A Survey of Compressed GPU-Based Direct Volume Rendering

M. Balsa Rodríguez[1], E. Gobbetti[1], J.A. Iglesias Guitián[1,3], M. Makhinya[2], F. Marton[1], R. Pajarola[2], S. K. Suter[2]

[1] Visual Computing Group, CRS4, Pula, Italy – http://www.crs4.it/vic/
[2] Visualization and MultiMedia Lab, University of Zürich, Switzerland – http://vmml.ifi.uzh.ch/
[3] Universidad de Zaragoza, Spain

Abstract

Great advancements in commodity graphics hardware have favored GPU-based volume rendering as the main adopted solution for interactive exploration of rectilinear scalar volumes on commodity platforms. Nevertheless, long data transfer times and GPU memory size limitations are often the main limiting factors, especially for massive, time-varying or multi-volume visualization, or for networked visualization on the emerging mobile devices. To address this issue, a variety of level-of-detail data representations and compression techniques have been introduced. In order to improve capabilities and performance over the entire storage, distribution and rendering pipeline, the encoding/decoding process is typically highly asymmetric, and systems should ideally compress at data production time and decompress on demand at rendering time. Compression and level-of-detail pre-computation does not have to adhere to real-time constraints and can be performed off-line for high quality results. In contrast, adaptive real-time rendering from compressed representations requires fast, transient, and spatially independent decompression. In this report, we review the existing compressed GPU volume rendering approaches, covering compact representation models, compression techniques, GPU rendering architectures and fast decoding techniques.

Categories and Subject Descriptors (according to ACM CCS): Computer Graphics [I.3.3]: Picture/Image Generation—Computer Graphics [I.3.7]: Three-dimensional graphics and realism—Coding and Information Theory [E.4]: Data compaction and compression—Compression (Coding) [I.4.2]: Approximate methods—

1. Introduction

GPU accelerated direct volume rendering (DVR) on consumer platforms is nowadays the standard approach for interactively exploring rectilinear scalar volumes. Even though the past several years witnessed great advancements in commodity graphics hardware, long data transfer times and GPU memory size limitations are often the main limiting factors, especially for massive, time-varying, or multi-volume visualization in both local and networked settings. To address this issue, a variety of level-of-detail (LOD) data representations and compression techniques have been introduced.

In this context, data compression associated to GPU decompression is of great importance to save storage space and bandwidth at all stages of the processing and rendering pipelines. Few methods, however, support on-demand, fast and spatially independent decompression on the GPU, which is required for maximum benefits [FM07]. While the domain of real-time GPU volume rendering has been exceptionally well covered by well established surveys [EHK*06], surveys on rendering from compressed representations have mostly focused on CPU decoding techniques [Yan00] and offline compression performance [KFDB07]. In recent years, a number of new techniques have appeared in the GPU volume graphics literature (e.g., tensor representations [SIM*11] or sparse coding methods [GIM12]), hardware supported methods are evolving [Ell12], and established techniques are gaining an increased interest in the context of emerging networked applications.

Our presentation of the state-of-the-art in GPU-based direct volume rendering from compressed data starts with a characterization of the basic concepts common to all current architectures and of the requirements of each component (see Sec. 2). After reviewing the most common compact data models used as underlying data representations (Sec. 3), we dis-

DOI: 10.2312/conf/EG2013/stars/117-136

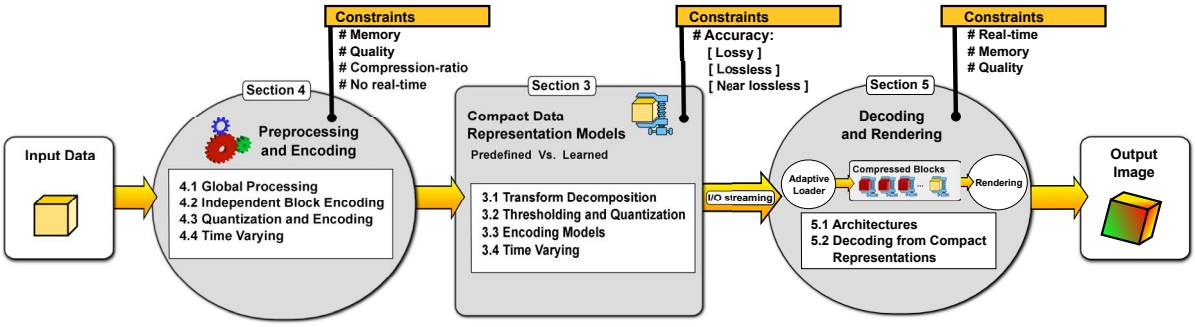

Figure 1: *Generic GPU-based compressed direct volume rendering architecture. This figure serves as a visual table of contents for the STAR. Basic building blocks are listed together with references to the sections that discussed them.*

cuss the issues related to compression and encoding (Sec. 4), as well as decoding and rendering (Sec. 5). The survey concludes with a short summary of the current achievements in the domain and an indication of open issues.

2. Compressed GPU Volume Rendering

Volume visualization and massive models techniques are well researched subjects. In this state-of-the-art report, we limit our discussion to the issues related to the effective incorporation of compression methods into real-time volume rendering pipe-lines. We refer the reader to established surveys on massive model visualization [DGY07, GKY08, YGKM08] and GPU-accelerated volume rendering [EHK*06] for a broader coverage of other relevant techniques for attacking massive data.

Compression and level-of-detail (LOD) precomputation are effective approaches for facilitating volume processing due to the large data sizes, limited resources, and highly redundant nature of volume data. Compression/decompression can appear at many stages of the volume rendering pipeline, whose generic structure is depicted in Fig 1. In the context of volume rendering, the optimal configuration with respect to data transport is to compress the stored model (on disk) and decompress the data at the latest possible stage of the rendering pipeline. This approach will reduce transport bandwidth usage, and will better exploit of the memory hierarchy, with larger capacity and slower access times on storage components, and smaller capacity, but faster access times near the GPU. However, note that the overall throughput and frame rate may in the end depend on a balanced use of all available resources such as the CPU, GPU, main and graphics memory.

Compression and LOD precomputation do not have to adhere to real-time constraints and can best be performed off-line for high quality results. In contrast, adaptive real-time rendering from compressed representations requires the incorporation of low-delay and spatially selective decompression into a multiresolution out-of-core renderer. The characteristics of the major pipeline stages are elaborated next.

Preprocessing The input volume is generally processed off-line to get a compressed representation. When dealing with large volume datasets exceeding the available main memory resources, the compression preprocess should ideally use an out-of-core data partitioning and processing strategy to handle the full dataset most efficiently. Furthermore, to reduce the preprocessing time it is desirable to exploit parallelization over multiple machines, multi-core processors or additionally involve the GPUs as well. At this stage the compression algorithms are typically parametrized to achieve the desired compression ratio or ensure a maximum tolerated error. Data compression approaches suitable for DVR are reviewed and presented in Secs. 3 and Sec. 4.

Adaptive loading and streaming of data At run-time, the viewer application should adaptively select an appropriate working set of nodes from the LOD data structure that constitutes the desired best possible representation for the current frame(s). These LOD nodes are eventually processed by the renderer itself to generate the image. Random access, in particular to large out-of-core data, is considered a critical issue. Different criteria drive the loading of data, potentially involving viewing and time-dependent parameters, available hardware resources or the required image accuracy. We can, moreover, consider different types of streaming channels that our system will need to use: sequentially accessing data from the network or files, coherently processing that data on the CPU or GPU, as well as sending the data eventually through the GPU renderer. The compression scheme should be configured to exploit the bandwidth along the streaming hierarchy (e.g. network, disk bus, PCI bus, up to GPU memory). Volume decomposition approaches for preprocessing are presented in Sec. 4, while adaptive rendering architectures are discussed in Sec. 5.

Decompression and rendering As previously mentioned, asymmetric compression schemes are desired, as they are designed to provide fast decoding at runtime at the expense of increased (but high quality) encoding time. Data decompression could be placed at different stages of the DVR pipeline, and it can further be subdivided to exploit

both the CPU as well as the GPU. In general, the data should travel through the pipeline in compressed form as far as possible to avoid any bandwidth bottleneck. However, any decoding overload on the CPU or GPU must be avoided as well. Moreover, in a compressed volume rendering architecture, decompression should ideally be *transient* and *local*, that is, a fully reconstructed volume is never produced. The advantage of this approach is in efficient use of limited memory resources, such that larger and/or more volume data is available for visualization at any given time. This requires, however, even full random access at the voxel level, or an appropriate reordering of rendering operations to schedule and process sub-portion of the dataset. This is further discussed in Sec. 5.

3. Compact Data Representation Models

The main idea behind volume data compression is to find a more compact representation of the data, which requires fewer bits for encoding than the original volume. A typical approach is to decompose or transform the input dataset $\mathcal{A}$ by means of a mathematical framework into a *compact data representation*, which can be sent over the network and up to the GPU in a light-weight form. When the data needs to be visualized, the inverse process is applied to obtain the approximation $\widetilde{\mathcal{A}}$ of the original volume. As previously outlined, this decomposition-reconstruction process is nowadays usually highly asymmetric [FM07] (see Fig. 2). The data decomposition step is usually an offline process that is not time critical, while the reconstruction process needs to be performed in real-time while satisfying a certain application-defined image quality error tolerance.

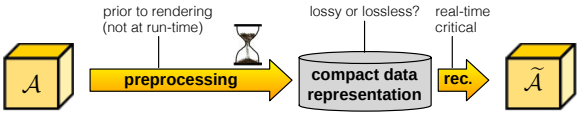

Figure 2: *Compression-domain DVR from compact data representations – an asymmetric process: First, a dataset $\mathcal{A}$ is decomposed into a compact data representation during a preprocess prior to rendering. Second, the approximation $\widetilde{\mathcal{A}}$ from the original is reconstructed at run-time.*

Volumetric datasets are most commonly represented as a (possibly time-varying) rectilinear grid of samples, i.e., with a digitally sampled representation of a signal using a sum of delta functions in space (and eventually time). While convenient for many applications, this data model is mostly inefficient for compression, which would benefit from more meaningful representations capturing the significant part of the signal with only a few coefficients. The choice of the model that sparsifies the input signal, and its combination with efficient encoding methods, are crucial steps for the success of the compression approach.

In recent years, much effort has been put into the development of mathematical representations that reconstruct/approximate volumetric datasets. A popular approach is to find a limited set of elementary signals (most of the time forming one or more bases), together with a corresponding set of coefficients, whose weighted linear combination is a close approximation of the original data.

Generally, there are two different types of models: *pre-defined models* and *learned models* (see Fig. 3). Pre-defined models apply a given general analytical model to the dataset and output its coefficients, while learned models are generated individually for every dataset before the coefficients corresponding to the data and the learned model are produced. Pre-defined and learned bases should be seen as two alternatives with both assets and drawbacks. Approaches using pre-defined bases are often computationally cheaper, while approaches using learned models require more precomputation time. However, learned bases are potentially able to remove more redundancy from a dataset.

