Lossless Volume Data Compression Schemes

Philippe Komma*, Jan Fischer, Frank Duffner†
Visual Computing for Medicine Group
University of Tübingen, Germany

Dirk Bartz‡
ICCAS/Visual Computing
University of Leipzig, Germany

Abstract

Volumetric data is one of the most frequently generated type of data in modern medical imaging. Technical advances in the respective scanning technologies also increase the amount of data that is generated by a typical scanner. Since 1971 with the introduction of the first CT scanner, the space requirements for representing data have increased rapidly, in essence at an exponential rate.

In this paper, we examine various compression methods for their applicability for volume data, focusing on the reduced space requirements of the approaches. We apply these methods to a wide range of 8bit and 10-12bit volume datasets, thus exposing strengths and weaknesses of the compression methods. In summary, significant differences in compression performances clearly indicate which compression techniques should be used.

Keywords: Volume Data, Compression, Large Data

1 Introduction

Since the introduction of the first Computed Tomography (CT) in 1971, volumetric datasets have their common place in medical imaging and other applications fields (eg., non-destructive material testing). Ever since, many other volume scanners have been introduced, such as Magnetic Resonance Imaging (MRI), 3D X-Ray (ie., rotational angiography), 3D ultrasound, and many more.

While volume datasets in their early years were quite small by today's standards – first commercial CT scanners produced volumetric resolutions of 64^3 voxels –, they always posed a challenge at the time for processing, storage and data transfer. Datasets in current clinical practice can consist of $512 \times 512 \times 1200$ voxels[1] and are expected to increase more than sixteen-times to 2048^3 in the next few years. Overall, medical imaging (and similar

*philippe.komma@uni-tuebingen.de
†Frank Duffner is with the Department of Neurosurgery of the University of Tübingen.
‡bartz@iccas.de

[1]Right now, standard datasets in clinical practice are as large as 512^3 voxels. Some CT exams, in particular of peripheral blood vessels, can include a significant higher number of slices.

application fields) faces an exponential growth of dataset sizes, which generates increasing difficulties for the installed base of medical processing workstations.
An akin situation can be observed in scientific computing, where volumetric datasets from simulations grow at a similar rate.
In order to address the huge size requirements, several approaches have been examined. Body-centered grids provide an approximately 30% more compact voxel representation than cartesian grids [25]. Out-of-core methods provide a special main memory management, where only currently examined data partitions are held in main memory [4]. Compression methods reduce redundancy in volume data to provide a significantly more compact representation [11]. Last but not least, hierarchical methods, which can be combined with most other methods, provide processing means on different levels of detail, thus trading space requirements (and also computational costs) with data representation quality [10, 32].
In this paper, we focus on lossless data compression methods[2] that represent volume datasets at their full data fidelity, without reducing quality. While the presented compression methods are frequently combined, eg., hierarchical techniques and with out-of-core techniques, we concentrate on straight-forward compression methods of volume datasets with standard voxel addressing.
In the following, we present related work in the context of compression for volumetric datasets (Section 2) and provide a rather brief introduction in the examined compression methods in Section 3, where we also briefly discuss the usage of the presented approaches in our study. In Section 4, we introduce the examined datasets and present our compression results. Finally, we conclude our paper in Section 5.

2 Related Work

Several approaches in computer graphics and visualization employ compression techniques for a more efficient data representation. In most cases, it is used for geometry compression [23, 24] and the compression of volumetric datasets. In this paper, we focus on the latter application of compression.
Probably the first use of compression techniques for volumetric datasets employed the Run-Length-Encoding scheme (RLE), where sequences of voxels with identical intensity values were combined into one entry, which contained the intensity value and the length of the sequence. Although this scheme has been used before, first uses for volume datasets are referenced in Stony Brook's SLC file format for VolVis [2] and in Lacroute's volren package [15]. Recently, Schneider et al. have presented a technique for hierarchical quantization, where the levels of a multiresolution representation are represented through vector quantization, where the input vectors are encoded by indices from a codebook (dictionary) [21].
Another compression approach was presented by Fowler and Yagel, which is based on differential pulse-code modulation and Huffman encoding [8]. Yeo et al. presented a volume rendering approach that handles scalar volume datasets that are compressed by the discrete cosine transformation (DCT) [30]. The DCT scheme is also used in the JPEG image stan-

[2]Several of the discussed methods allow also for lossy compression. However, we used only a lossless parameter setting.

