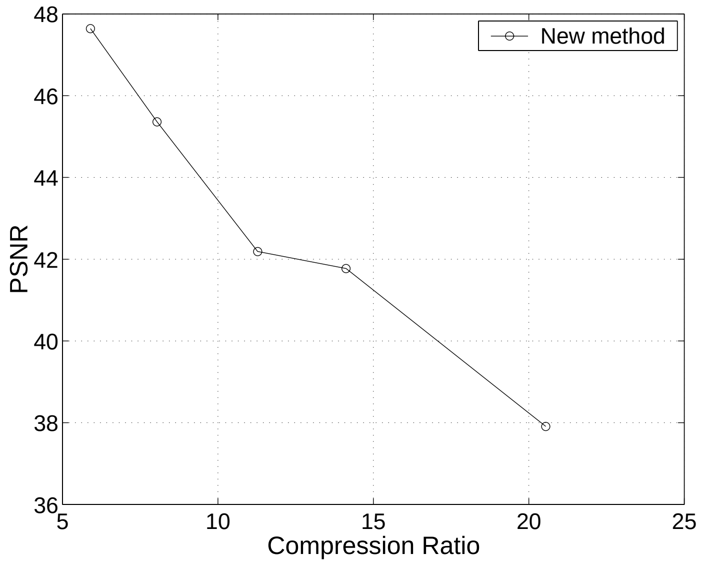


Figure 6: Rate-distortion curve for the Engine Block data set.

noise ratio (PSNR[1]) as a function of the compression ratio. The PSNR was calculated from the voxel differences between the original voxel values and the voxel values reconstructed from the compressed data. In Figure 5 we also show the rate-distortion performance of the methods in [15, 35]. As can be seen from the figure, our method gives strictly better performance in the distortion range investigated, and it significantly outperforms the previous best methods, nearly doubling the compression ratio compared to the method in [35] at high ratios. For low compression ratios it seems that the three methods converge. As mentioned in Section 2 an improvement to [15] has recently been proposed in [3]. The reported improvement was a 10%-15% increase in compression ratio for the RGB color volume of the Visible Human. Even if a similar increase holds for the Visible Man data set used in our experiments, our method still performs best, rate-distortion wise.

Figure 6 shows that the achieved compression ratios for the Engine Block data set are not as large as for the Visible Man data set. This can be explained by observing the percent of non-zero wavelet coefficients in Tables 2 and 3. We observe that the wavelet decomposition performs better on the Visible Man data set, in terms of obtaining a sparse representation for a given distortion, than on the Engine Block data set. This can partially be explained by noting that large areas of the Visible Man data set contains only zero valued voxels.

Even though the PSNR measure is an accepted measure in lossy compression visual inspection of the quality is very important. To that end Figures 9 and 10 shows sample slices for the various compression ratios. We observe that for the best PSNRs the reconstructed slices are virtually indistinguishable from the original. However, for a PSNR of 43 blocking artifacts and loss of small level

[1]PSNR $= 10\log_{10}\left(\frac{\max_i x[i]^2}{MSE}\right)$, $x[i]$ being the original data.

detail occur.

We also generated rendered images of the original and compressed volumes. The rendered images are depicted in Figures 11 to 14. For the Visible Man data set, the images at compression ratios of up to 43:1 are essentially indistinguishable from the original. However, the close ups in Figure 12 reveal beginning blocking artifacts. For the images at ratio 80:1 and 109:1 reconstruction artifacts become noticeable, however most of the features and details are still preserved. For the Engine Block data set, artifacts begin to appear at compression ratio 11:1 but not until a ratio of 21:1 do they become significantly noticeable. Again, when zooming the blocking artifacts appear sooner.

Compression	Ratio	109:1	80:1	44:1	27:1	15:1
	Size (Mb)	2.36	3.68	5.83	9.48	17.57
Errors	PSNR	43.28	45.76	48.64	51.19	56.07
	SNR	25.25	27.74	30.62	33.17	38.05
	MSE	788.7	445.4	229.3	127.5	41.4
Fraction of non-zero wavelet coefficients		0.96%	1.56%	2.60%	4.68%	8.24%

Table 2: Results on compression ratio and quality for for the Visible Man data set.

Compression	Ratio	20.6:1	14.1:1	11.3:1	8.0:1	5.9:1
	Size (Mb)	0.34	0.50	0.62	0.87	1.19
Errors	PSNR	37.91	41.77	42.19	45.36	47.64
	SNR	22.29	26.15	26.54	29.74	32.02
	MSE	10.53	4.32	3.95	1.89	1.12
Fraction of non-zero wavelet coefficients		3.60%	4.89%	6.79%	8.78%	11.06%

Table 3: Results on compression ratio and quality for the Engine Block data set.

6.2 Timing Results

To evaluate the time for accessing voxels from the compressed data, we measured the time it took to access 1,000,000 randomly selected voxels in the compressed volumes, using C's `rand()` function. These experiments were conducted on two different computers. The first computer used was a SGI Octane workstation with a 175 MHz R10000 CPU with a 32 Kbyte level 1 data cache and a 1 Mbyte level 2 cache. This machine is very similar (except maybe differences in the amount of installed memory and cache) to the one used in [15, 35], so direct comparison is possible. Programs were compiled with the `cc` compiler version 7.3.1.1m, using optimization flags `-Ofast=IP30`. The second computer used was a PC with an 800 MHz Intel Pentium III processor with 16 Kbyte level 1 data cache and 256 Kb level 2 cache. On this machine programs were compiled with `gcc` version 2.95.2, using optimization flags `-O9 -fomit-frame-pointer -fno-rtti`. Both machines were equipped with 256 Mbytes of main memory. Results are listed in Tables 4 and 5 with and without the significance map optimization discussed in the previous section.

CPU	Significance map	Uncompressed	Compression Ratio				
			109:1	80:1	44:1	27:1	15:1
PC	Yes	0.50	1.61	1.82	2.09	2.44	2.86
	No		4.73	4.84	4.94	4.82	4.95
SGI	Yes	1.65 – 3.03	3.79	4.63	5.68	7.13	8.67
	No		14.88	15.78	16.30	16.70	17.00

Table 4: Voxel reconstruction times in seconds for the Visible Man data set.

CPU	Significance map	Uncompressed	Compression Ratio				
			20.6:1	14.1:1	11.3:1	8.0:1	5.9:1
PC	Yes	0.49	2.10	2.47	2.70	3.14	3.45
	No		4.51	4.64	4.73	4.76	4.97
SGI	Yes	1.17	5.28	6.25	7.09	8.26	8.86
	No		12.93	12.97	13.06	13.12	13.27

Table 5: Voxel reconstruction times in seconds for the Engine Block data set.

As a reference, we also list in the tables the time it takes to access 1,000,000 randomly selected voxels from the uncompressed data. For the Engine Block data set, we used the original volume. However, because the Visible Man data set could not fit into main memory on the two computers used we simulated accessing voxels in uncompressed data by allocating a piece of memory accessing entries of this piece at random. For the PC we noted that the time it took to randomly access 1,000,000 elements varied very little with the size of the allocated piece of memory, as long as it was significantly larger than the cache. However, for the SGI we found that there is a linear dependence between the size of the allocated memory and the time it takes to access the random voxels. For a piece of memory varying between 20%–80% of the size of the Visible Man volume, the time it took to randomly access 1,000,000 voxels varied between 1.65 and 3.03 seconds.