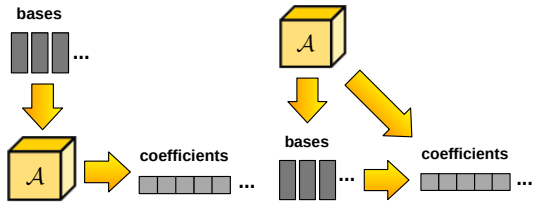

Figure 3: *Pre-defined vs. learned bases for compact data representations. The coefficients show the relationship between the bases and the original $\mathcal{A}$.*

By definition, a compact data representation can be *lossless*, *near lossless*, or *lossy*. Lossless algorithms ensure a binary value recovery in the reconstruction and guarantee zero error. Near lossless algorithms are theoretically lossless, however, may suffer from numerical floating point accuracy reconstruction issues. Lossy algorithms incorporate a controlled reduction of approximation quality, e.g., through some sort of reduced bit quantization levels or removing coefficients, typically driven by a certain tolerated reconstruction error. Near lossless schemes occur usually when coefficients of a decomposition or a transform need to be represented as floating point values. However, in theory such models are lossless. For many medical imaging applications a lossy compression is viewed as unacceptable [FY94], unless the compression error is within the limit of acquisition errors, or a progressive improvement of the representation can be guaranteed. In other applications, where the data only has to look the same as the original, lossy compression schemes provide a much greater compression ratio. Lossy models are mainly used in order to increase the compression ratio or the data reduction level. Accordingly, the great advantage of many lossy approaches is the possibility to have a parameter tuning according to the desired reconstruction quality (accuracy).

Note that even though some of the models are by definition lossless, they are in practice often utilized in their lossy form, either because their implementation is not fully lossless due to machine precision, or, more commonly, because representations are truncated in order to obtain a larger compression ratio. Large data visualization applications nowadays mostly work on lossy data representations.

A lossy representation from a lossless compact model can often be introduced in all stages of the preprocessing pipeline, the model can be compressed in a lossy way, insignificant coefficients can be truncated, a lossy quantization scheme and/or a lossy encoding scheme can be applied. Actually, the thresholding of certain coefficients can even be applied in the rendering step of the visualization pipeline. The fact that data loss can be introduced at any stage of the visualization pipeline shows that the error tolerance is highly application-driven and/or user-defined.

In general, once the data is represented in a compact form, further data reduction and compression steps are applied. For instance, insignificant coefficients in the transform domain or the compact representation can be neglected. This *coefficient truncation* is typically done by a thresholding or a ranking of coefficients. In a way, this corresponds to a quantization of possible reconstructed value spaces. Vector quantization, e.g., explicitly includes a quantization in the decomposed compact data model. However, in most pipelines, there is another *quantization step* that, e.g., converts floating point coefficients to integer values. Eventually, the quantized values can be further encoded with entropy coding. Depending on the volume compression algorithm all three steps are implemented or only a subset thereof.

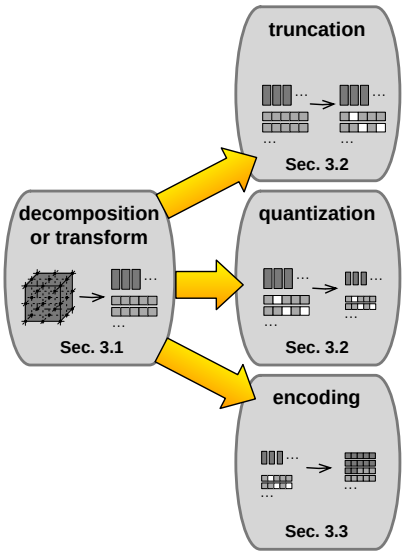

Figure 4: *Stages that can be applied to produce a compact data representation model. Truncation, quantization, and encoding are applied in various combinations.*

Next, we introduce the main compact data representation models and the entropy encoders that have been proposed for volume compression of rectilinear volume datasets in direct volume rendering frameworks. The goal of this compact models compendium is to give a systematic overview of the available compressed volume rendering models and encoders. While many possible organizations exists, for ease of discussion within the context of volume rendering architectures, we have chosen to classify methods into (1) transform/decomposition models (Sec. 3.1), (2) thresholding and quantization approaches (Sec. 3.2) and (3) encoding models (Sec. 3.3). In fact, one compressed volume rendering approach may include several of the compact model concepts combined: e.g., a dataset can be first transformed into another domain, is then represented by a codebook, quantized and is eventually entropy encoded. Therefore, the summarized methods do not exclude each other, on the contrary, some of them match well together, as we will discuss in the following sections.

It should also be noted that something particular to large volume visualization is that typically the input datasets are divided into subblocks and into multiresolution data structures, see also [EHK*06]. Accordingly, the compact models should provide hierarchical (LOD) data structures as well as independent bases or model generation from the individual subblocks. These aspects will be further analyzed in Sec. 4 and Sec. 5.

3.1. Transform/Decomposition Models

The idea of transform domain approaches is to get a more biased data distribution in the transform domain and eventually accelerate rendering. That way, artifacts from thresholding or quantization can be minimized such as the thresholding of wavelet coefficients typically performed in the frequency domain. While many of the transform domain approaches convert the data into frequency domain, transform coding is not exclusively limited to only do that. For example, wavelets additionally preserve spatial locality; PCA-like models transform the data into a different coordinate system that better fits the given variability of the statistical data distribution. In the following, we describe the major domain transform/decomposition models as well as their applications in compressed volume rendering.

Discrete Fourier Transform (DFT) Volume data compression in the frequency domain was first applied by means of the Fourier transform – initiated by [DNR90, MK91, Mal93]. The DFT compactly represents a dataset as a collection of sine and cosine signals. The main disadvantages of Fourier-based representation is the insufficient localization of spatial structures. However, the DFT gained popularity in DVR applications with the first algorithms that performed rendering directly in the compression domain. Fourier-compressed rendering bases on the *Fourier projection-slice theorem* [Lev92], which says that the inverse transform of a slice extracted from the frequency

domain representation of a volume corresponds to a projection of the volume in a direction perpendicular to the slice. First, a volume dataset is transformed by a 3D DFT into the frequency domain. Then, during reconstruction, a projection slice is selected within the frequency domain according to the view angle and reconstructed with only using a faster 2D inverse DFT. The great thing is that this theorem potentially reduces the rendering complexity of a volume dataset of size N^3 to $O(N^2 \log(N))$. In fact, [CYH*97] developed the first DFT-based method that renders directly from the frequency domain, i.e., no prior decompression step is needed for the rendering. It should be noted, however, that this approach works only in very limited settings, as it does not allow the direct application of non-trivial shading models, transfer functions, and perspective projections.

Discrete Hartley Transform (DHT) For frequency domain rendering, the Hartley transform is often used instead of the DFT [Mal93, CYH*97, VKG04], since DFT and DHT basically do the same transform. However, the Hartley transform works with real numbers only, whereas Fourier transform outputs complex values, too. As for DFT, the main application is frequency-domain rendering.

Discrete Cosine Transform (DCT) Another frequency domain transform is the discrete cosine transform. DCT can be seen as a simplified alternative to DFT since it uses only cosine waves as a basis representation. Furthermore, similar to DHT, the DCT has the property of mapping real input values to real output values. The DCT, however, does not allow direct rendering from the compression domain. Therefore, Yeo et al. [YL95] developed a strategy to decompress blocks of data on demand. DCT is often chosen in practice since it offers independent bases as well as a fast forward and inverse transforms.

Discrete Wavelet Transform (DWT) In contrast to DFT and DCT that transform the data only into the frequency domain, the discrete wavelet transform transforms the input data into the frequency domain, while maintaining the spatial domain. Maintaining the spatial information is a great advantage for direct volume rendering, in particular for block-based and multiresolution volume rendering. In DWT, the dataset is filtered along every data direction with a given function/basis/filter. This separates the data according to a low-pass and a high-pass filter into coefficients. The low-pass filtered data results in the approximation coefficients, while the high-pass filtered data results in the detail coefficients. In multilevel wavelet transform (e.g., applied to multiresolution volume visualization), the approximation coefficients are recursively processed again by a DWT.

DWT is near lossless in general, but integer DWT methods can be fully lossless. However, the wavelet decomposition is often further reduced by thresholding the coefficients, which are close to zero. This allows a sparse and compact representation with a limited error for the lossy approximation. The loss of data is furthermore dependent on the choice of wavelet. There are reversible wavelet functions, which allow a lossless reconstruction, and there are irreversible wavelet functions, which result in a lossy approximation.

DWT was introduced for compressed volume rendering by Muraki [Mur93] who triggered a great many of follow-up works [Wes94, GEA96, GLDH97, LGK97, IP99, Rod99, KS99, NS01, GWGS02, LLYM04, WSKW05, WQ05, She06, KLW*08, WM08]. The discrete wavelet packet transform (DWPT) offers an even richer decomposition since, both, the detail and the approximation coefficients are further decomposed. The method, that has found application in volume segmentation [ASA11], is not however typically applied in volume rendering. The work of Westermann et al. [Wes94] indicates that the wavelet transform is particularly suited for multiresolution volume visualization. Guthe et al. [GWGS02] created a block-wise hierarchical decomposition (octree) of volume data followed by an entropy encoding of wavelet coefficients introducing a priority-based decompression by using block caching and projective classification during rendering. This approach is currently shared by many compressed rendering systems.

Laplacian Transform/Pyramid The Laplacian transform or the Laplacian pyramid are formed by pre-applying a Gaussian filter, expansion and prediction of values. During the rendering process, each voxel can be reconstructed on the fly. Due to its pyramid data structure, matches well for multiresolution representations, and shares many similarities with the wavelet transform. A volume compressed DVR application by Laplacian pyramid was presented in [GY95, SW03].

Burrows-Wheeler Transform (BWT) The transform by Burrows-Wheeler 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. After this re-sorting, the data blocks are organized in a more compression-friendly way, ready for example to apply entropy coders. Komma et al. [KFDB07] compared the compressibility of BWT against others for volume data compression.

Karhunen-Loeve Transform (KLT) KLT is a linear transform that removes the redundancy by decorrelating the data – closely related to PCA (also referred to as *eigenvector transform*). KLT estimates the covariance matrix and computes the eigenvectors. In terms of decorrelation, the Karhunen-Loeve transform is better than the DCT. Disadvantages of the KLT include that the bases are not independent and that there are no fast forward and inverse transforms. Fout et al. [FM07] selected KLT for a better quality decorrelation. Since they combine KLT with vector quantization the reconstruction is mostly offline, prior to rendering, when producing the vector quantization dictionary. To overcome the dependent bases they use block-wise decomposition.

Tensor Approximation (TA) The general idea behind tensor decomposition is to approximate a volume dataset with

SVD/PCA-like tools. The most common TA approaches are higher-order extensions of the matrix singular value decomposition: the volume (or small volume blocks), is interpreted as a multi-dimensional input dataset in array form, i.e., a tensor, which is factorized into a sum of rank-one tensors (CP decompostion) or into a product of a core tensor and matrices), i.e., one for each dimension (Tucker decomposition) [KB09]. TA is thus a method that works with learned data-specific bases. Data reduction is obtained by reducing the rank of the reconstruction. Suter et al. [SZP10] have shown that rank-reduced TA is more able to capture non-axis aligned features at different scales than wavelet transforms using the same amount of coefficients. Since TA, similarly to SVD/PCA, extracts specific features based on statistical properties like the major direction, the learned bases can be used both for data compression/reduction and feature extraction. TA has been recently applied interactive large volume visualization, using a single TA for every brick in a multiresolution hierarchy [SIM*11]. The same work introduced a tensor-specific quantization scheme and an optimized GPU reconstruction method.