dard, which in turn is used as compression scheme in the DICOM standard for medical imaging [6]. Cochran et al. extended fractal image compression to provide a more compact representation of volumetric datasets [5, 20]. Note that both DCT and fractal-based approaches are not examined in this paper.
Many approaches use wavelets as a basis for a more compact representation for volumetric data, an idea which was introduced by Muraki [17]. A similar scheme for time sequences (animated) of volume datasets was presented by Westermann [28]. Ihm and Park presented another Wavelet-based compression scheme for volumetric data in 1999 [13]. In 2001, Guthe et al. used the Wavelet transformation combined with the Huffman encoding as access function for an encoding and decoding of animated volume datasets in order to provide high rendering performance [11]. Very recently, this approach has been extended into a hardware implementation for efficient memory management in a FPGA-based architecture [29] and on a GPU-based system [7]. However, we focus here only on lossless data compression for fully maintained data accuracy.
Other approaches for lossless image compression have been examined by Sahni et al. [19, 26].

3 Volumetric Compression Methods

In this section, we examine the compression methods that are discussed in this paper. Since we examine only compression methods for scalar volume datasets, we use the term voxel value interchangeably with data element of an alphabet (or dataset).

Huffman Coding and Arithmetic Encoding

The general idea of an entropy encoder is to represent frequent voxel values with a short code, and infrequent voxel values with a longer code. Huffman encoding [12] generates a binary tree, where one branch represents the current voxel value, and the other branch represents the remaining voxels values. Therefore, this binary tree actually re-samples a list of voxel values, sorted according to their appearance probability. For each branch a '0' or '1' code is assigned such that all voxel values can be represented by a prefix free binary number. Frequent voxel values are coded by a short code and infrequent voxel values are coded with longer codes. Some variations of Huffman encoding generate a balanced tree to optimize the average code length.
Arithmetic encoding functions [16] follow a similar principle, where the voxel values are encoded into a half-open interval $[0; 1)$ according to their appearance probability. Voxel values with a high probability occupy a large interval, and voxel values with a low probability occupy a short interval, thus requiring more bits for its representation. Usually, arithmetic encoding is considered to generate near optimal entropy encoding (in contrast to Huffman encoding), but usually at higher computational costs.
In the context of volumetric image data, both approaches performed somewhat better if they compressed slice-by-slice instead of the whole data volume. Furthermore, they usually provided a better compression if the image data was provided as sequence of voxel value differences to the previous voxels (difference buffer).

Run Length Encoding

One of the simplest and oldest compression schemes is Run-Length-Encoding (RLE). A data stream is compressed such that a run of equal voxel values is represented the voxel value and a number how often it appears in one continuous run. Since codes in the encoded data stream have a limited bit size, a sequence of identical voxel value longer than a maximum size is split into several RLE packages. Note that the original data from volumetric scanners does contain noise, therefore these original data does not contain long runs of identical voxel values.
RLE benefited most if the whole volume was compressed, instead of individual slices. This is in particular not surprising, since this enables RLE to also compress the right and left background voxels of neighboring slices with the same intensity. In contrast to most other cases, RLE did not benefit from a difference buffer.

Variable Bit Length Encoding

Not all voxel values require the same number of bits. Low intensity 8bit voxels (0..15) for example require a significantly smaller number of bits than high intensity voxel values (128-255) [19, 26]. Consequently, we can save bits per voxels for low intensity voxel values, but need to invest bits to represent the code size for the voxels. Instead of storing that information for every voxel, we generate a run-length-encoded representation of segments of the same bit length. This is a reasonable assumption for scanned volumetric data, since most of background voxels will have low intensity noise voxel values that require a smaller number of bits than the actual voxels of the scanned object.
In total, we need three data buffers that contain the segment lengths (segments of more than 2^8 elements are split in two segments), the number of bits required for the voxel values of the segments, and finally the actual voxel value stream, encoded in variable bit length. In order to further reduce the space requirements, each of the data buffers is encoded with an entropy encoder [19].
VBL allowed the best compression rates if the image data was provided through a difference buffer and in some cases if the whole data volume was compressed.

Dictionary-based Encoding

Dictionary-based encoding schemes are among the most popular schemes in data compression, where the encoded data is represented by codes that represent sequences that occur in the dataset. In our investigation, we employed the LZ77 compression approach [31], its variation, the LZW approach [27], and the commercially available ZIP package.
The older LZ77 scheme provides two different-sized buffers; the look-ahead-buffer (LA) and the much larger search buffer (SB). The LA represents the current window into the character stream that will be encoded, while the SB represents the stream of already processed characters. The LZ77 scheme traces appearances of character streams of the LA window in the SB, from the end to the start in the SB. This reference is then stored as a token instead of the traced voxel sequence.