From tables 4 and 5, we observe that accessing random voxels from the Visible Man data set on a PC using the significance map is about 3 to 6 times slower than accessing uncompressed data. Running on the SGI, it is more difficult to evaluate how much slower it is to access compressed data because of the mentioned variation when accessing uncompressed data. For the Engine Block data set, accessing compressed data on both computers is about 4.5 to 7.5 times slower. Figure 7 shows the time for accessing 1,000,000 random voxels in compressed data on the SGI as a function of the PSNR level. From the figure, we observe that our method outperforms the schemes in [15, 35]. For high distortion (small PSNR), our method is about twice as fast as [15]. It should be noted that a speedup of about a factor of 2.5 compared to Ihm and Parks method [15] has recently been obtained in [3] for color volumes. This speedup, if it holds for the Visible Man data set, results in a method faster than ours, especially at high PSNR levels. However, our method achieves significantly higher compression ratios for the same PSNR level.

Observe that accessing voxels from highly compressed data is faster than accessing voxels from less compressed data. The main reason for this is that at high compression ratios fewer wavelet coefficients are kept, resulting in a more efficient significance map because of more zero coefficients. Furthermore, as the compression ratio increases, more of the compressed data might fit into the data cache of the computer, which again results in a speedup. This last type of speedup can be noticed in the timings where the significance map was disabled. From Tables 4 and 5 we see that enabling the use of the significance map gives a speedup of about 2-4 times for the Visible Man data set compared to having it disabled. The improvement is largest at high compression ratios where the significance map is more effective because of more zeros. For the Engine Block data set, the speedup is slightly smaller. The reason for this, as discussed above, is that the Visible Man data set contains more voxels with value zero.

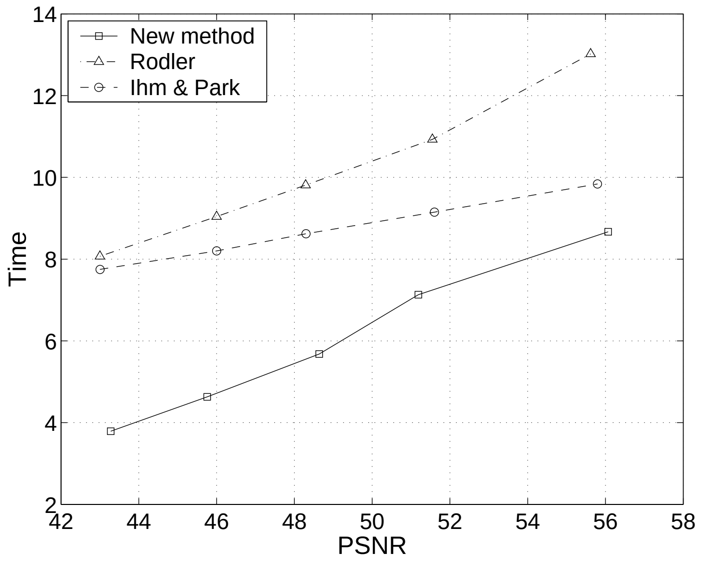


Figure 7: Time for accessing 1,000,000 voxels in compressed data for the Visible Man data set on the SGI.

6.3 Selective Block-wise Decoding

As mentioned in Section 1, not all applications access data completely at random. To improve decoding efficiency when data is accessed locally, our method supports selective block-wise decoding. As mention in Section 5, the voxels in an $8 \times 8 \times 8$ block can be decoded efficiently by reusing both wavelet coefficients and computions in the reconstruction formulas. The effectiveness of selective block-wise decoding was evaluated by decoding the entire volume block by block. The timings are shown for the two data sets in Tables 6 and 7. We also experimented with how long it takes to access the uncompressed volumes block by block. This was done by using 6 nested loops. The first three loops indexed the blocks, while the last three loops accessed voxels within each block. As the Visible Man data set could not fit into the memory of the computers used, we simulated block-wise access by accessing a $256 \times 512 \times 512$ array (half the size of the data set), multiplying the time it took by two. On the PC we obtained different times for the Visible Man data set depending on whether the loops were arranged in a cache friendly way or not. This effect was not observed for the SGI. We believe that the SGI compiler optimizes for such effects, as using lesser optimization options resulted in significantly longer timings. From tables 6 and 7, we observe that decoding the whole volume block-wise is significantly slower than accessing uncompressed data. However, considering the memory requirements needed for large uncompressed volumes we think this overhead is quite acceptable. Also, applications such as some volume renderers require orders of magnitudes longer processing time. From the tables we also note that as the compression ratio decreases, the decoding time increases. This can be explained by the fact that the significance map is less effective at low ratios. Finally, in Figure 8, we compare our method to the methods in [15, 35] for the Visible Man data set. For a PSNR level below 50 our method performs best. However, if the significance map improvement in [3] is able to improve the results of [15] by a factor of 4-5.5 this becomes by far the fastest method at high PSNR levels.

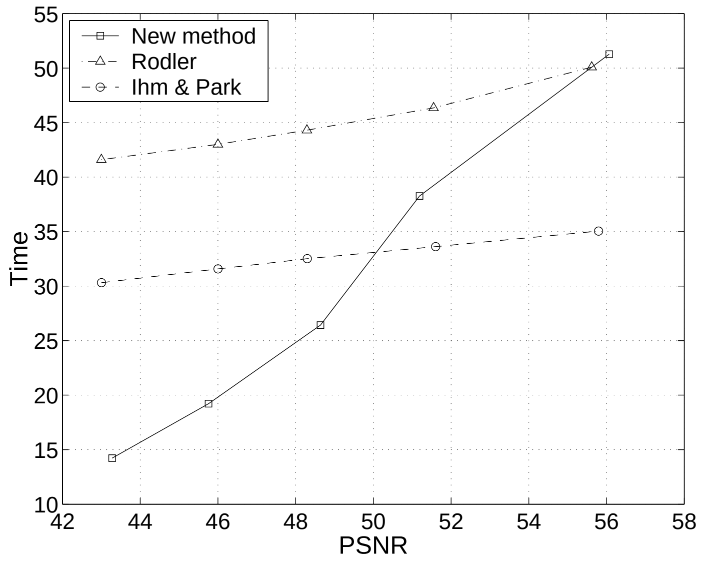


Figure 8: Selective block-wise decoding for the Visible Man data set on the SGI.

CPU	Uncompressed	Compression Ratio				
		109:1	80:1	44:1	27:1	15:1
PC	1.48–2.20	6.24	7.75	10.06	13.12	17.17
SGI	3.04	14.22	19.22	26.42	38.27	51.28

Table 6: Selective block-wise reconstruction times in seconds for the Visible Man data set.

CPU	Uncompressed	Compression Ratio				
		20.6:1	14.1:1	11.3:1	8.0:1	5.9:1
PC	0.08	0.59	0.71	0.85	1.08	1.33
SGI	0.15	1.46	1.67	2.08	2.65	3.28

Table 7: Selective block-wise reconstruction times in seconds for the Engine Block data set.