Dictionaries for Vector Quantization The idea of dictionaries or codebooks is to represent the input volume with a set of pre-defined or learned dictionary entries (codewords). For that purpose, the initial volume is divided into small data blocks. Then, an algorithm searches for every block the dictionary entry that best matches this data block. Finally, the volume is represented by an index list, where each index corresponds to a data block and points to a certain dictionary entry. The method is very popular in the volume rendering literature since decompression is thus extremely fast.

Dictionaries exploit the fact that many subregions of the volume exhibit a similar 3D pattern that can be encoded with dictionary entries. When choosing enough entries, the dictionary approach can be lossless, but typically little or no compression is achieved. Most applications thus limit the choice of dictionary entries what is referred to as *vector quantization* (VQ) (data blocks can be seen as vectors). Critical issues for a good VQ performance are first the dictionary size and second the chosen dictionary generation algorithm. Data-specific dictionaries learned from a the original volume data tend to perform better than dictionaries based on predefined mathematical basis. One main problem of learned dictionaries is the time needed in pre-processing to obtain a good dictionary, specially, when dealing with massive volumes. Many methods have been proposed aiming at finding best dictionary (see survey by Lu and Chang [LC10]) and Knittel et al.'s [KP09] PCA-based dictionary generation algorithm). *Hierarchical vector quantization* (HVQ) is an improved VQ scheme that has the ability to efficiently process multi-dimensional data and is based on a multiresolution covariance analysis of the original domain.

Vector quantization was first introduced in volume rendering by [NH92, NH93]. The first GPU implementation of HVQ has been proposed by Schneider et al. [SW03] and Fout et al. [FM07] have combined HVQ with a previous domain transformation. The error obtained by just using one dictionary entry to encode voxel blocks can be improved by combining it with many different techniques (e.g., residual VQ [PK09b, PK09a]). However, dictionary size imposes a hard limit on achievable quality and compression rate of vector quantization solutions [Ela10].

Dictionaries for Sparse Representation Modeling The concept of dictionary is much more general than the concept of VQ outlined above. By representing the dictionary as a matrix $D = \mathbb{R}^{m \times K}$ that contains K prototype signal-atoms for columns, $\{d_j\}_{j=1}^{K}$, a linear combination of these atoms can be used to represent (or approximate) any signal $S \in \mathbb{R}^m$. In this view, the dictionary concept can be used to replace the transformations discussed above. These dictionaries can be either a mathematical model (e.g., wavelets), or learned from the data. Dictionaries of the first type are characterized by an analytic formulation, and thus a fast reduced-memory implicit implementation, while those of the second type deliver increased flexibility and the ability to adapt to specific signal data. While historically signal decomposition was performed by projecting signals on an orthogonal or bi-orthogonal basis, there is now much interest in the use of over-complete dictionaries, i.e., containing linearly dependent vectors, which are very well suited for sparse coding, an area which has witnessed a growing interest in the recent years (see survey of Rubinstein et al. [RBE10]). The sparse coding problem can be viewed as a generalization of VQ, in which we allow each input signal to be represented by a linear combination of dictionary elements instead of a single ones. Therefore the coefficients vector is now allowed more than one nonzero entry, and these can have arbitrary values. Recent architectures based on the sparse representation of voxel blocks have demonstrated state-of-the-art performance on very large static and dynamic volumes [GIM12].

Fractal Compression Fractal compression relies on the fact that some parts of the dataset (volume) often resemble other parts (self-similarity). Similar to vector quantization, the idea of fractal compression is to work on block entities and to find for each entity a most similar one. Compared to vector quantization, where each block stores only a reference to its codeword, fractal compression stores per data block the position of another different-sized data block of the same volume and additionally stores the affine transformations needed convert the reference block to the current block. During fractal compression, for each block (range block) a larger most similar data block (domain block) of the same volume is found. At the end of the matching algorithm, only the transformations from a domain block to the range block is stored. The decompression is performed by applying the stored transformations iteratively until the final volume does not improve any longer. The compression is effective since the transformation data oc-

cupies little space, however, the search procedure itself is extremely computationally intense. While the decompression is straightforward, the rendering from compressed data is problematic (due to the iterative reconstruction algorithm). So far, the main application is thus off-line volume compression [CHF96]. Resctricted forms of this method might, however, find application in real-time rendering in combination with dictionary methods, as already proposed for image coding [HS00].

An orthogonal approach to the presented models had been proposed by Wang et al. [WWLM11], where they use grid-deformations prior to data reduction. In order to better preserve features during data reduction (downsampling), the data grid is transformed according to user defined regions. They report improved results to wavelets and pure downsampling.

Table 1 summarizes the most popular compact models and their essential features for compressed volume rendering. For example, we mention models, which are inherently hierarchical - even tough we are aware of the fact that many of the models can be constructed into a hierarchical data structure.

3.2. Thresholding and Quantization

The idea of thresholding and quantization is to further reduce the amount of data by neglecting some of the coefficients of the produced compact model. One simple approach is to threshold coefficients that are close to zero (e.g., typical case for wavelets). Other approaches sort the values and truncate the least significant coefficients (e.g., PCA-like models). Dictionaries are often designed such that they explicitly include a quantization step (e.g., vector quantization), i.e., they only allow a limited number of dictionary entries (codewords) and the available dictionary entries are learned to best approximate a dataset. Another typical quantization step is to represent floating point coefficients with integer values. All those approaches have in common that they make a compact model lossy. However, the user can define what error is allowed to be introduced.

3.3. Encoding Models

The reduced and limited number of coefficients can be further compressed by additional encoding. Almost always, this is a lossless process. Well-known encoders are run-length encoders (RLE) or so called entropy encoders like Huffman coding and arithmetic coding. *Run-length encoders* analyze a stream of data by counting the number of equal values occurring after each other. This value count is then stored together with the actual value in the stream. The main idea of *entropy encoding* is to represent frequent voxel values with a short code and infrequent values with a longer code, i.e., the encoding is lossless. However, this variable length encoding is inconvenient to quickly reconstruct an arbitrary sequence of data. Therefore, even tough entropy encoders are effective, i.e., result in a high compression ratio, they also imply more work during decoding. That is why a lot of research (mainly for wavelet coefficient encoding) was put into finding encoding schemes, which have a fast random access for the reconstruction, e.g., [Rod99]. As we will see in the following sections, many of the systems thus use hybrid methods, e.g., using entropy coders to reduce on-disk storage, but performing entropy decoding before rendering time, which is performed on a more GPU friendly representation.

In compressed DVR, RLE was widely used since the early applications of volume rendering [ASK92, LL94, YL95, GEA96, KS99, KFDB07, WYM10]. Entropy encoders, too, are frequently used in compressed volume rendering: Mostly they used Huffman encoding [FY94, YL95, Rod99, GWGS02, LLYM04, WSKW05, KFDB07] or arithmetic encoding [GWGS02, XWCH03, KFDB07] also combined with RLE [GWGS02, SMB*03]. In order to accelerate the decoding of variable length codewords in entropy encoding *fixed length Huffman coders* were explicitly used and combined with a RLE [GWGS02, She06, KLW*08]. Another attempt to accelerate the decoding was achieved by using a less known entropy encoder, the *Golomb-Rice encoder*, which [WQ05] favored over arithmetic coding and Huffman coding since it has better random access due to its lower computational complexity. Other approaches (mainly for wavelet coefficient encoding) [LGK97, Rod99, IP99] tried to fully avoid entropy encoding and RLE by using *significance maps* of wavelet coefficients and bit-wise encodings.

Komma et al. [KFDB07] benchmarks and analyses lossless compression encodings for compressed volume data and covers entropy encodings, run-length-encoding (RLE), variable bit length encoding (VBL), LZ77, LZW, ZIP, BZIP2, (lossless) JPEG-2000 until the wavelet transformation.

Finally, there exist models that have a particular hardware support during the decoding. An example is the *block truncation coding* (BTC). During BTC, the data is decomposed into equal-sized blocks. Then each block is truncated as long as a given criteria is satisfied. For instance, each block is truncated as long as the standard deviation and the average value of the original data are maintained. That means, the error tolerance (threshold level) is different per block. We can consider three adaptations of BTC that support hardware accelerated random access: (1) VTC [Cra04] which is the extension to 3D of S3TC [Bro01], (2) ETC2 [SP07] texture compression using invalid combinations, and (3) *adaptive scalable texture compression* (ASTC) [NLP*12], which shares some similarities with S3TC. Agus et al. [AGIM10] proposed a method to handle segmented datasets where small blocks are quantized to have only one or two different components inside. The encoded data is constituted by the type of block and the splitting plane which subdivides the block.

3.4. Models for Time-Varying Data

For real time rendering of time-varying volumetric datasets the use of a compressed representation is essentially a must-

Compact Models and Encodings	First in DVR	Transform domain	Analytical	Learned	Hierarchical	Lossless	Near lossless	Lossy
Discrete Fourier Transform (DFT)	[DNR90, Mal93]	✓	✓				✓	
Discrete Hartley Transform (DHT)	[Mal93]	✓	✓				✓	
Discrete Cosine Transform (DCT)	[YL95]	✓	✓				✓	
Discrete Wavelet Transform (DWT)	[Mur93]	✓	✓		✓	✓	✓	
Laplacian Pyramid/Transform	[GY95, SW03]	✓	✓		✓		✓	
Burrows-Wheeler Transform (BWT)	[KFDB07]	✓	✓			✓		
Karhunen-Loeve Transform (KLT)	[FM07]	✓		✓			✓	
Tensor Approximation (TA)	[SZP10, SIM*11]	✓		✓		✓		
Dictionaries for Vector Quantization	[NH92, SW03]		✓	✓				✓
Dictionaries for Sparse Representation Modeling	[GIM12]	✓	✓	✓		✓		✓
Fractal Compression	[CHF96]			✓				✓

Table 1: *Features of compact data representation models.*

have feature. In fact, due to the limited bandwidth is not feasible to interactively load decode and render each time step in GPU in real time. Three main compression strategies can be highlighted in the time-varying domain.

First, the previous compression methods can be extended in the dimensionality from 3D to 4D. This kind of approach can exploit the correlation of voxels in subsequent frames to obtain good compression ratios, but on the other hand it requires to have in memory a small set of frames for rendering a single time step [ILRS03, WWS03, WWS*05]. Moreover, difficulties frequently arise due to the discrepancy between the spatial and temporal resolutions, and handling these dimensions equally often prevents the possibility of detecting the peculiar coherence of the temporal domain.

The second possibility is to treat the fourth dimension in a different way, as it is done in video encoding. This approach, too, has the advantage of obtaining good compression rates exploiting the temporal coherence [Wes95, MS00, WGLS05, She06, KLW*08, MRH10, WYM10, SBN11, JEG12]. Voxel data is delta encoded with respect to some reference time step(s), which must be already available to decode the current one. In this framework, key frames plus delta encoding of arbitrary time steps can theoretically be used to provide better access to random points in the sequence. In such approaches it is also common to exploit temporal coherence first performing a spatial subdivision and then encoding compactly corresponding blocks of adjacent time steps.