While the alphabet of codes in the LZ77 scheme is generated in the process of reading the character stream, an initial alphabet is already created with the LZW approach [27]. If for example an extended alphabet of 12 bits is used, the first 256 positions are filled with the ASCII code, and the remaining 3840 positions are available for longer codes. A character sequence (initially of one character) is searched in the already filled code book. If it is found, the next character is appended, and the search is continued. If an appropriate code is not found, the code of the previous found code (with one character less) is written to the output stream, and the current sequence is stored as code into the next available dictionary position. While the standard implementation of the LZW algorithms uses a hash table (4096 entries), other variations use different data structures, like a ring buffer.
The advantage of the LZ77 and LZW approaches lies in the implicitly stored dictionaries, that do not require space for an explicitly stored one. Typical examples for LZ77 compression schemes are gzip (combined with Huffman coding) [9] and ZIP. Unix's compress uses LZW.
All dictionary-based compression algorithms clearly benefited, if the dataset was compressed as a whole volume and if a difference buffer was used for compression.

Burrows-Wheeler Transformation-based Algorithms

The Burrows-Wheeler transformation (BWT) represents the basis for a whole family of compression algorithms. The BWT itself is a (reversible) permutation scheme that (re-) sorts a block of data according to the context [3, 18, 1], eg., they are sorting after the occurrence of characters. After this re-sorting, the data blocks are organized in a more compression-friendly type. For the popular BZIP2 compression algorithm, the BWT is combined with a Huffman coding, which encodes the permuted input block [22].
Similar to the previous approaches, BZIP2's compression rate increased if the whole volume was compressed instead of single image slices. In this case, it also benefited from a difference buffer.

Wavelet-Transformation

Wavelet transformations are the basis for many the state-of-the-art compression algorithms, including the JPEG-2000 standard [14]. Essentially, a coarse representation of a data block (scaling functions) is successively refined by various scaled and translated detail functions, the wavelets. In essence, this scheme splits the data in a low-pass part (scaling function) and high-pass details (wavelets). For volume datasets, this low-pass part represents a lower resolution dataset.
A major criterion for the compression quality is the type of wavelet base-functions that are used. Typical examples are the Haar-wavelets (usually only used for description of the principle) or B-Spline functions. In our investigation, we use Haar-wavelets, C6/2 wavelets, and quadratic and cubic B-splines as base-functions for the different datasets (always the one with the best compression performance), while the JPEG-2000 standard uses D5/3 base-functions.

Typically, wavelet-based compression schemes can exploit three-dimensional structures and thus provide a better compression rate if the whole volume dataset is compressed. While this assumption was correct for most datasets, it turned out to be not true for the sparse vessel dataset, where a slice-by-slice compression allowed better compression rates. In case of the used JPEG-2000 library [14], only an image-based implementation was provided, hence only this mode is used.

Dataset/ Bits/voxel	Resolution	Modality
Skull / 8	256x256x256	3D Xray
Engine / 8	256x256x128	CT
Teapot / 8	256x256x178	CT
Vessel / 8	256x256x256	3D Xray
Head / 10	512x512x55	MRI
Thorax / 12	512x512x169	CT
Colon/prone / 12	512x512x463	CT
Head Aneuryrism / 12	512x512x512	3D Xray

Table 1: Examined datasets - four 8bits-per-voxel datasets, one 10bits-per-voxel, and three 12bits-per-voxel dataset.

4 Results

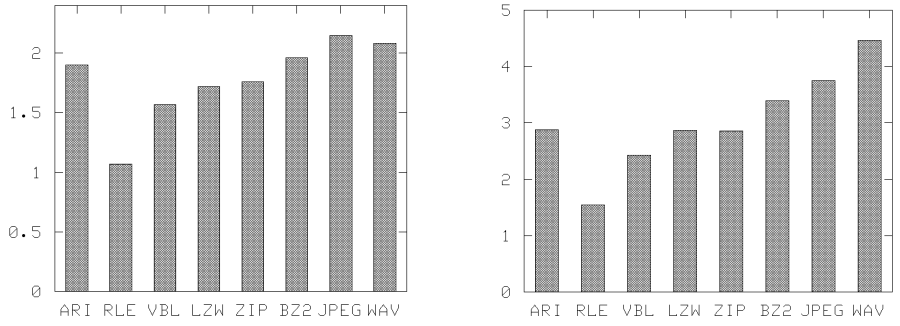
In our examination, we employ the compression approaches described in the previous section (Section 3) to encode a variety of volume datasets, which are described in Table 1. These datasets represent a wide diversity of different types of dataset, representing different scanning modalities (3D Xray, CT, MRI), different bit depths (8, 10, 12 Bits/voxel), different sizes (ranging from the rather small Engine dataset to rather large Head Aneuyrism dataset). While most of the datasets are of medical origin, two (Engine, Teapot) and from industrial applications or were modeled. Note that the Vessel dataset is already preclassifed to remove background noise. This results in an extra sparse dataset, while the Head Aneuyrism still provides that background noise which must be represented by each lossless compression approach.
We examined various compression schemes, each with several parameters to optimize. In fact, the number of individual experiments for each dataset could be as large 560 depending on data size and bit depth. Since we don't have the space to elaborate on each in detail, we only present results with parameter settings (see Section 3) that generated the highest compression rates (Eq. 1) shown in Table 2, where the best method is highlighted in bold faces. The results of a subset of tested methods are presented in Figures 1- 3 and discussed in the following paragraphs.