7 Multiresolutional Decoding

While relatively fast, the decoding process might still be too slow for interactive applications. In some applications it is possible to work with a low resolution representation of the volume. Volume rendering is such an example. When the user is satisfied with the setup at the low resolution, the application switches to full resolution producing the final result. Since our method uses a three-dimensional wavelet transform, it supports multiresolutional decoding in a natural way. For example, by assuming that the wavelet coefficients at subband level 3 are all zero we can omit the last reconstruction step. This results in a volume half the size of the original volume in each dimension. By omitting coefficients at subband levels 2 and 1, our algorithm can return voxels from a volume that is downsampled by a factor of 2, 4, or 8 in each direction. Since the algorithm does not need to decode coefficients assumed to be zero, this results in a speedup. In Table 8 and Table 9 we present results on decoding randomly selected voxels from the two compressed volumes at different resolutions. As expected, the reconstruction time of a voxel decreases as the downsampling factor increases. When downsampling 8 times we notice that the access time does not depend on the compression ratio. This is because we access only the uncompressed average coefficients, and these do not change with the compression ratio. Accessing and decoding the average coefficients take slightly longer than accessing uncompressed data since we still need to apply the scaling in the reconstruction formulas (6).

CPU	Downsampling factor	Uncompressed	Compression Ratio				
			109:1	80:1	44:1	27:1	15:1
PC	1	0.50	1.61	1.82	2.09	2.44	2.86
	2		1.49	1.63	1.82	2.07	2.34
	4		1.18	1.26	1.37	1.50	1.59
	8		0.81	0.81	0.81	0.81	0.81
SGI	1	1.65 – 3.03	3.79	4.63	5.68	7.13	8.67
	2		3.32	3.93	4.68	5.55	6.50
	4		1.18	1.26	1.37	1.50	1.59
	8		1.20	1.20	1.20	1.20	1.20

Table 8: Voxel reconstruction times in seconds for the Visible Man data set when accessing voxels at lower resolutions.

CPU	Downsampling factor	Uncompressed	Compression Ratio				
			20.6:1	14.1:1	11.3:1	8.0:1	5.9:1
PC	1	0.50	2.10	2.47	2.70	3.14	3.45
	2		1.80	2.08	2.20	2.47	2.61
	4		1.30	1.42	1.44	1.42	1.44
	8		0.73	0.73	0.73	0.73	0.73
SGI	2	1.65 – 3.03	5.28	6.25	7.09	8.26	8.86
	2		4.48	5.28	5.83	6.54	6.55
	4		2.82	3.23	3.20	3.20	3.22
	8		1.15	1.15	1.15	1.15	1.15

Table 9: Voxel reconstruction times in seconds for the Engine Block data set when accessing voxels at lower resolutions.

8 Conclusion and Final Remarks

We have presented a new method for compression of volumetric data that supports fast random access to individual voxels within the compressed data. The method is based on a wavelet decomposition of the data, and a new technique for storing sparse tables using hashing. The new hashing technique may be of independent use in other compression applications. Results comparing our new compression method to other techniques show an improvement over the entire distortion range investigated. For high ratios, the improvement in ratio is 50% to 80%. We also reported competitive access times for our method.

One interesting idea for future research in lossy compression is the concept of not storing the most significant wavelet coefficients, but a slightly different set of coefficients. This might yield a combined rate-distortion improvement when used in combination with existing techniques. The idea has been mentioned before by Xiong et al. [42] but it has still not attracted the attention we think it deserves.

References

[1] B. Aiazzi, P. Alba, S. Baronti, and L. Alparone. Threedimensional lossless compression based on a separable generalized recursive interpolation. *Proceedings of the 1996 IEEE International Conference on Image Processing*, pages 85–88, 1996.

[2] Bruno Aiazzi, Pasquale Alba, Luciano Alparone, and Stefano Baronti. Lossless compression of multi/hyper-spectral imagery based on a 3-D fuzzy prediction. *IEEE Transactions on Geoscience and Remote Sensing*, 37(5):2287–2294, 1999.

[3] Chandrajit Bajaj, Insung Ihm, and Sanghun Park. 3D RGB image compression for interactive applications. *ACM Transactions on Graphics*, 20, March 2001. To appear.

[4] Atilla M. Baskurt, Hugues Benoit-Cattin, and Christophe Odet. "a 3-d medical image coding method using a separable 3-d wavelet transform. In

Yongmin Kim, editor, *Medical Imaging 1995: Image Display*, 2431, pages 184–194. SPIE, 1995.

[5] A. Bilgin, G. Zweig, and M. W. Marcellin. Three-dimensional image compression using integer wavelet transforms. *Applied Optics: Information Processing, Special Issue on Information Theory in Optoelectronic Systems*, 39:1799–1814, April 2000.

[6] Andrej Brodnik and J. Ian Munro. Membership in constant time and almost-minimum space. *SIAM J. Comput.*, 28(5):1627–1640 (electronic), 1999.

[7] Peter J. Burt and Edward H. Adelson. The laplacian pyramid as a compact image code. *IEEE Trans. on Communications*, 31(4):532–540, April 1983.

[8] J. Lawrence Carter and Mark N. Wegman. Universal classes of hash functions. *J. Comput. System Sci.*, 18(2):143–154, 1979.

[9] Arnold I. Dumey. Indexing for rapid random access memory systems. *Computers and Automation*, 5(12):6–9, 1956.

[10] Michael L. Fredman, János Komlós, and Endre Szemerédi. Storing a sparse table with $O(1)$ worst case access time. *J. Assoc. Comput. Mach.*, 31(3):538–544, 1984.

[11] Mohammad H. Ghavamnia and Xue D. Yang. Direct rendering of laplacian pyramid compressed volume data. *Proceedings of Visualization '95*, pages 192–199, October 1995.

[12] Steven J. Gortler, Radek Grzeszczuk, Richard Szeliski, and Michael F. Cohen. The lumigraph. *Computer Graphics, Proceedings of SIGGRAPH '96*, pages 43–54, 1996.

[13] Roberto Grosso, Thomas Ertl, and Joachim Aschoff. Efficient data structures for volume rendering of wavelet-compressed data. *WSCG '96 - The Fourth International Conference in Central Europe on Computer Graphics and Visualization*, February 1996.

[14] Torben Hagerup and Torsten Tholey. Efficient minimal perfect hashing in nearly minimal space. In *Proceedings of the 18th Symposium on Theoretical Aspects of Computer Science (STACS '01)*, pages 317–326. Springer-Verlag, Berlin, 2001.

[15] Insung Ihm and Sanghun Park. Wavelet-based 3D compression scheme for very large volume data. In *Graphics Interface*, pages 107–116, June 1998.

[16] Insung Ihm and Sanghun Park. Wavelet-based 3D compression scheme for interactive visualization of very large volume data. *Computer Graphics Forum*, 18(1):3–15, March 1999.

[17] ISO/IEC JTC1 and ITU-T. Digital compression and coding of continuous-tone still images. *ITU-T Recommendation T.81 – ISO/IEC 10918-1 (JPEG)*, 1992.