The third way is to encode each time step separately with one of the previous static volumes compression methods. The main advantages of this method are simplicity of implementation and temporal full random access [GIM12], but it can not reach the compression rates of the previous two approaches not exploiting temporal coherence.

Finally there are some methods, which are based on a mix of previous approaches like [LMC01, BCF03, FAM*05, NIH08, WYM08, CWW11]. A survey on time-varying volumetric datasets has been presented by De Floriani et al. [WF08].

3.5. Discussion

The first two important choices made in compressed volume rendering architecture are how to transform data in order to sparsify its representation and how to efficiently encode the sparse representations.

Of the various methods presented, sparse coding and most of the methods based on analytical transforms supports a variable compression ratio, from extremely lossy to (near) lossless. Vector quantization with limited dictionary size has, on the other hand, a limit on achievable quality.

It is important to emphasize that the different data models presented do not exclude each other. Among the methods based on analytical transforms, the most commonly applied transform in compact volume rendering application is the wavelet representation, which is often combined with vector quantization and run-length encoding of coefficients. Block truncation approaches are, instead, the most common choice for fixed hardware implementations, due to very simple decompression and fast random access into the compressed volume. They are, however, not fully scalable in terms of quality, and exhibit a lower signal over noise ratio with respect to other transform methods.

PCA-like models as tensor approximation (TA) were only recently introduced to direct volume rendering. While the TA-approach demonstrated that the learned and rank-reducible bases are extremely compact and are able to capture non-axis aligned features at different scales, current tests indicate that wavelet approaches are still advantageous to reconstruct the

overall statistical distribution of a dataset at coarser resolutions [SZP10].

Vector quantization is often chosen and applied to transform-domain data. Recently, overcomplete dictionaries (i.e., linearly dependent vectors as dictionary entries) have drawn attention and have triggered recent contributions in sparse compact models for DVR. Early results in this area indicate state-of-the-art performance for sparse-coding techniques [GIM12]. Moreover, a sparse-coding implementation can be seen as a generalization of HVQ and VQ.

The selection of the compressed data model in a compressed volume rendering application is not only dictated by compression performance target, but has to take into account the applicability to the model to very large data, and the capability to provide fast and localized decoding at rendering time. These issues are discussed in the following sections.

4. Preprocessing and Encoding

The preprocessor has to transform the input volume into a compressed representation using the models described in Sec. 3. In order to support adaptive, transient and local reconstruction, compression algorithms are often paired with a block structure embedded within a LOD hierarchy. Since preprocessing is applied when volumes hardly fit into the available resources, it should be designed to be scalable in terms of memory and time. In particular, it needs to work in an out-of-core manner, to avoid hard limits on input data size, and in a parallel setting, to speed-up processing on current multi-core/multi-cpu environments. During the preprocessing phase, two steps are typically identified: (1) a global data processing step, that requires access to all the input data, typically to learn some representation, and (2) local data processing steps, which only need to access independent blocks, possibly using the representation learned in the global step to encode them.

4.1. Global Processing

Global processing of the volume is typically employed in dictionary-based methods in order to learn a dictionary that will later be used in the local step to independently encode the small blocks. Early transform coding methods [Lev92, Mal93] also performed a global preprocessing step transforming the whole volume with DFT or DHT transforms. On more recent approaches, however, these transforms are only applied to small independent blocks.

Finding the dictionary requires to analyze all the input data. This global step working on billions of samples has two main problems: first the algorithm must be able to handle all the data within the available memory resources and also within acceptable times. Second handling these huge datasets could introduce numerical instability problems which must be properly handled to produce reliable results.

A common way to limit the computational complexity of the global step, is to split the volume into manageable subvolumes [FM07] and for each of them find a proper dictionary. This scheme has the benefit that is easy to implement, but on the other hand increases the size of the data which need to be stored (one dictionary for each subvolume) and tends to introduce discontinuity among subvolumes encoded with different dictionaries. For these reasons, other more sophisticated methods have been devised.

4.1.1. Vector Quantization: Global Processing

The global step of vector quantization algorithms generally do not require to access all the input volume at the same time, but need only to keep some local statistics for each group of similar elements, while streaming over all the input elements. Such *online learning* algorithms do not incur into memory limitation issues, having memory occupancy proportional to the size of the dictionary, but generally have long processing times, due to a large number of streaming iterations and their slow convergence behavior. Preprocessing is generally based on the generalized Lloyd algorithm [Llo82, LBG80] which exploit these two conditions: first, given a codebook the optimal partition is found on nearest neighbor assignment, second, given a partition the optimal codevector is the centroid of the partition. Thus starting from an initial partition we can easily associate all the vectors to their corresponding cells, next all the cell representatives are updated to the cell centroids. These two steps are iterated until convergence. Various volume rendering vector quantization papers are based on variations of this technique, like [NH92, NH93]. In particular is important how initial partition is identified: Knittel et al. [KP09] find initial codevectors based on principal component analysis (PCA) and uses error-directed subdivision of the eigenspace in reduced dimensionality, they also include shape-directed split decisions based on eigenvalue ratios to improve the visual appearance. However, dictionary size imposes a hard limit on achievable quality and compression rate of vector quantization solutions [RBE10]. In hierarchical vector quantization [SW03] the learning of the codebooks is performed through a modification of the LBG [LBG80] algorithm. Each new seed is not inserted randomly, but a split plane is identified in the cell with the higher residual distortion. The split plane is selected performing principal component analysis of the cell, producing two sub-cells of roughly equal residual distortions. Finally the iterative subdivision is concluded applying some [LBG80] LBG-steps as post-refinement to relax the centroids quickly generating stable Voronoi regions. To improve the quality of the decoded data Knittel et al. [PK09a, PK09b] use Residual Vector Quantization, performing VQ encoding of the signal and of its residuals due to the encoding error, up to a certain level (4 in the paper). VQ codebook is computed in parallel distributing the dataset and the codebook and performing the computation with GPU and OpenMP.

Fout and Ma [FM07] subdivide the volume in small blocks

which are processed with Karhunen-Loeve transform. Each block is encoded with VQ and then decoded. If the residual is above a certain threshold it is encoded with a second dictionary. The process is applied again. In such a way three bands (low-, band-, high-pass) are identified and encoded with three dictionaries. Most of the elements are encoded with just the low-band, reducing occupancy, while retaining the possibility of encoding more details for higher-frequency signals with the other two codebooks. The inverse transform of the Karhunen-Loeve is computationally intensive thus during preprocessing partial reprojections are precomputed and stored in the codebooks instead of the coefficients.

Fractal compression [CHF96] also shares common problems with vector quantization methods. A block-based search is performed looking for near-matches using some transformations of the blocks in the dictionary (color and voxel permutations). The fractal component in this compression scheme comes when there are no near-matches in the search space. Then the blocks are subdivided and new candidates for near-matches are searched for. Such a technique is very computationally intensive and is not currently applied on very large volumes.

4.1.2. Sparse Coding: Global Processing

A major improvement over Vector Quantization methods is to employ a representation in which each block is represented as a sparse linear combination of few dictionary elements. Recent years have witnessed a growing interest in such sparse representations (see the recent survey of Rubinstein et al. [RBE10] for an overview of the state-of-the-art). Data-specific dictionaries learned from each training set tend to perform better than dictionaries based on a mathematical model of the data (e.g., wavelets) [Ela10]. Gobbetti et al. [GIM12] employ the K-SVD algorithm [AEB06] for dictionary training, a state-of-the-art method in the domain [Ela10]. Performing K-SVD calculations directly on massive input volumes would, however, be prohibitively expensive. Even though memory problems could be circumvented with emerging online training techniques [MBPS10, SE10], massive datasets still lead to large computational time and possible numerical instabilities. For Bidirectional Texture Functions (BTF) compression, Ruiters and Klein [RK09] attacked this problem by reducing the dataset prior to K-SVD training through a truncated SVD. Instead, Gobbetti et al. [GIM12] perform data reduction by smartly subsampling and reweighting the original training set, applying the concept of *coreset* [AHPV05, CS07].

4.2. Independent Block Encoding

Since all current architectures are based on independent blocks, all current compressors/encoders currently perform an encoding pass working locally. This pass can eventually use information gathered during a global step, as in learned dictionary techniques. This local encoding step is very often applied within a multiresolution framework, which encodes blocks at various resolution levels, typically obtained by digital filtering and subsampling. Since the processing is local, this step requires memory resources independent from the input size. Moreover, encoding time grows linearly with dataset size, and parallelization can be achieved by distributing blocks to different threads/processes. The differences between the various approches mostly rely on the specific compact model employed for block coding.

4.2.1. Transform Models: Local Processing

In these methods, the volume is transformed into another representation projecting it into a set of pre-defined bases, like what is done with DFT, DHT, DCT or wavelets. After that transform, the data is more suitable for compression because most of the signal information tends to be concentrated in a few low-frequency components, which can be compacted with quantization and various types of entropy coding techniques. Fourier Volume Rendering algorithms perform rendering directly from transformed data thanks to the *Fourier projection-slice theorem*. In [CYH*97] the volume is subdivided into subcubes which are transformed by DFT, then frequency coefficients are quantized and organized with a zigzag order and compacted with run-length and Huffman encoding. DCT transform method [YL95] partitions data in blocks which are further subdivided into subblocks. Then subblocks are transformed by the DCT, like what is done in 2D with JPEG lossy compression. DC coefficients are grouped quantized and delta encoded, while AC coefficients are compressed as previously described for the DFT approach of [CYH*97].

Many approaches apply a wavelet transformation then quantize and encode the resulting coefficients. Lippert et al. [LGK97] use B-spline wavelets, in preprocessing to compute wavelet coefficients and positions which are arranged with decreasing importance, then are compressed with delta encoding, quantization, run-length and Huffman coding. Insung et al. [IP99] and Kim et al. [KS99] partition the volume into small blocks which are further subdivided into subblocks which are then transformed with a three dimensional Haar wavelet applied respectively two and three times. Then coefficients are quantized and encoded. Rodler [Rod99] presents a compression schema based on wavelets that treats 3D data as separate correlated slices in order to exploit results in the area of video coding like using temporal/spatial coherence amongst these slices. In [NS01, GWGS02] volume is partitioned in blocks which are transformed independently respectively with the the biorthogonal 9/7-tap Daubechies wavelet filter and biorthogonal spline wavelets. Wetekam et al. [WSKW05] propose a multiresolution approach where they build an octree storing for each node its wavelet transform which is computed as in [GWGS02]. Data is decoded at run-time through a FPGA hardware decoder. Wu et al. [WQ05] uses a modified 3-D dyadic wavelet trans-

form tailored to volumetric medical images and an optimized Rice code of very low complexity scalable lossless coding scheme for compression of volumetric medical images.

In the Laplacian pyramid [GY95] approach a sequence of low pass filtering operations, and computation of deltas between the filtered expanded signal and the original one is performed. The delta values are encoded with variable length encoding and at run time the root plus the residuals permit to reconstruct the original signal.