$$compressionRate = \frac{OriginalDataSize}{CompressedDataSize} \tag{1}$$

Dataset	Entropy Enc. Arith/Huff	RLE	VBL	Dictionary-based LZ77/LZW/ZIP	BWT BZ2	JPEG (C6/2)	WAV
Skull	1.90/1.88	1.07	1.57	1.66/1.72/1.76	1.96	**2.15**	2.08
Engine	2.88/2.70	1.54	2.43	2.67/2.87/2.86	3.39	3.75	**4.46**
Teapot	6.68/4.85	5.15	6.45	8.48/8.35/8.75	14.58	9.96	**15.94**
Vessel	40.61/7.14	30.95	28.19	51.24/56.04/52.27	**74.77**	19.64	19.74
Head	2.70/n.a.	1.96	2.10	n.a./2.40/2.65	2.83	**3.60**	2.68
Thorax	2.44/n.a.	1.90	2.20	n.a./1.90/2.53	2.94	**3.77**	3.53
Colon /prone	2.32/n.a.	2.10	2.23	n.a./2.26/2.35	2.56	**3.24**	2.50
Head Aneur'	2.68/n.a.	2.35	2.84	n.a./2.85/3.05	4.49	**10.53**	5.53

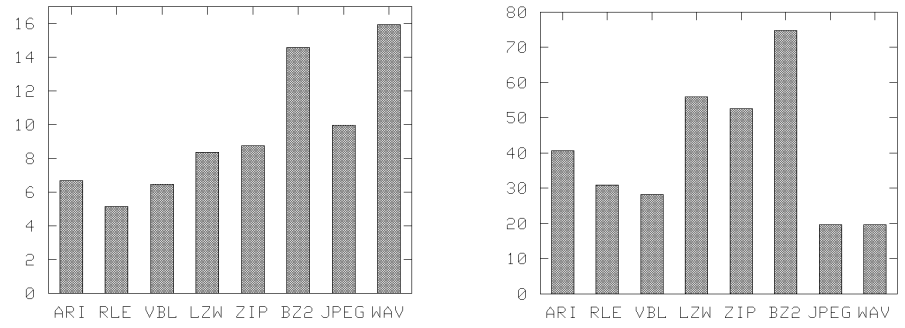
Table 2: Best compression rates for the various compression schemes and datasets. The best performing algorithm is marked in bold faces. BWT - Burrows-Wheeler transformation, JPEG - JPEG-2000, Arith - Arithmetic Encoding, Huff - Huffman Encoding, RLE - Run-Length-Encoding, VBL - Variable Bit Length, LZ77 - Lempel-Ziv 1977, LZW - Lempel-Ziv-Welch, BZ2 - BZIP2, WAV - Wavelets.

Specifically, we looked into arithmetic and Huffman-coding, run-length-encoding (RLE), variable bit length encoding (VBL), LZ77, LZW, ZIP, BZIP2, (lossless) JPEG-2000, and the wavelet transformation. For practical reasons and since Huffman-encoding was outperformed by arithmetic encoding of 8bits dataset, the 10 and 12 bits datasets were not compressed with Huffman-encoding. Furthermore, the compression performance of LZ77 is already reflected by ZIP, which uses LZ77. Therefore, we omitted the results for Huffman



(a) (b)

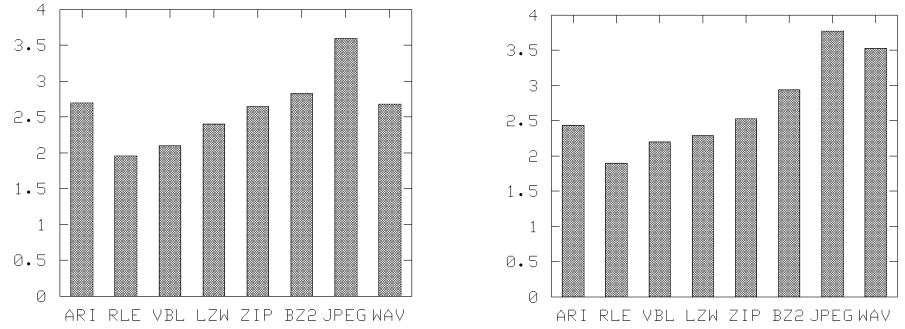
Figure 1: Graph of compression rates for the skull (a) and engine (b) datasets. Huffman and LZ77 results are omitted, due to page limitations.