[18] ISO/IEC JTC1 and ITU-T. Information technology - generic coding of moving pictures and associated audio information - part 2: Video. *ITU-T Recommendation H.262 – ISO/IEC 13818-2 (MPEG-2)*, 1994.

[19] Tae-Young Kim and Yeong Gil Shin. An efficient wavelet-based compression method for volume rendering. In *Seventh Pacific Conference on Computer Graphics and Applications*, pages 147–156, October 1999.

[20] Donald E. Knuth. *Sorting and Searching*, volume 3 of *The Art of Computer Programming*. Addison-Wesley Publishing Co., Reading, Mass., second edition, 1998.

[21] Philippe Lacroute and Marc Levoy. Fast volume rendering using a shear-warp factorization of the viewing transformation. In *Proceedings of SIGGRAPH 94*, pages 451–458, 1994.

[22] K. Lau, W. Vong, and W. Ng. Lossles compression of 3-d images by variable predictive coding. *Proceedings of 2nd Singapore International Conference on Image Processing, 1992, pp. 161–165.*, 1992.

[23] Marc Levoy and Pat Hanrahan. Light field rendering. *Computer Graphics Proceedings, Annual Conference Series (SIGGRAPH '96)*, pages 31–42, 1996.

[24] Jiebo Luo, Xiaohui Wang, Chang W. Chen, and Kevin J. Parker. Volumetric medical image compression with three-dimensional wavelet transform and octave zerotree coding. In Rashid Ansari and Mark J. Smith, editors, *Visual Communications and Image Processing '96*, 2727, pages 579–590. SPIE, 1996.

[25] Stéphane Mallat. *A Wavelet Tour of Signal Processing, 2nd Edition*. Academic Press, 1999.

[26] Shigeru Muraki. Approximation and rendering of volume data using wavelet transforms. *Proceedings of Visualization '92*, pages 21–28, October 1992.

[27] Shigeru Muraki. Volume data and wavelet transforms. *IEEE Computer Graphics & Applications*, 13(4):50–56, July 1993.

[28] Gregory M. Nielson, Hans Hagen, and Heinrich Müller, editors. *Scientific Visualization: Overviews, Methodologies, and techniques*. IEEE Computer Society, 1997.

[29] Paul Ning and Lambertus Hesselink. Fast volume rendering of compressed data. *Proceedings of Visualization '93*, pages 11–18, October 1993.

[30] Rasmus Pagh. Low redundancy in static dictionaries with $O(1)$ lookup time. In *Proceedings of the 26th International Colloquium on Automata, Languages and Programming (ICALP '99)*, volume 1644 of *Lecture Notes in Computer Science*, pages 595–604. Springer-Verlag, Berlin, 1999.

[31] Rasmus Pagh. On the cell probe complexity of membership and perfect hashing. In *Proceedings of the 33rd Annual ACM Symposium on Theory of Computing (STOC '01).* ACM Press, New York, 2001. To appear.

[32] Rasmus Pagh and Flemming Friche Rodler. Lossy dictionaries. Manuscript, 2001.

[33] S.M. Perlmutter, P.C. Cosman, R.M. Gray, R.A. Olshen, D. Ikeda, C.N. Adams, B.J. Betts, M. Williams, K.O. Perlmutter, J. Li, A. Aiyer, L. Fajardo, R. Birdwell, and B.L. Daniel. Image quality in lossy compressed digital mammograms. *Signal Processing*, 52(2):189–210, 1997.

[34] Michael A. Pratt, Chee-Hung H. Chu, and Stephen T. Wong. Volume compression of MRI data using zerotrees of wavelet coefficients. In Michael A. Unser, Akram Aldroubi, and Andrew F. Laine, editors, *Wavelet Applications in Signal and Image Processing IV*, 2431, pages 752–763. SPIE, 1996.

[35] Flemming Friche Rodler. Wavelet based 3D compression with fast random access for very large volume data. In *Seventh Pacific Conference on Computer Graphics and Applications*, pages 108–117, Seoul, Korea, October 1999.

[36] Khalid Sayood. *Introduction to Data Compression.* Morgan Kaufmann Publishers, Inc., USA, 1996.

[37] Jerome M. Shapiro. Embedded image coding using zerotress of wavelet coefficients. *IEEE Trans. on Signal Processing*, 41(12):3445–3462, December 1993.

[38] Heung-Yeung Shum and Li-Wei He. Rendering with concentric mosaics. *Computer Graphics Proceedings, Annual Conference Series (SIGGRAPH '99)*, pages 299–306, 1999.

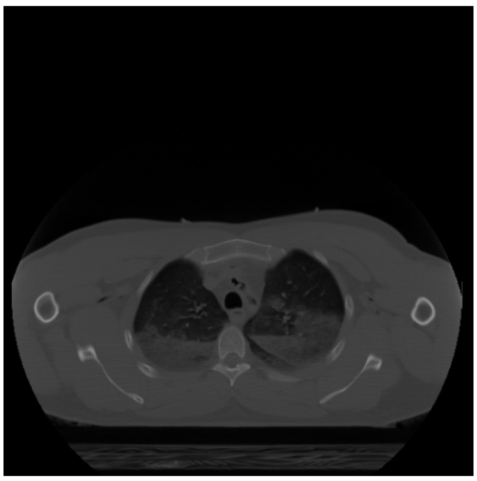
[39] The visible human project. NLM homepage - http://www.nlm.nih.gov/research/visible/visible_human. html.

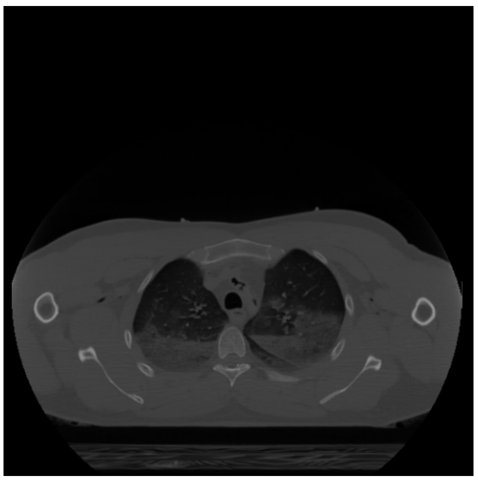
[40] Geoge R. Thoma and L. Rodney Long. Compressing and transmitting visible human images. *IEEE Multimedia*, 4(2):36–45, 1997.

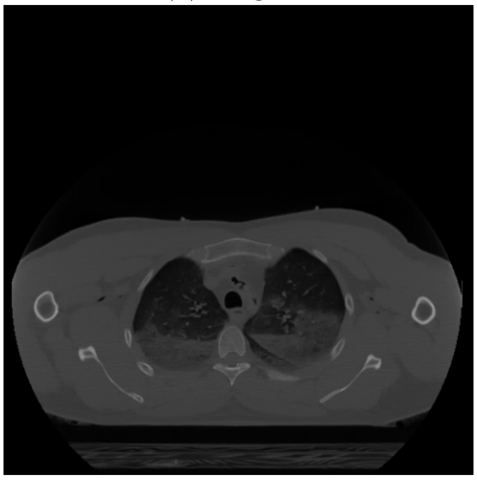
[41] J. Wang and H. Huang. Medical image compression by using threedimensional wavelet transformation. *IEEE Transactions on Medical Imaging*, 15:547–554, August 1996.