The tensor approximation (TA) approach used for DVR as presented by Suter et al. [SIM*11] builds a multiresolution structure where each block is encoded into a rank-reducible basis learned from the input dataset. Generally in TA, a tensor decomposition is applied to a N-dimensional input dataset $\mathcal{A} \in \mathbb{R}^{I_1 \times I_2 \times \cdots \times I_N}$ (stored as data array), which is either decomposed (1) into a sum of R rank-one tensors (CP model), or (2) into a product of N basis matrices $\mathbf{U}^{(n)} \in \mathbb{R}^{I_n \times R_n}$ and an N-dimensional core tensor $\mathcal{B} \in \mathbb{R}^{R_1 \times R_2 \times \cdots \times R_n}$ (Tucker model). The Tucker model used in Suter et al. [SIM*11] is based on a higher-order singular value decomposition (HOSVD) – also known as higher-order PCA. The HOSVD is applied to the input dataset along every data direction. In fact, an *alternating least squares* (ALS) algorithm – called the *higher-order orthogonal iteration* (HOOI) – optimizes the basis matrices one after each other. The core tensor is only produced once the basis matrices cannot be improved anymore. The core tensor $\mathcal{B}$ represents a projection of the input dataset $\mathcal{A}$ onto the basis matrices $\mathbf{U}^{(n)}$, i.e., the coefficients of the core tensor show the relationship between the original data and the bases. Since the core tensor of the Tucker model is all-orthogonal (i.e., produced from orthonormal basis matrices), a rank-reduction similar to the one of the matrix SVD can be applied to the tensor decomposition.

The Tucker model is defined for a *multilinear rank* $(R_1, R_2, \ldots, R_N)$, i.e., one rank R_n per dataset direction. In fact, the HOOI is already produced for a given initial multilinear rank, which typically corresponds already to a data reduction when starting with an initial rank $R_n = \frac{I_n}{2}$ [SIM*11]. However, further adaptive rank reductions (i.e., data reduction) can be applied after the initial decomposition. Even tough the ordering of the coefficients in the core tensor is not strictly decreasing, as in the matrix SVD case, in practice it can be shown that progressive tensor rank reduction in the Tucker model works well for adaptive visualization. For multiresolution volume rendering with equally sized blocks, the multilinear rank is typically equal as well, i.e., $R_1 = R_2 = R_2$. Finally, Suter et al. [SIM*11] quantize the floating point coefficients of the tensor decomposition with a TA optimized quantization scheme (8bit encoding for core tensor coefficients, 16bit encoding for basis matrices).

It was shown that by applying the tensor rank reduction in the Tucker model to volume datasets, features at different scales can be extracted and visualized and a good approximation quality and data reduction level could be achieved.

4.2.2. Dictionary Based Methods: Local Processing

In vector quantization the block encoding part is generally trivial, being performed finding the block nearest neighbor in the dictionary. Instead in sparse representation the block encoding phase requires a more complex algorithm: each block of size $m = M^3$ is mapped to a column vector $\mathbf{y} \in \mathbb{R}^m$. Given the dictionary $\mathbf{D} \in \mathbb{R}^{m \times K}$ computed in the learning phase, a sparse representation of $\mathbf{y}$ of at most S entries in each column can be found by solving the following problem:

$$\min_{\lambda_i} \left\{ \|\mathbf{y}_i - \mathbf{D}\lambda_i\|^2 \right\} \text{ subject to } \|\lambda_i\|_0 \leq S \qquad (1)$$

where $\|\lambda_i\|_0$ is the number of non-zero entries in λ_i. The exact solution to this problem is NP-hard, but several efficient greedy pursuit approximation algorithms exist [Ela10]. Gobbetti et al. [GIM12] employ ORMP via Choleski Decomposition [CARKD99] because of its simplicity and efficiency. Compression of $\mathbf{y}_i$ is thus achieved by storing the sparse representation of the vector λ_i, specifying the indices of its nonzero elements and their magnitudes.

4.3. Quantization and Encoding

Most of the presented transformation and dictionary based compression techniques apply a quantization and encoding step to further diminish the memory occupancy of the new data representation. The encoding can vary from simple quantization techniques to more sophisticated variable length encoding approaches. In most of the early compression algorithms the variable bit length encoding was preferred due to its capability of achieving high compression rates. Many algorithms, in a special way the ones that use pre-defined bases [CYH*97, YL95, GY95, LGK97, IP99, KS99, Rod99, NS01, WSKW05, WQ05], quantize and then encode the coefficients resulting from transformation, exploiting various combinations of entropy coders. Later there has been some techniques using fixed length encoding at the cost of worst compression ratios. The scheme based on fixed length Huffman plus run length encoding revealed really high decompression speed [GWGS02, She06, KLW*08]. Variable length approach is less suited for a GPU implementation than the fixed length encoding which instead is widely used from more recent methods which exploit full GPU decoding. Variable bit length schemes still result useful in data compression to leverage disk and network fetching times. But up to now the corresponding decompression step is executed in CPU to unpack coefficients before uploading them to GPU.

4.4. Preprocessing of Time-varying Data

Handling time-varying data generally leads to mixing and matching the previously discussed methods to build compressed structures while trying to exploit temporal coherence. Many of the contribution in this are are mostly at the system level.

A few methods build a *Time Space Partitioning Tree* structure which is made by an octree whose nodes contain a binary tree with temporal information [MS00]. In [WCCA04, She06] a similar approach has been applied to a a 3D wavelet transformation of the original data using a *Wavelet-based TSP*. Westermann et al. [Wes95] and Wang et al. [WYM10] build a hierarchical structure for each time step then group and encode together space-corresponding nodes. Fout et al. [FAM*05] propose two methods for multivariate time-varying volumes: one grouping voxels in temporal domain and on the other in spatial domain, then applying vector quantization algorithms to corresponding blocks. Nagayasu et al. [NIH08] pack three time steps into RGB channels then use standard VTC compression for 3D-RGB data. Ko et al. [KLW*08] classify each time step as intra coded frame or predictive frame. All frames are encoded with Haar wavelet, but predictive frames are also expressed as delta with respect to previous frames and then encoded. Mensman et al. [MRH10] perform delta encoding of block data with respect to the corresponding block of the previous time step. Each block is then quantized with a proper bit count per voxel depending on the range of data inside the brick. Finally all blocks are encoded with a LZO algorithm. She et al. [SBN11] first perform delta encoding of adjacent time steps, then learn this data with hierarchical vector quantization. Jang et al. [JEG12] propose an approach similar to [KLW*08] but encode data with a functional representation based on spherical ellipsoids derived by [JBL*06]. Wavelet transform was frequently applied to time-varying volume data, too, in particular from these systems [Wes95, GS01, WS04, WGLS05, She06, WS09, MA09, WYM10].

4.5. Discussion

Producing compact representations from massive volumetric datasets requires application developers to take into account a number of requirements.

First of all, the output should be produced in a way that will allow the streaming and rendering components to perform efficiently. This leads to the generation of datasets that are most commonly based on a multiresolution hierarchy of blocks, where each block is compactly encoded using one or more of the models presented in Sec. 3. Such an organization into independent blocks is currently the best suited method to support localized access and data loading.

Second, the processing technique should be scalable: processing should be applicable to arbitrarily large datasets, without hard limits due to memory usage, numerical errors, or exceedingly long times. While methods based on independent block coding (such as all the methods based on analytical models) do not pose particular problems in this regard, dictionary based methods are more problematic. In particular, in the sparse-coding area, on-line techniques currently do not offer the same performance of off-line training methods, and the current approaches for reducing training datasets are either intrinsically limited (localized dictionaries [FM12]) or require an a-priori model of the significance of the data [GIM12]. Dictionary training for massive volumes thus remains an open research area.

Finally, the particular data model selected should support efficient decoding by the renderer. As we will see in Sec. 5, not all the models support, for instance and efficient GPU decoder. Some of the solutions presented, thus, imply the careful selection of hybrid models. For instance, Fout and Ma [FM07] precompute at processing time the partial reprojections required for KLT decoding, and use VQ to compactly encode the results. A common approach is also to use a generic entropy encoding scheme on top of a representation well suited for GPU decoding (e.g., [NIH08, MRH10]).

5. Decoding and Rendering

Interactive visualization of compressed massive volume data is generally paired with multiresolution data structures commonly computed in a preprocessing step. At run time a working set, adaptively selected from this structure, is incrementally updated to provide a good representation of the volume according to transfer function and viewing criteria. Images are generated by frame-buffer compositing of individual blocks rendered through slicing [LMHJ99, BNS01, GS04] or ray-casting [HQK05, KWAH06]. Small blocks, required for adaptivity, lead, however, to high communication overhead and pressure on compositing hardware. For this reason, researchers have introduced out-of-core GPU methods, which traverse adaptively maintained space-partitioning GPU data structures covering the full working set [Lju06, GMI08, CNLE09]. Another choice is to employ per-block multiresolution, where each block is encoded with a few levels of detail like what happens with hierarchical vector quantization [SW03], or in many wavelet representations [KS99, Rod99, IP99, LLYM04] Both the global and per block multiresolution strategies could be present inside the same structure like what is presented by Wetekam et al. [WSKW05].

As volume rendering demands interactive performance, the interaction between decompression and rendering is a fundamental issue. As it has been discussed in Sec. 2, asymmetric encoding/decoding schemes are preferred, as the available hardware resources in the GPU is the major limiting factor affecting GPU-based architectures for rendering from compressed data.

5.1. Architectures

Current architectures propose to accomplish the decompression at different stages of the visualization pipeline since the nature of the decompression methods could differ notably. There are several points at the visualization pipeline eligible to perform data decompression:

Decompression architectures	References	GPU		General	
		bandwidth	memory	Quality scalability	HQ filtering
Full working set CPU decompression	[KLW*08, She06, GWGS02, IP99]	HIGH	HIGH	LOW	✓
Full working set GPU decompression	[SIM*11, MRH10, WSKW05]	LOW	HIGH	MEDIUM	✓
HW-supported GPU decompression	[NLP*12, IGM10, YNV08, SP07, NIH08, Cra04, Fen03, Bro01, INH99]	MEDIUM	MEDIUM	MEDIUM	✓
Custom GPU decompression	[PK09b, NIH08, SW03]	LOW	LOW	MEDIUM	
Partial working set GPU decompression	[GIM12, WYM10, FM07, FAM*05, VKG04]	LOW	LOW	HIGH	✓

Table 2: *Compression schemes*

5.1.1. Decompression in CPU before Rendering

Decompress the data stream while loading from disk reduces the storage needs and allow faster and remote access to data. This is the proffered stage for including lossless decompression stages, typical in pure CPU based implementations [She06]. The main disadvantages of such approaches are the large memory resources needed to fit the data of the full decompressed working set and the waste of the GPU memory bandwidth due to data transmission in a not compressed format. Since for time-varying datasets those tasks need to be performed almost for each frame, using only a CPU-based decompression scheme is generally not sufficient. In fact, CPU decompression is typically used in a first lossless decompression step within hybrid CPU-GPU architectures [MRH10, NIH08] while data is maintained in some compact representation for later decompression steps in the GPU. From an architectural point of view, recently Beyer et al. [BHJ*11] proposed a multi-resolution run-time brick reconstruction from 2D slices where sampling is performed on a virtual octree volume. Authors don't explicitly mention details about a data compression strategy but it could probably be integrated at least in a per-slice manner when loading data from disk to the CPU/s.