(a) (b)

Figure 2: Graph of compression rates for the teapot (a) and sparse vessel (b) dataset. Huffman and LZ77 results are omitted, due to page limitations.

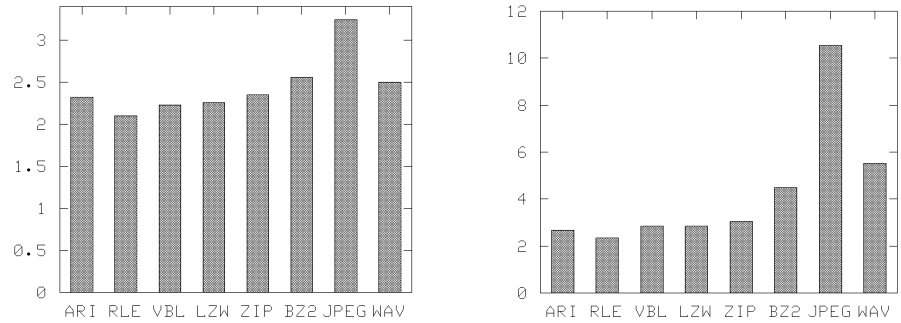
and LZ77 in the graphs (Fig. 1- 3) for similar reasons (and due to page size limitations).
Overall, we found very little influence of the data origin (modality) on the compression method. More important is the characteristic of the dataset that is not necessarily determined by its origin. A typical example are 3D Xray datasets, which can be very sparse if pre-classified (Vessel), or rather dense if it includes the background noise (Skull and Head Aneuyrism).
We observed that for the whole dataset variety, BZIP2 performe well and it always achieved one of the three highest compression rates. Only for dense datasets (Figs. 1, 2a, 3) it was exceeded by JPEG-2000 and the wavelet transformation. However, these two approaches



(a) (b)

Figure 3: Graph of compression rates for the head (a) and thorax (b) dataset. Huffman and LZ77 results are omitted, due to page limitations.

provided a very poor performance for sparse datasets, like the 3D Xray angiography dataset "vessel" (Fig. 2b).



(a) (b)

Figure 4: Graph of compression rates for the colon/prone (a) and head aneuyrism (b) dataset. Huffman and LZ77 results are omitted, due to page limitations.

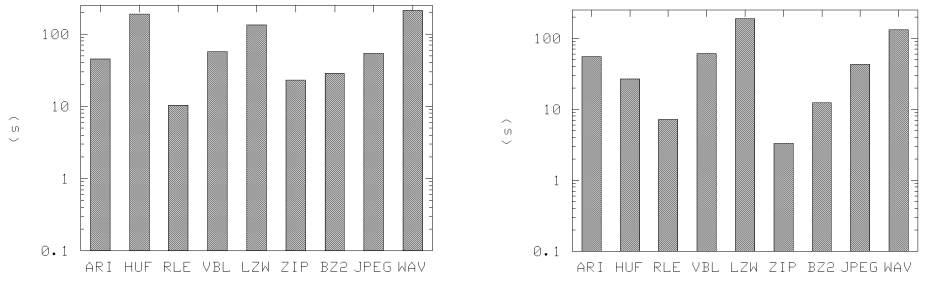
In contrast, JPEG-2000 provided the best results for 10 and 12bits datasets (Fig. 3). Please note that all wavelet schemes (including lossless JPEG-2000) achieved the best performance with different base-functions. On the skull, engine, and teapot datasets, quadratic B-spline wavelets achieved the best results, while the thorax dataset has the best result with cubic B-spline wavelets. The head dataset was compressed best with C6/2 wavelets. The above mentioned poor performance for the vessel dataset was achieved with Haar-wavelets, while all other used wavelet functions performed even worse.

Entropy encoding schemes like arithmetic and Huffman-coding also performed poorly on rather sparse datasets (teapot and vessel, Fig. 2), and only reached average compression rates for the dense datasets. However, for 10 and 12bits datasets, it reached a compression performance close to BZIP2 (Fig. 3).

RLE and VBL always provided only a poor performance for all datasets. In particular RLE, however, is a simple and fast algorithm, which is among the oldest compression schemes.