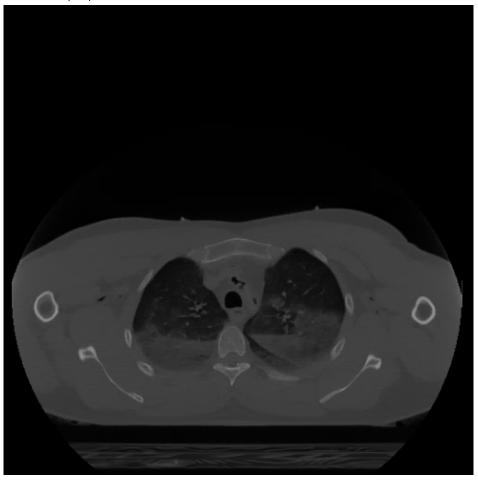
[42] Xixiang Xiong, Kannan Ramchandran, and Michael T. Orchard. Space-frequency quantization for wavelet image coding. *IEEE Transactions on Image Processing*, 6(5):677–693, May 1997.

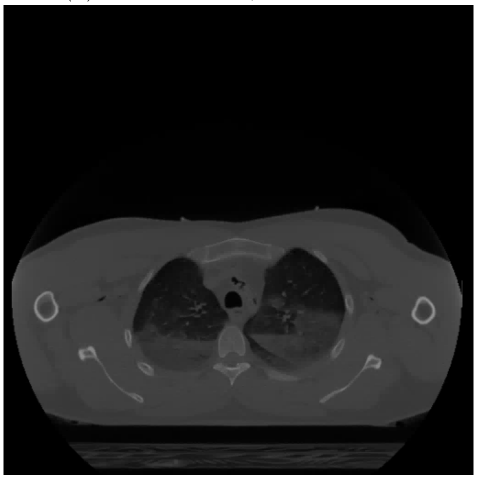
[43] *The Engine Block dataset.* http://www9.informatik.uni-erlangen.de/~cfrezksa/volren/.


(a) Original


(b) Ratio: 15:1, PSNR: 56.1


(c) Ratio: 27:1, PSNR: 51.2


(d) Ratio: 44:1, PSNR: 48.6


(e) Ratio: 80:1, PSNR: 45.8

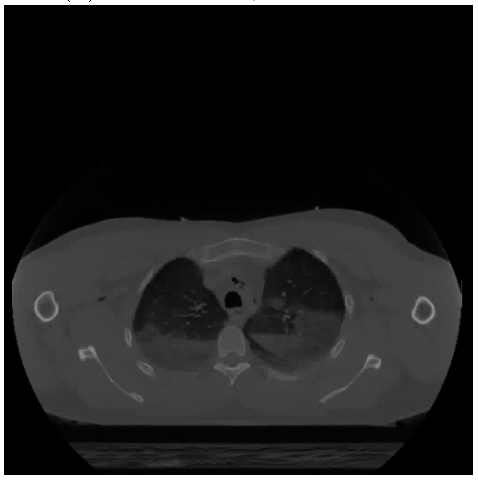
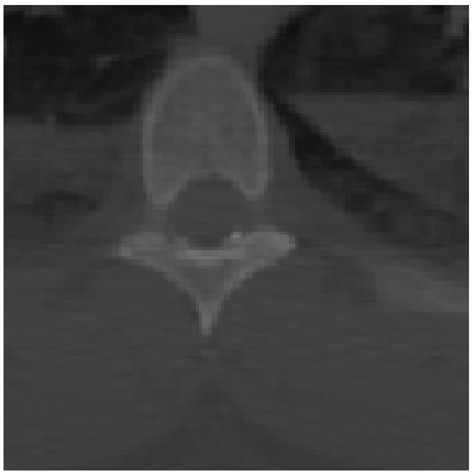
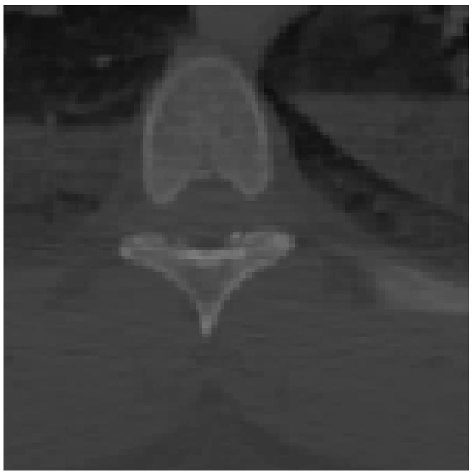
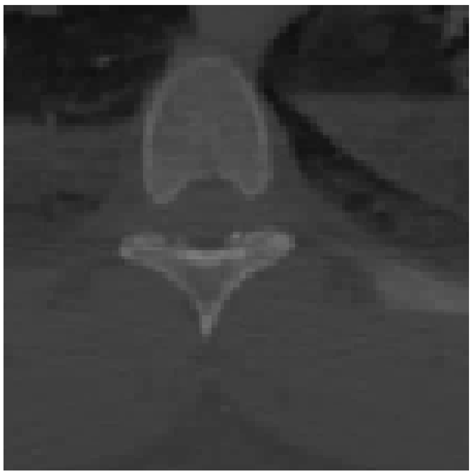
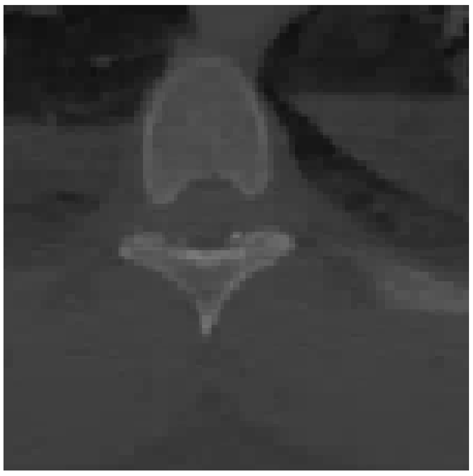

(f) Ratio: 109:1, PSNR: 43.3

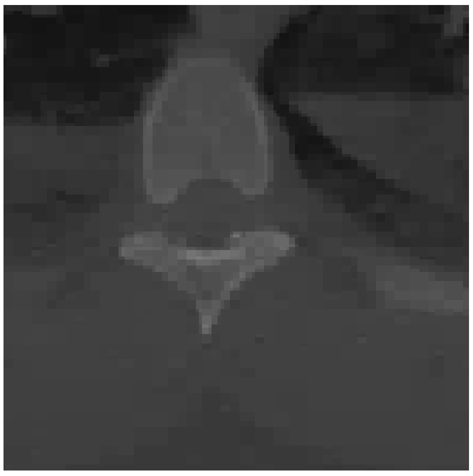
Figure 9: Sample slice (no. 345) of the Visible Man data set for various compression ratios.