5.1.2. Decompression in GPU before Rendering

Some authors have proposed to speed-up decompression by using the GPU. A straight-forward implementation of this idea needs a prior decompression of the full working set in GPU memory. Such kind of architecture has been proposed by Wetekam et al. [WSKW05], in that case using an FPGA decoder to decompress wavelet coefficients based on Huffman variable bit decoding. More recently Mensmann et al. [MRH10] visualize time-varying animations and assembly several volume subblocks at each frame in a single final three-dimensional texture before the rendering. Suter et al. [SIM*11] proposed a GPU block-based decoder based on multiscale tensor reconstruction. In general, the idea of decompressing the full working set before rendering introduces the main disadvantage of limiting the maximum working set size by its size in uncompressed form. Nevertheless, this solution may work out fine for cases of time-varying visualizations where the information contained in each single frame fits completely in the GPU memory, which may not always be the case.

5.1.3. Decompression during Rendering

In this case data is transmitted in a compressed form from the CPU to the GPU, but decoding occurs on-demand during the rendering process, and the entire dataset is never fully decompressed in GPU memory. This way, memory bandwidth is better exploited when the data is compressed for its transmission to the GPU. There are two main strategies at this point, the first option is to perform a pure random-access to the volume data and the second choice is to support a full or partial decompression of the working set before its rendering.

Pure random-access. In this case data decompression is performed each time the renderer needs to access to a single voxel in the volume. This strategy corresponds with a literal interpretation of the compression-domain volume rendering concept. Only few hardware implementations fulfill the requirements for on-demand, fast and spatially independent decompression on the GPU, which is required for maximum benefits [FM07].

A first group of methods is formed by those directly supported by the graphics hardware. The simplest hardware-supported fixed-rate block-coding methods (e.g., OpenGL VTC [Cra04, NIH08] have, however, limited flexibility in terms of supported data formats and achievable compression. GPU implementations of per-block scalar quantization can be considered as HW supported random-access (e.g., [YNV08, IGM10]). Recently, Nystad et al. [NLP*12] presented a fixed-rate lossy texture compression method

called Adaptive Scalable Texture Compression (ASTC) which shares some similarities with other block-based methods like S3TC [INH99] which is in turn an adaptation of Block Truncation Coding [DM79].

The second category of pure random-access approaches are those that, while not being directly supported by the graphics hardware, they could be easily implemented using shaders or GPGPU programming techniques. Examples of this kind of methods are GPU implementations of VQ or HVQ [SW03, FM07]. In general those GPU implementations of random-access are costly in performance and rarely implement multisampling or high-quality shading without relying on deferred filtering strategies.

Local and Transient Decoding. A second choice to avoid such problems is to perform a partial decompression of the working set and perform some sort of interleaving between decompression and rendering. This approach better exploit the GPU memory bandwidth and, in general, improves the usage of the available GPU memory resources. Deferred filtering solutions [FAM*05, WYM10] exploit spatial coherence and supports high-quality shading decompressing groups of slices which can be reused during rendering for gradient and shading computations. Nevertheless, this solution was proposed as a single resolution system and therefore limiting the size of the volume data set. An extension of this idea to work with multiresolution data representations was proposed in the COVRA [GIM12] architecture. In that case, the volume is subdivided in a small number of octrees of compressed blocks that are rendered and composited in a GPU ray-caster in front-to-back order. Working over these subtrees of the original octree makes possible to better exploit the GPU memory resources as data can be discarded as soon as it has been used for rendering. Authors extended deferred filtering to supported multiresolution representations.

5.2. Decoding from Compact Representations

The different compression methods presented in Sec. 3 can be employed within one or many of the previously described architectures. In general, it can distinguished between decompression methods that operate directly in the transformed domain (pure compression-domain approaches), methods that perform a per-voxel access (pure random-access) or per-block accesses with the idea to amortize the decompression computation, supporting high quality filtering and shading for a group of voxels. Next, we describe how those methods perform the decoding and rendering in the existing architectures, giving priority to those methods for which a GPU implementation exists.

5.2.1. Transform/Decomposition Models

Original transform-domain approaches operate directly in the transformed domain to perform the accumulation process [Lev92, Wes94]. In general the decoding consists in the inverse stages of the forward encoding. In the case of the Fourier transform the projection slice is extracted thanks to the *Fourier projection-slice theorem* [Lev92] using the *Inverse 2D Fast Fourier Transform*. Viola et al. [VKG04] proposed the following single-resolution strategy for a slice-based rendering. After the compressed data is uploaded to the GPU, the rendering is divided into two stages: (i) slicing in the frequency domain and (ii) the inverse transform. The slicing refers to the re-sampling of the projection slice from the 3D frequency data. For this, the authors used a setup where the slice is perpendicular to the viewing direction intersecting the frequency volume in the origin of the frequency spectrum. When changing the viewing direction, the texture is rotated around the frequency volume origin. The algorithm performs in $O(M^2)$ for a $M \times M$ slice, instead of $O(N^3)$ that would be required to perform the projection of a $N \times N \times N$ volume in the spatial domain. The inverse transform re-sample the frequency slice back to the spatial domain and results in the projection of accumulated intensities into the frame-buffer. In the original paper the authors proposed to use the Hartley transform because of its memory efficiency with respect to the Fourier transform, but equivalent implementations for fast inverse transformations exist for other algorithms (e.g FFT, DCT, etc.) and GPU implementations are common, at least for the 2D case. This theorem allows the generation of attenuation-only renderings of volume data in $O(N^2 logN)$ time for a volume of size N^3.

In the case of the Wavelet transform, as with the Fourier transform there also exist the possibility to perform a projection in the frequency domain an employ the inverse wavelet transform to go back to the spatial domain. The volume is commonly decoded per blocks and in many cases combined with some sort of multiresolution data structure [Wes94, IP99, KS99, GWGS02]. During the traversal of such multiresolution data structures, the blocks belonging to the working set are filtered out. Wavelets can be combined with many other compression techniques, (e.g., run-length encoding [LGK97], Rice-coding [WQ05]). The decoding of the needed parameters are commonly plugged in a sequential order from one method to the next one. Another authors combine the wavelets with time-varying data structures, e.g., *wavelet-based time-space partitioning* (WTSP) [WS04, WGLS05, She06]. Ko et al. [KLW*08] eliminated the hierarchical decompression dependency commonly found in the hierarchical wavelet representation methods, by applying video based compression techniques that lead to a more efficient reconstruction of data along the time axis. Garcia et al. [GS05] proposed a GPU-based algorithm for reconstructing 3D wavelets using fragment programs and tileboards to distribute the 3D wavelet coefficients. An FPGA hardware implementation has been proposed by Wetekam et al. [WSKW05] in which authors proposed a HW implementation of the reverse wavelet transformation.

In the case of the Karhunen-Loeve Transform (KLT), Fout et al. [FM07] optimized the inverse transform computation

precomputing partial re-projections and storing them in what they called *associated codebooks*. Thus, a portion of the volume can be represented by four progressive levels of approximation $x_i = \sum_{p=1}^{P} C_p^*(k_p, i))$ where C_p^* represents the associated dictionaries for the $p-th$ volume partition.

The tensor approximation method has been proposed in combination with a multiresolution octree hierarchy [SIM*11]. The decompression of the full working set is performed before the rendering stage using a per-brick solution. The reconstruction, proposed and implemented in the GPU, is based on the Tucker formulation and implements the tensor times matrix (TTM) multiplications. In their optimized implementation, each kernel thread is in charge of the decompression of one voxel with a computational complexity $O(M)$, instead of $O(M^3)$, and grows only linearly with the rank M. In the hierarchical tensor-based transform proposed by Wu et al. [WXC*08] the entire domain of the input volume is recursively partitioned into smaller blocks and a truncated tensor-product basis for each block is defined. Over this idea, the original tensor is represented as a summation of incomplete tensor approximations at multiple levels. The tensors at each level are subdivided in residual tensors passed from the higher levels.

In dictionary based methods the pre-processing could be a tedious and costly process, but in contrast the decoding and rendering are quite easy and fast, and several GPU implementations exist. In methods like VQ, HVQ [SW03, FM07], RVQ [PK09a, PK09b] or *flag based classified hierarchical vector quantization* (FCHVQ) [ZFW12, ZYX*11] each voxel is commonly reconstructed on-the-fly just accessing to the selected and needed codewords, thus supporting per-voxel random-access. Dictionaries can be uploaded to GPU memory and just one additional memory indirection per codeword is needed. The computational complexity of reconstructing VQ is linear $O(N)$, being N the number of blocks. In HVQ the complexity is $O(N \times M)$, where M is the number of dictionaries (note for M = 1 you get the complexity of VQ). The sparse coding of voxel blocks employ a linear combination of different codewords of a dictionary, thus the computational complexity of the method remains very similar to HVQ and is $O(N \times S)$ where S is the sparsity. Similarly to VQ based implementations, each voxel can be decompressed in parallel by accessing to the corresponding codewords and their weighting coefficients, like proposed by Gobbetti et al. [GIM12].

5.2.2. Thresholding and Quantization Models

Random-access decompression methods can be easily implemented in GPU as a per-block scalar quantization [YNV08, IGM10]. In those methods a minimum and maximum value per block is computed, as well as a local index in between those values. Finally, the decompressed value is computed as $v_{ijk} = min + (max - min) * \frac{index}{2^{i_{bits}}-1}$ where i_{bits} is the number of bits used to represent the index values.

Smelyanskiy et al. [SHC*09] employed a differences based compression scheme. During decompression, they essentially load the data for multiple elements (eight in the case of 128-bit SSE), and expand the stored differences into 16-bit values. These values are then added to the base value to obtain the final values. Since different blocks may have different number of bits per element, they pre-compute and store the different shift patterns in a separate table, and at run-time simply look up the appropriate data shuffle patterns to expand the data into 16-bit values. In practice, the overhead of lookups is negligible.

Mensmann et al. [MRH10] proposed an hybrid scheme. The decompression part has to resolve the variable-length coding, resolve the delta encoding, and perform the brick assembly. As the variable-length coding requires different addressing modes based on with how many bits a brick is stored, the authors implemented individual kernels for handling each of the supported bit lengths. Implementing the delta encoding is trivial, as the kernel just needs to add the calculated value to the existing value in the volume instead of overwriting it.

5.2.3. Encoding Models

Most of the hardware-supported fixed-rate block-coding techniques are nowadays based on simple fixed-rate block-coding methods (e.g., VTC [Cra04] or S3TC [INH99]). For example VTC, and similarly other DXT variations of S3TC, approximate voxel values by selecting between different linear interpolation modes in a lookup table, then applying the selected formula over a precomputed subset of representative values. This interpolation is done independently for each block, and thus exploits spatial coherence in small blocks. Note that this interpolation is quickly done by the on-the-fly decoder implemented as a hardware component. In addition, the on-the-fly decompression does not require additional memory to store the decompressed data, because the entire volume is not decompressed at a time. Thus, rendering can be performed directly accessing to the coordinates of the voxel and reconstructing the value in real-time.