Dictionary-based approaches like LZ77, LZW, and ZIP provided only an average compression rate for all datasets. In terms of encoding and decoding speed, dictionary-based approaches have a clear advantage to most other compression schemes (ZIP in Fig. 5). Only RLE and in many cases BZIP2 could reach a similar decoding speed. In particular ZIP provided a very high speed, while the significantly lower speed of LZ77 (not documented in Fig. 5) and LZW is due to the non-optimized implementation. Wavelet-based coding approaches (Wavelets and JPEG-2000) are together with the Huffman implementation (encoding only) among the slowest encoding and decoding schemes. Compared to BZIP2 – which achieves a comparable compression rate in most cases – they are about an order of magnitude slower.

In summary, the important criterion to select a compression approach is the characteristic of the dataset. A sparse dataset may be compressed more efficiently with BZIP2 than with



(a) (b)

Figure 5: Log-scale graph of average encoding (a) and decoding (b) time. LZ77 results are omitted.

wavelet-based approaches, as we could see for the Vessel dataset. However, the somewhat less sparse Teapot dataset showed reasonable results for BZIP2 and Wavelets (but not for JPEG-2000). For dense datasets, wavelet-based approaches, in particular JPEG-2000, performed very well.

In Figures 6 and 7, we examine the local compression rate at voxel level to visualize areas of high and low compression rate for the vessel and the thorax datasets. We mapped the compression rate to a clamped hue range (see Fig. 6d for legends of the colormap) to visualize areas of high and low compression rate for the vessel and the thorax datasets. To enable a better discrimination, the hue range did not include magenta values. In order to allow the inspection of the compression rate of interior voxels in the volume rendered representation of the vessel dataset (upper two rows), we assigned high transparency values to high compression rates, and high opacities to low compression rates. This results in a highly transparent representation of highly compressible boundary areas. Note that a voxel-oriented representation of the compression rate is not possible for all compression schemes. Hence. and due to page size restrictions, we show only the results for arithmetic encoding (ARITH), run length encoding (RLE), variable bit length encoding (VBL), and Lempel-Ziv-Welch (LZW).

5 Conclusions

In this paper, we presented results on the suitability of various compression algorithms for volumetric datasets. As we showed in Section 4, the choice of the optimal compression algorithms depends on the type of datasets. For dense datasets, wavelet-based schemes provide an excellent compression rate, while JPEG-2000 is in particular suited for datasets with a higher voxel accuracy (10bits and more). In contrast, the compression performance for sparse datasets exposed a clear weakness of wavelet-based compression, where its performance was more than three times worse than for the BZIP2 performance. Furthermore, wavelet-based methods have significant computational costs for the encoding and decoding.

If nothing is known on the nature and structure of the datasets, BZIP2 is the clear winner of all examined compression schemes. Its compression performance was always among the top three for all datasets (surpassed only by wavelet-transformation and JPEG-2000 in some cases). Furthermore, BZIP2 is also a fast compression method, in particular for the decoding of the data stream. It was only slower than ZIP and RLE decoding.

Acknowledgments

This work has been supported by the Department for Neurosurgery of the University Hospital Tübingen and by the project VIRTUE of the focus program 1124 on "Medical Navigation and Robotics" of the German Research Council (DFG) and the Center of Competence for Minimally Invasive Medicine and Technology Tübingen-Tuttlingen (MITT). Most datasets are available through http://www.volvis.org. We thank Urs Kanus and Gregor Wetekam for proof-reading.

References

[1] J. Abel. Grundlagen des Burrows-Wheeler Kompressionsalgorithmus. *Informatik - Forschung und Entwicklung*, 18(2):80–87, 2003.

[2] R. Avila, L. Sobierajski, and A. Kaufman. Towards a Comprehensive Volume Visualization System. In *Proc. of IEEE Visualization*, pages 13–20, 1992.

[3] M. Burrows. A Block-Sorting Lossless Data Compression Algorithm. Technical Report 124, Digital Equipment Corporation, SRC, Palo Alto, CA, 1994.

[4] Y. Chiang, J. El-Sana, P. Lindstrom, and C. Silva. Out-Of-Core Algorithms for Scientific Visualization and Computer Graphics. In *IEEE Visualization Tutorial T4*, 2002.

[5] W. Cochran, J. Hart, and P. Flynn. Fractal Volume Compression. *IEEE Transactions on Visualization and Computer Graphics*, 2(4):313–322, 1996.