(a) Original


(b) Ratio: 15:1, PSNR: 56.1


(c) Ratio: 27:1, PSNR: 51.2


(d) Ratio: 44:1, PSNR: 48.6


(e) Ratio: 80:1, PSNR: 45.8

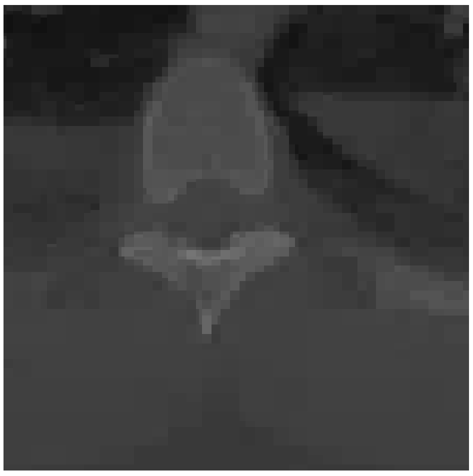
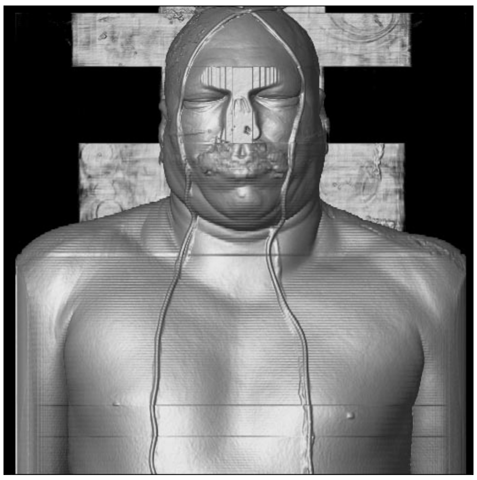
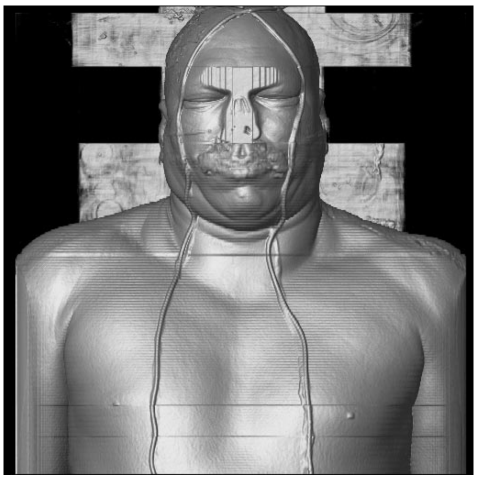
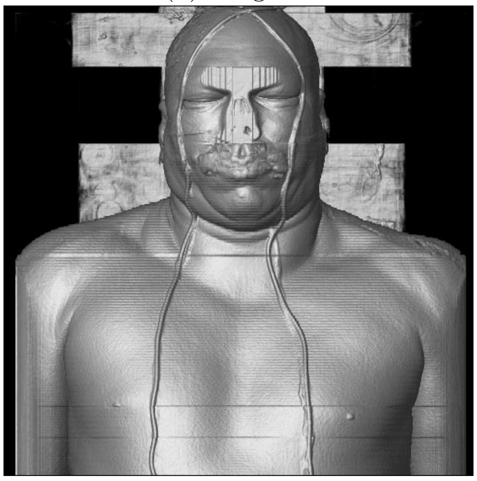
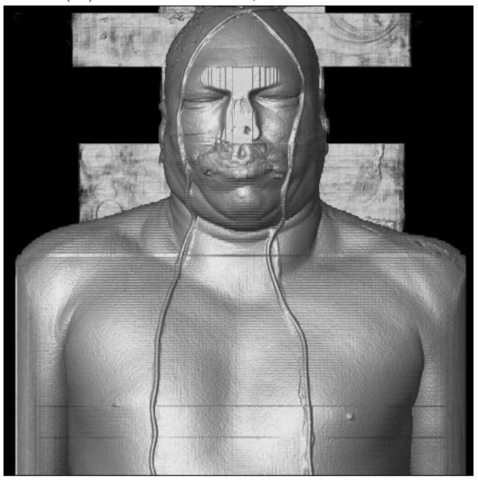

(f) Ratio: 109:1, PSNR: 43.3

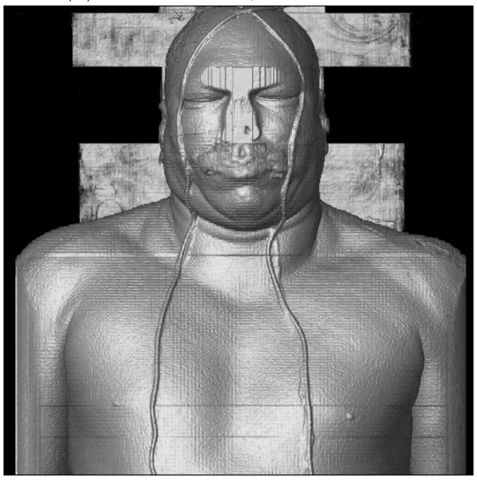
Figure 10: Zoom of sample slice (no. 345) of the Visible Man data set for various compression ratios.