Adaptive Scalable Texture Compression (ASTC) [NLP*12] is a fixed-rate lossy texture compression method. ASTC is scalable and adaptive in the sense that the proposed hardware supports different block-sizes configurations covering a wide range of bit rates (e.g., for 3D textures, 6x6x6 blocks result in 0.56 bps and 3x3x3 in 4.74 bps). ASTC proposes a general method for representing value sequences using a fractional number of bits per value, and a system for constructing per-texel color weights from sparse samples. Authors claim that their results are competitive with the most advanced HW-supported formats in use today, and to improve over industry standards such as DXT [Bro01], ETC2 [SP07] and PVRTC [Fen03]. Nevertheless, the 3D part of the full ASTC specification is still undergoing detailed evaluation for its inclusion in next OpenGL releases [Ell12].

Fraedrich et al. [FBS07] proposed a general approach for applying common sequential data compression schemes to

blocks of data, which can be decoded on-the-fly by the graphics processing unit (GPU) in a single render pass. The block is decomposed into parallel stripes of voxels. For decoding, a quad is rendered with the size of the index texture and each fragment decodes one stripe of the block. The decoded scalar values are written into different color channels of multiple render targets, which are used as textures for volume rendering in a subsequent render pass.

5.2.4. Discussion

Local and transient reconstruction is a key feature for a compressed volume renderer. In order to support it, decisions must be taken both in terms of compact data model and rendering architectures.

A number of solutions have been presented that perform rendering directly in the transform domain, e.g., using Fourier, Hartley, or wavelet transform. This approach maximally reduces rendering complexity and memory pressure. However, full rendering in transform domain is only applicable to a very reduced number of situations (e.g., X-ray imaging), since it does not efficiently support perspective rendering, transfer functions, or complex shading. Other authors have also proposed to exploit dictionaries, e.g., obtained by vector quantization, to speed-up some portion of rendering (e.g., global illumination or occlusion) by performing computations in the transform domain (see Hadwiger et al. [HLSR09] for a survey). In the general case, however, current compressed direct volume renderers need to decode data prior to rendering.

Of the three possible architectures presented, only the solutions based on pure voxel-level random access on the GPU and the on-line partial decoders during rendering achieve the goal of avoiding full data decompression. Shaded rendering with good quality interpolation when using pure voxel-level random access is, however, currently doable only when using hardware-supported methods, thus limiting the flexibility in terms of supported data formats and achievable compression. The most flexible approach thus remains partial decompression.

Doing decompression at rendering time thus requires the ability to decompress blocks of data on-the-fly within real-time constraints. Few of the current methods currently have such a performance, and most of them only within very limited settings. For instance, analytical transforms, Tensor approximation [SIM*11] and Sparse Coding [GIM12] achieve their best performance when using variable per-block rank/sparsity, which are not support by current GPU reconstructors due to difficulty in handling complex memory layouts. Efficiently handling variable local bit rate, while not impacting on reconstruction speed, is an area of research.

Efficient decompression methods are particularly important in view of the current move to lighter client, especially in the mobile settings. In this case, also due to the more constraining network speeds, hybrid architectures that perform partial decoding at data loading time, and another decoding step at rendering time seem promising.

In terms of quality, one of the limitations shared by all current block-based compressed volume-rendering architectures is that reconstruction artifacts at low bit rates manifest themselves as visible discontinuities between adjacent blocks. It should be noted that such an artifact also appears when employing LOD scheme, g even in conjunction with uncompressed data, since the different local resolutions within adjacent blocks then cause discontinuities in the rendered image. In the case of uncompressed multiresolution data, this artifact has been handled with various data replication schemes [WWH*00, GS04] or with interblock interpolation methods [LLY06, BHMF08, HLPS08]. Deblocking has not, however, been tackled in the volume rendering literature when dealing with compressed data.

6. Conclusions

The investigation of compressed volume rendering methods to dynamically render models whose size exceeds current hardware capabilities has been, and still is, a very active computer graphics research area, which is obviously impossible to fully cover in a short survey. In this state-of-the-art report, we have provided a characterization of the basic concepts common to all current architectures and of the requirements of each component. After reviewing the most common compact data models used as underlying data representations, we discussed the issues related to compression and encoding, as well as decoding and rendering, with a special focus on GPU techniques fully supporting transient and local decoding.

Even though the domain is mature and has a long history, open problems remain. The current trend consists in moving all the decompression to the last part of the pipeline, being able to keep all data compressed in GPU, thus better exploiting available bandwidth and memory resources. Current fully GPU accelerated methods have some limitations in terms of compression rate vs. quality. In particular, few of the methods can efficiently cover the full spectrum from extreme compression to (near)lossless compression. Wavelet-based techniques and recent methods based on tensor analysis or sparse coding theoretically cover a wide rate and quality spectrum, but most of current implementations supporting fast GPU decoding do not optimally support variable bit-rate encoding. Compression performance is thus currently not on par with the performance obtained with GPU techniques.

Moreover, block based techniques inherently introduce blocking artifacts. Various approaches have been proposed to reduce perceptual effects in image or video applications and many of these methods focus on "post-processing", that is, processing images when received or viewed. So far, none of these methods have been applied in volume rendering. Moreover, elimination of compression artifacts in volumetric multiresolution structures is still an open problem.

On current desktop platforms, the compression methods surveyed in this report will have their main future applicability when dealing with time-varying data, multi-volume visualization, or multi-attribute datasets. While current high end graphics system currently have enough GPU memory for handling common single scalar dataset, efficiently handling multiple time steps, multiple volumes, and multiple modalities within a single visualization is not within reach. Handling these cases imposes research not only at the level of compression models, but also at the level of adaptive rendering algorithm, and at the system level.

Finally, given the increasingly wide diffusion of mobile platforms with high quality graphics support, such as tablets or smart-phones, and the increasing interest in moving many infrastructures to "the cloud", networked and mobile volume visualization is expected to gain increased interest. Algorithms which perform pretty well in current desktop platforms are, however, still far from being integrated on the constrained environments provided by mobile devices.

Acknowledgments. This work is partially supported by the People Programme (Marie Curie Actions) of the European Union's Seventh Framework Programme FP7/2007-2013/ under REA grant agreements n°290227 (DIVA) and n°251415 (GOLEM).

References

[AEB06] AHARON M., ELAD M., BRUCKSTEIN A.: K-svd: An algorithm for designing overcomplete dictionaries for sparse representation. *IEEE Transactions on Signal Processing 54*, 11 (Nov. 2006), 4311–4322. 10

[AGIM10] AGUS M., GOBBETTI E., IGLESIAS GUITIÁN J. A., MARTON F.: Split-voxel: A simple discontinuity-preserving voxel representation for volume rendering. In *Proc. Volume Graphics* (2010), pp. 21–28. 7

[AHPV05] AGARWAL P., HAR-PELED S., VARADARAJAN K.: Geometric approximation via coresets. *Combinatorial and Computational Geometry 52* (2005), 1–30. 10

[ASA11] ALZUBI S., SHARIF M., ABBOD M.: Efficient implementation and evaluation of wavelet packet for 3d medical image segmentation. In *Proceedings of IEEE International Workshop on Medical Measurements and Applications Proceedings (MeMeA)* (may 2011), pp. 619 –622. 5

[ASK92] AVILA R. S., SOBIERAJSKI L. M., KAUFMAN A. E.: Towards a comprehensive volume visualization system. In *Proceedings of IEEE Visualization* (1992), IEEE Computer Society Press, pp. 13–20. 7

[BCF03] BINOTTO A. P. D., COMBA J., FREITAS C. M. D. S.: Real-time volume rendering of time-varying data using a fragment-shader compression approach. In *Proceedings of IEEE Symposium on Parallel and Large-Data Visualization and Graphics* (2003), pp. 69–76. 8

[BHJ*11] BEYER J., HADWIGER M., JEONG W.-K., PFISTER H., LICHTMAN J.: Demand-driven volume rendering of terascale em data. In *Proceedings of ACM SIGGRAPH Talks* (2011), pp. 57:1–57:1. 13

[BHMF08] BEYER J., HADWIGER M., MÖLLER T., FRITZ L.: Smooth mixed-resolution GPU volume rendering. In *Proc. IEEE/EG Symposium on Volume and Point-Based Graphics* (2008), pp. 163–170. 16

[BNS01] BOADA I., NAVAZO I., SCOPIGNO R.: Multiresolution volume visualization with a texture-based octree. *The Visual Computer 17* (2001), 185–197. 12

[Bro01] BROWN P.: Ext texture compression s3tc. OpenGL Extension Registry, 2001. 7, 13, 15

[CARKD99] COTTER S. F., ADLER R., RAO R. D., KREUTZ-DELGADO K.: Forward sequential algorithms for best basis selection. In *Proceedings of Vision, Image and Signal Processing* (1999), vol. 146, IET, IET, pp. 235–244. 11

[CHF96] COCHRAN W. O., HART J. C., FLYNN P. J.: Fractal volume compression. *IEEE Transactions on Visualization and Computer Graphics 2*, 4 (Dec. 1996), 313–322. 7, 8, 10

[CNLE09] CRASSIN C., NEYRET F., LEFEBVRE S., EISEMANN E.: Gigavoxels: ray-guided streaming for efficient and detailed voxel rendering. In *Proceedings of symposium on Interactive 3D graphics and games* (2009), ACM, pp. 15–22. 12

[Cra04] CRAIGHEAD M.: Gl_nv_texture_compression_vtc. OpenGL Extension Registry, 2004. 7, 13, 15

[CS07] CZUMAJ A., SOHLER C.: Sublinear-time approximation algorithms for clustering via random sampling. *Random Structures & Algorithms 30*, 1-2 (2007), 226–256. 10

[CWW11] CAO Y., WU G., WANG H.: A smart compression scheme for gpu-accelerated volume rendering of time-varying data. In *Proceedings of the International Conference on Virtual Reality and Visualization* (2011), IEEE Computer Society, pp. 205–210. 8

[CYH*97] CHIUEH T.-C., YANG C.-K., HE T., PFISTER H., KAUFMAN A.: Integrated volume compression and visualization. In *Proceedings of IEEE Visualization* (oct. 1997), pp. 329 –336. 5, 10, 11

[DGY07] DIETRICH A., GOBBETTI E., YOON S.: Massive-model rendering techniques: A tutorial. *IEEE Computer Graphics and Applications 27*, 6 (nov/dec 2007), 20–34. 2

[DM79] DELP E., MITCHELL O. R.: Image compression using block truncation coding. *IEEE Transactions on Communications 27*, 9 (1979), 1335–1341. 14

[DNR90] DUNNE S., NAPEL S., RUTT B.: Fast reprojection of volume data. In *Proceedings of Conference on Visualization in Biomedical Computing* (may 1990), pp. 11 –18. 4, 8

[EHK*06] ENGEL K., HADWIGER M., KNISS J. M., REZK-SALAMA C., WEISKOPF D.: *Real-time volume graphics*. A K Peters, 2006. 1, 2, 4

[Ela10] ELAD M.: *Sparse and Redundant Representations*. Springer, 2010. 6, 10, 11

[Ell12] ELLIS S.: GL_KHR_texture_compression_astc_ldr. OpenGL (4.3 & ES 3) Registry, 2012. 1, 15