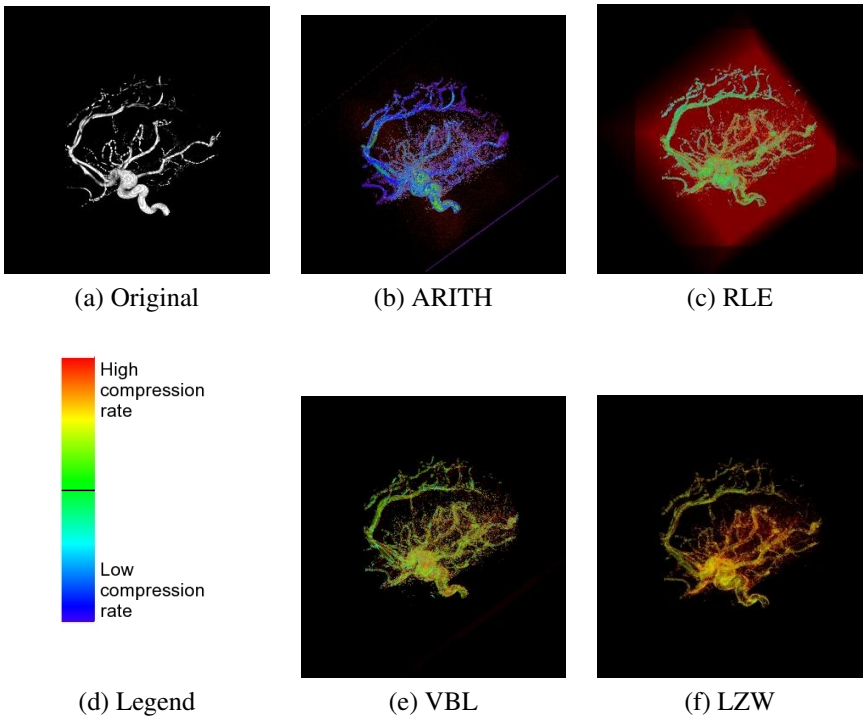
[6] DICOM Standards Committee. Dicom - homepage. http://medical.nema.org, accessed 2005.

[7] N. Fout, H. Akiba, K. Ma, A. Lefohn, and J. Kniss. High Quality Rendering of Compressed Volume Data Formats. In *Proc. of Eurographics/VGTC Symposium on Visualization*, 2005.

[8] J. Fowler and R. Yagel. Lossless Compression of Volume Data. In *Proc. of Symposium on Volume Visualization*, pages 43–50, 1994.

[9] J. Gailly and M. Adler. The gzip home page. http://www.gzip.org, accessed 2005.

[10] M. Garland. Multiresolution Modeling: Survey and Future Opportunities. In *Eurographics STAR report 2*, 1999.



(a) Original (b) ARITH (c) RLE

(d) Legend (e) VBL (f) LZW

Figure 6: Visualization of compression rates according to color map in (d). Note that only interior voxels (with a lower compression rate are shown. A volume rendering of the vessel (a) datasets is compressed by arithmetic encoding (b), RLE (c), VBL (e), and LZW (f). High and low compression rates are relative to the achieved min/max compression rates for this dataset (d); see also Table 2.

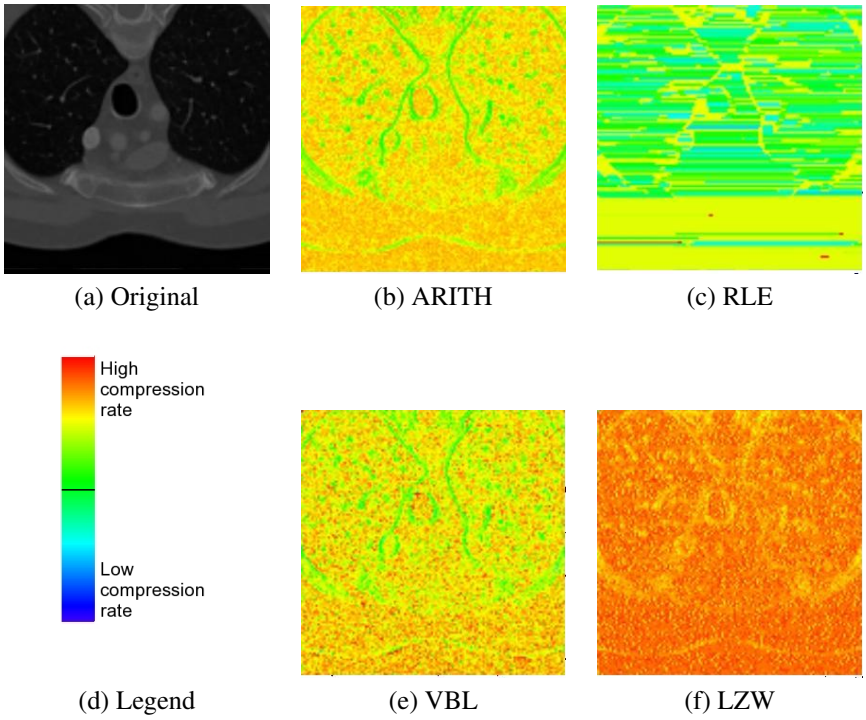
[11] S. Guthe and W. Straßer. Real-Time Decompression and Visualization of Animated Volume Data. In *Proc. of IEEE Visualization*, 2001.