(a) Original


(b) Ratio: 15:1, PSNR: 56.1


(c) Ratio: 27:1, PSNR: 51.2


(d) Ratio: 44:1, PSNR: 48.6


(e) Ratio: 80:1, PSNR: 45.8

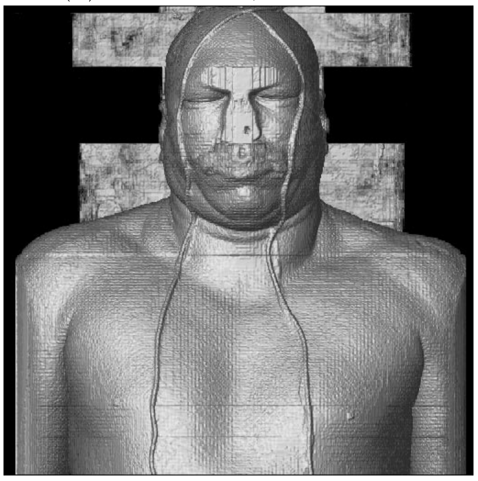
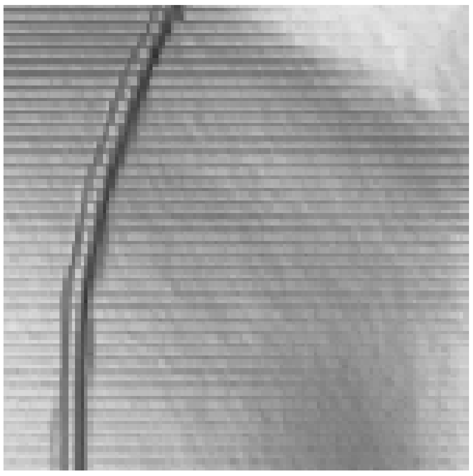
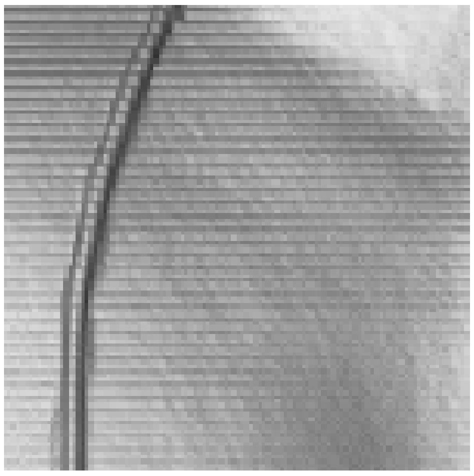
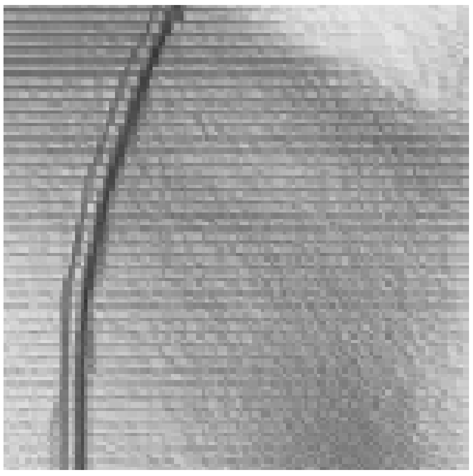
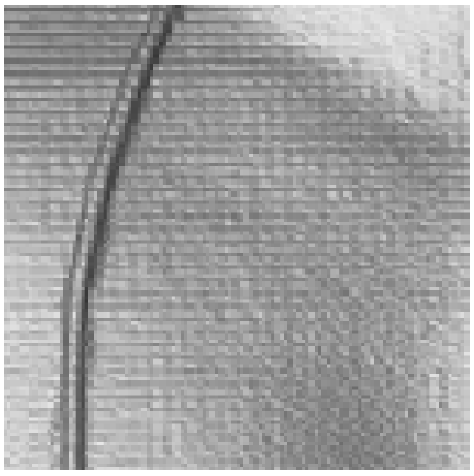

(f) Ratio: 109:1, PSNR: 43.3

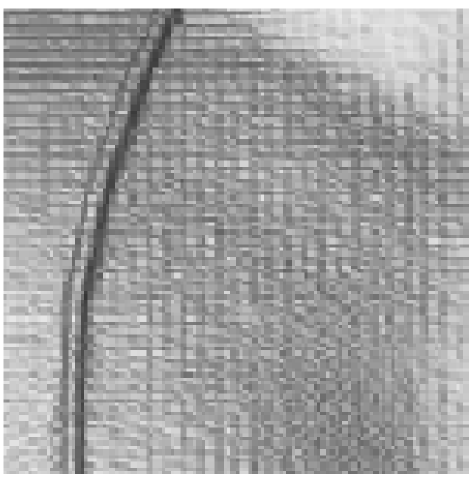
Figure 11: Rendered images of the Visible Man data set for various compression ratios.


(a) Original


(b) Ratio: 15:1, PSNR: 56.1


(c) Ratio: 27:1, PSNR: 51.2


(d) Ratio: 44:1, PSNR: 48.6


(e) Ratio: 80:1, PSNR: 45.8

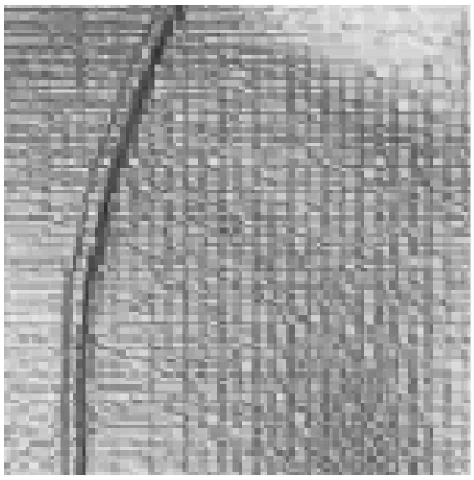
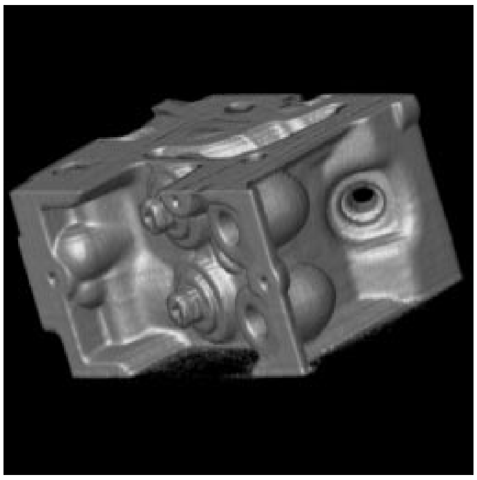
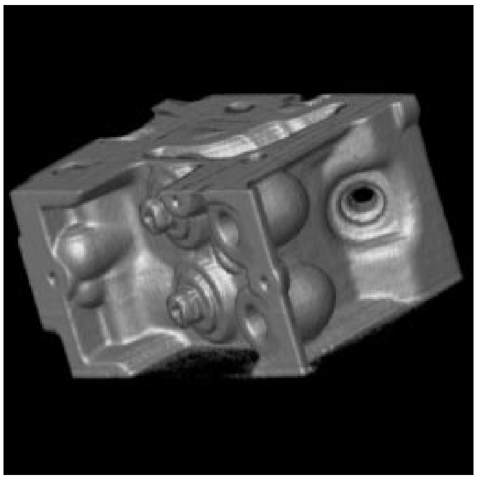

(f) Ratio: 109:1, PSNR: 43.3

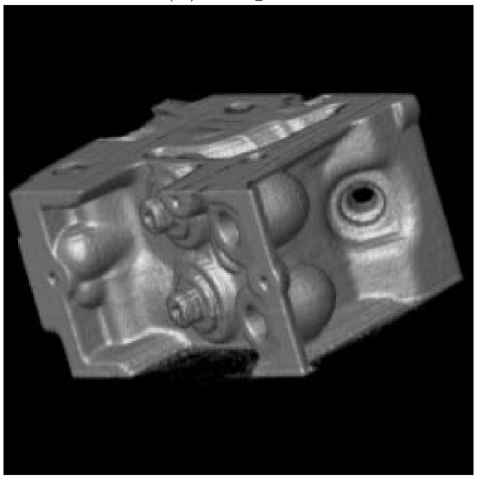
Figure 12: Zoom of rendered images of the Visible Man data set for various compression ratios.