[12] D. Huffman. A Method for the Construction of Minimum Redundancy Codes. *Proc. of IRE*, 40(9):1098–1101, 1952.

[13] I. Ihm and S. Park. Wavelet-Based 3D Compression Scheme for Interactive Visualization of Very Large Volume Data. *Computer Graphics Forum*, 18(1):3–15, 1999.

[14] D. Janssens. J2000 - A JPEG-2000 Codec. http://www.j2000.org, accessed 2005, now no longer available.

[15] P. Lacroute and M. Levoy. Fast Volume Rendering Using a Shear-Warp Factorization of the Viewing Transformation. In *Proc. of ACM SIGGRAPH*, pages 451–458, 1994.



(a) Original (b) ARITH (c) RLE

(d) Legend (e) VBL (f) LZW

Figure 7: Visualization of compression rates according to color map in (d). A slice from the thorax datasets (a) is compressed by arithmetic encoding (b), RLE (c), VBL (e), and LZW (f). High and low compression rates are relative to the achieved min/max compression rates for this dataset (d); see also Table 2.

[16] A. Moffat, R. Neal, and I. Witten. Arithmetic Coding Revisited. In *Proc. of Data Compression Conference*, pages 202–211, 1995.

[17] S. Muraki. Volume Data and Wavelet Transforms. *IEEE Computer Graphics and Applications*, 13(4):50–56, 1993.

[18] M. Nelson. Data Compression with the Burrows-Wheeler Transformation. *Dr. Dobbs Journal*, 21(9):46–50, 1996.

[19] S. Sahni, B. Vemuri, F. Chen, C. Kapoor, C. Leonard, and J. Fitzsimmons. State-of-the-Art Lossless Image Compression Algorithms. Technical report, University of Florida, 1997.

[20] D. Saupe. *Fractal Image Encoding and Analysis (Y. Fischer (Ed))*, chapter Fractal Image Compression via Nearest Neighbor Search. Springer Verlag, 1996.

[21] J. Schneider and R. Westermann. Compression Domain Volume Rendering. In *Proc. of IEEE Visualization*, pages 293–300, 2003.

[22] J. Seward. bzip2 - Homepage. http://www.bzip.org, accessed 2005.

[23] G. Taubin. 3D Geometry Compression and Progressive Transmission. In *Eurographics STAR report 3*, 1999.

[24] G. Taubin, M. Deering, C. Gotsman, S. Gumhold, and J. Rossignac. 3D Geometry Compression. In *ACM SIGGRAPH Course 38*, 2000.

[25] T. Theußl, T. Möller, and E. Gröller. Optimal Regular Volume Resampling. In *Proc. of IEEE Visualization*, pages 91–98, 2001.

[26] B. Vemuri, S. Sahni, F. Chen, C. Kapoor, C. Leonard, and J. Fitzsimmons. *Encyclopedia of Optical Engineering (R. Driggers and E. Lichtenstein (Eds.))*, chapter Lossless Image Compression. Marcel Drekker Inc., 2002.

[27] T. Welch. A Technique for High-Performance Data Compression. *IEEE Computer*, 17(6):8–19, 1984.

[28] R. Westermann. Compression Domain Rendering of Time-Resolved Volume Data. In *Proc. of IEEE Visualization*, pages 168–175, 1995.

[29] G. Wetekam, D. Staneker, U. Kanus, and M. Wand. A Hardware Architecture for Multi-Resolution Volume Rendering. In *Eurographics/SIGGRAPH Symposium on Graphics Hardware*, 2005.

[30] B. Yeo and B. Liu. Volume Rendering of DCT-based Compressed 3D Scalar Data. *IEEE Transactions on Visualization and Computer Graphics*, 1(1):29–43, 1995.

[31] J. Ziv and A. Lempel. A Universal Algorithm for Sequential Data Compression. *IEEE Transactions on Information Theory*, 23(May):337–343, 1977.

[32] D. Zorin, P. Schröder, T. DeRose, L. Kobbelt, A. Levin, and W. Sweldens. Subdivision for Modeling and Animation. In *ACM SIGGRAPH Course 23*, 2000.