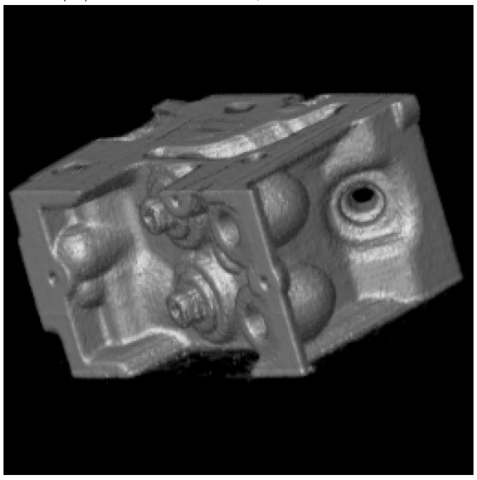
(a) Original



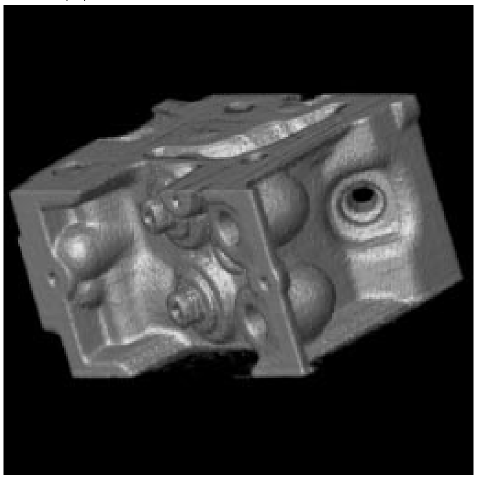
(b) Ratio: 5.9:1, PSNR: 47.6



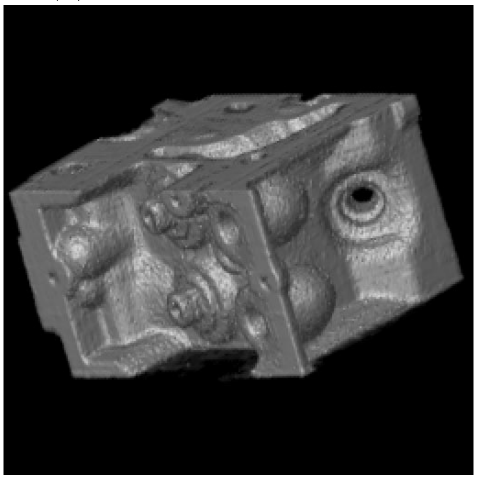
(c) Ratio: 8.0:1, PSNR: 45.4



(d) Ratio: 11.3:1, PSNR: 42.2

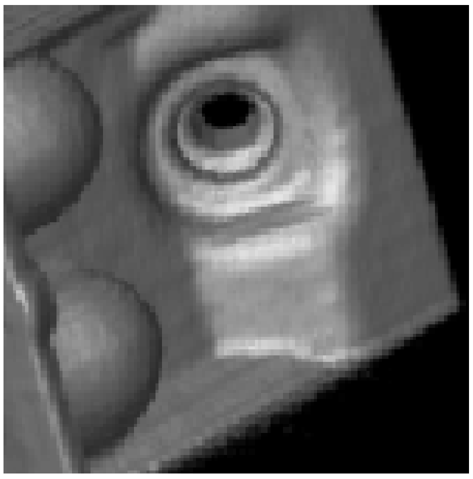


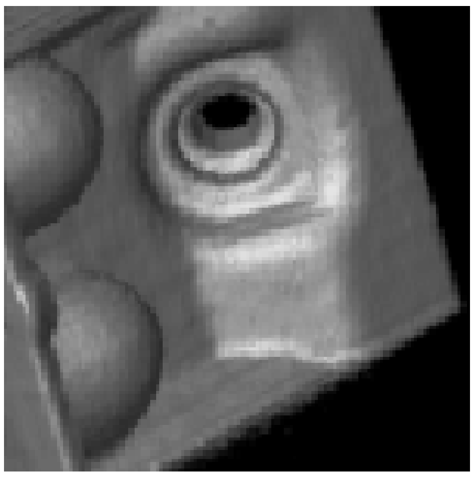
(e) Ratio: 14.1:1, PSNR: 41.8

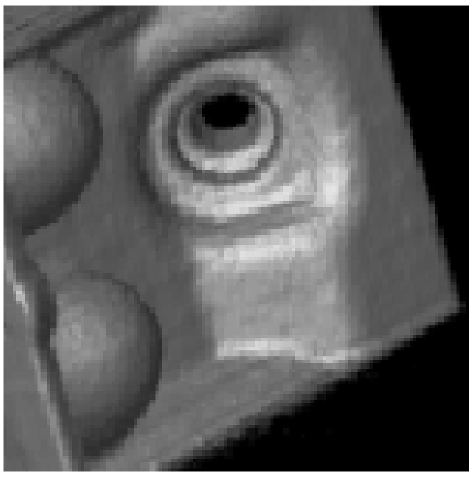


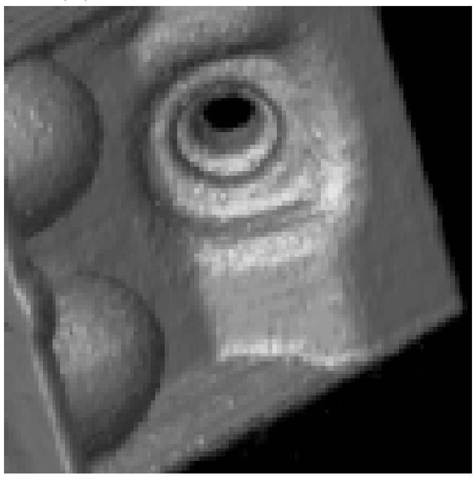
(f) Ratio: 20.6:1, PSNR: 37.9

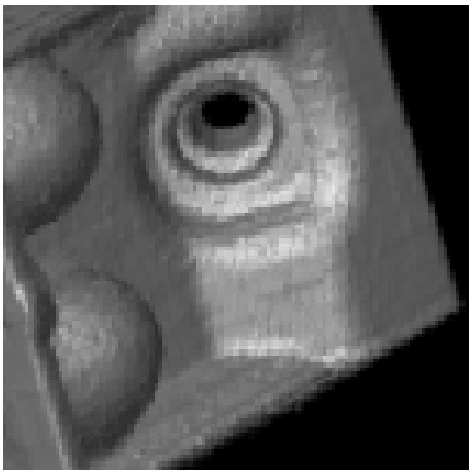
Figure 13: Rendered images of the Engine Block data set for various compression ratios.


(a) Original


(b) Ratio: 5.9:1, PSNR: 47.6


(c) Ratio: 8.0:1, PSNR: 45.4


(d) Ratio: 11.3:1, PSNR: 42.2


(e) Ratio: 14.1:1, PSNR: 41.8

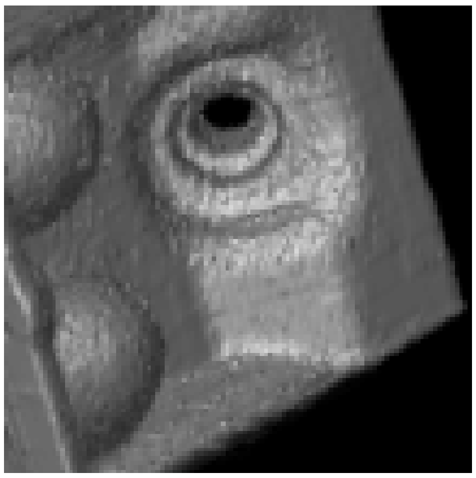

(f) Ratio: 20.6:1, PSNR: 37.9

Figure 14: Zoom of rendered images of the Engine Block data set for various compression ratios.

Recent BRICS Report Series Publications