[FAM*05] FOUT N., AKIBA H., MA K.-L., LEFOHN A., KNISS J. M.: High-quality rendering of compressed volume data formats. In *Proceedings of EG/IEEE Symposium on Visualization* (2005). 8, 12, 13, 14

[FBS07] FRAEDRICH R., BAUER M., STAMMINGER M.: Sequential data compression of very large data in volume rendering. In *Proceedings of the Conference Vision, Modeling and Visualization* (2007), pp. 41–50. 15

[Fen03] FENNEY S.: Texture compression using low-frequency signal modulation. In *Proceedings of the ACM SIGGRAPH/EUROGRAPHICS conference on Graphics hardware* (2003), pp. 84–91. 13, 15

[FM07] FOUT N., MA K.-L.: Transform coding for hardware-accelerated volume rendering. *IEEE Transactions on Visualization and Computer Graphics 13*, 6 (2007), 1600–1607. 1, 3, 5, 6, 8, 9, 12, 13, 14, 15

[FM12] FOUT N., MA K.-L.: An adaptive prediction-based approach to lossless compression of floating-point volume data. (Accepted by VisWeek/Vis 2012), 2012. 12

[FY94] FOWLER J. E., YAGEL R.: Lossless compression of vol-

ume data. In *Proceedings of the Symposium on Volume Visualization (VVS)* (1994), ACM. 3, 7

[GEA96] GROSSO R., ERTL T., ASCHOFF J.: Efficient data structures for volume rendering of wavelet-compressed data. In *Proceedings of Winter School of Computer Graphics* (1996), Computer Society Press. 5, 7

[GIM12] GOBBETTI E., IGLESIAS GUITIÁN J., MARTON F.: Covra: A compression-domain output-sensitive volume rendering architecture based on a sparse representation of voxel blocks. *Computer Graphics Forum 31*, 3pt4 (2012), 1315–1324. 1, 6, 8, 9, 10, 11, 12, 13, 14, 15, 16

[GKY08] GOBBETTI E., KASIK D., YOON S.: Technical strategies for massive model visualization. In *Proc. ACM Solid and Physical Modeling Symposium* (2008), ACM Press, New York, NY, USA, pp. 405–415. 2

[GLDH97] GROSS M. H., LIPPERT L., DITTRICH R., HÄRING S.: Two methods for wavelet–based volume rendering. *Computers & Graphics 21*, 2 (1997), 237–252. 5

[GMI08] GOBBETTI E., MARTON F., IGLESIAS GUITIÁN J. A.: A single-pass GPU ray casting framework for interactive out-of-core rendering of massive volumetric datasets. *The Visual Computer 24*, 7-9 (July 2008), 797–806. 12

[GS01] GUTHE S., STRASSER W.: Real-time decompression and visualization of animated volume data. In *Proceedings of IEEE Visualization* (2001), IEEE Computer Society Press, pp. 349–356. 12

[GS04] GUTHE S., STRASSER W.: Advanced techniques for high quality multiresolution volume rendering. *Computers & Graphics 28* (2004), 51–58. 12, 16

[GS05] GARCIA A., SHEN H.: Gpu-based 3d wavelet reconstruction with tileboarding. *The Visual Computer, Special Issues of Pacific Graphics 21*, 8–10 (2005), 755–763. 14

[GWGS02] GUTHE S., WAND M., GONSER J., STRASSER W.: Interactive rendering of large volume data sets. In *Proceedings of IEEE Visualization* (2002), pp. 53–60. 5, 7, 10, 11, 13, 14

[GY95] GHAVAMNIA M. H., YANG X. D.: Direct rendering of laplacian pyramid compressed volume data. In *Proceedings of IEEE Visualization* (1995), IEEE Computer Society Press, pp. 192– . 5, 8, 11

[HLPS08] HEGE H., LAIDLAW D., PAJAROLA R., STAADT O.: Smooth mixed-resolution gpu volume rendering. In *IEEE/EG Symposium on Volume and Point-Based Graphics* (2008). 16

[HLSR09] HADWIGER M., LJUNG P., SALAMA C., ROPINSKI T.: Gpu-based volume ray-casting with advanced illumination. In *Eurographics Tutorials* (2009), pp. 39–211. 16

[HQK05] HONG W., QIU F., KAUFMAN A.: GPU-based object-order ray-casting for large datasets. In *Proceedings of Volume Graphics* (2005), Eurographics Association, pp. 177–186. 12

[HS00] HAMZAOUI R., SAUPE D.: Combining fractal image compression and vector quantization. *Image Processing, IEEE Transactions on 9*, 2 (2000), 197–208. 7

[IGM10] IGLESIAS GUITIÁN J. A., GOBBETTI E., MARTON F.: View-dependent exploration of massive volumetric models on large scale light field displays. *The Visual Computer 26*, 6–8 (2010), 1037–1047. 13, 15

[ILRS03] IBARRIA L., LINDSTROM P., ROSSIGNAC J., SZYMCZAK A.: Out-of-core compression and decompression of large n-dimensional scalar fields. *Computer Graphics Forum 22*, 3 (2003), 343–348. 8

[INH99] IOURCHA K., NAYAK K., HONG Z.: System and method for fixed-rate block-based image compression with inferred pixel values. US Patent 5,956,431, 1999. 13, 14, 15

[IP99] IHM I., PARK S.: Wavelet-based 3d compression scheme for interactive visualization of very large volume data. *Computer Graphics Forum 18* (1999), 3–15. 5, 7, 10, 11, 12, 13, 14

[JBL*06] JANG Y., BOTCHEN R. P., LAUSER A., EBERT D. S., GAITHER K. P., ERTL T.: Enhancing the interactive visualization of procedurally encoded multifield data with ellipsoidal basis functions. *Computer Graphics Forum 25*, 3 (2006), 587–596. 12

[JEG12] JANG Y., EBERT D. S., GAITHER K. P.: Time-varying data visualization using functional representations. *IEEE Transactions on Visualization and Computer Graphics 18*, 3 (2012), 421–433. 8, 12

[KB09] KOLDA T. G., BADER B. W.: Tensor decompositions and applications. *Siam Review 51*, 3 (Sep 2009), 455–500. 6

[KFDB07] KOMMA P., FISCHER J., DUFFNER F., BARTZ D.: Lossless volume data compression schemes. In *Proceedings of the Conference Simulation and Visualization* (2007), pp. 169–182. 1, 5, 7, 8

[KLW*08] KO C.-L., LIAO H.-S., WANG T.-P., FU K.-W., LIN C.-Y., CHUANG J.-H.: Multi-resolution volume rendering of large time-varying data using video-based compression. In *Proceedings of IEEE Pacific Visualization Symposium* (2008), pp. 135–142. 5, 7, 8, 11, 12, 13, 14

[KP09] KNITTEL G., PARYS R.: Pca-based seeding for improved vector quantization. In *Proceedings of International Conference on Imaging Theory and Applications* (2009). 6, 9

[KS99] KIM T.-Y., SHIN Y. G.: An efficient wavelet-based compression method for volume rendering. In *Proceedings of Pacific Conference on Computer Graphics and Applications* (1999), IEEE Computer Society Press, pp. 147–. 5, 7, 10, 11, 12, 14

[KWAH06] KAEHLER R., WISE J., ABEL T., HEGE H.-C.: GPU-assisted raycasting for cosmological adaptive mesh refinement simulations. In *Proceedings of Volume Graphics* (2006), Eurographics Association, pp. 103–110. 12

[LBG80] LINDE Y., BUZO A., GRAY R.: An algorithm for vector quantizer design. *IEEE Transactions on Communications 28*, 1 (1980), 84–95. 9

[LC10] LU T., C.Y. C.: A survey of vq codebook generation. *Journal of Information Hiding and Multimedia Signal Processing 1* (2010), 190–203. 6

[Lev92] LEVOY M.: Volume rendering using the fourier projection-slice theorem. In *Proceedings of Conference on Graphics Interface* (1992), pp. 61–69. 4, 9, 14

[LGK97] LIPPERT L., GROSS M., KURMANN C.: Compression domain volume rendering for distributed environments. *Computer Graphics Forum 16*, 3 (September 1997), 95–107. 5, 7, 10, 11, 14

[Lju06] LJUNG P.: Adaptive sampling in single pass, GPU-based raycasting of multiresolution volumes. In *Proceedings of Volume Graphics* (2006), Eurographics Association, pp. 39–46. 12

[LL94] LACROUTE P., LEVOY M.: Fast volume rendering using a shear-warp factorization of the viewing transformation. In *Proceedings of Conference on Computer graphics and interactive techniques* (1994), ACM, pp. 451–458. 7

[Llo82] LLOYD S.: Least squares quantization in pcm. *IEEE Transactions on Information Theory 28*, 2 (1982), 129 – 137. 9

[LLY06] LJUNG P., LUNDSTRÖM C., YNNERMAN A.: Multiresolution interblock interpolation in direct volume rendering. In *Proceedings of EuroVis* (2006), pp. 259–266. 16

[LLYM04] LJUNG P., LUNDSTROM C., YNNERMAN A., MUSETH K.: Transfer function based adaptive decompression for volume rendering of large medical data sets. In *Proceedings of IEEE Symposium on Volume Visualization and Graphics* (2004), pp. 25–32. 5, 7, 12

[LMC01] LUM E. B., MA K. L., CLYNE J.: Texture hardware assisted rendering of time-varying volume data. In *Proceedings of IEEE Visualization* (2001), IEEE Computer Society Press, pp. 263–270. 8

[LMHJ99] LA MAR E. C., HAMANN B., JOY K. I.: Multiresolution techniques for interactive texture-based volume visualization. In *Proceedings of IEEE Visualization* (Oct. 1999), pp. 355–362.

12

[MA09] MEFTAH A., ANTONINI M.: Scan-based wavelet transform for huge 3d volume data. In *Proceedings of Picture Coding Symposium (PCS)* (may 2009), pp. 1 –4. 12

[Mal93] MALZBENDER T.: Fourier volume rendering. *ACM Transactions on Graphics 12*, 3 (1993), 233–250. 4, 5, 8, 9

[MBPS10] MAIRAL J., BACH F., PONCE J., SAPIRO G.: Online learning for matrix factorization and sparse coding. *The Journal of Machine Learning Research 11* (2010), 19–60. 10

[MK91] MALZBENDER T., KITSON F. L.: A fourier technique for volume rendering. In *Proceedings Focus on Scientific Visualization* (1991), pp. 305–316. 4

[MRH10] MENSMANN J., ROPINSKI T., HINRICHS K.: A gpu-supported lossless compression scheme for rendering time-varying volume data. In *Proceedings of Volume Graphics* (2010), Westermann R., Kindlmann G. L., (Eds.), Eurographics Association, pp. 109–116. 8, 12, 13, 15

[MS00] MA K.-L., SHEN H.-W.: Compression and accelerated rendering of time-varying volume data. In *Proceedings of the International Computer Symposium-Workshop on Computer Graphics and Virtual Reality* (2000), pp. 82–89. 8, 12

[Mur93] MURAKI S.: Volume data and wavelet transforms. *IEEE Computer Graphics and Applications 13*, 4 (July 1993), 50–56. 5, 8

[NH92] NING P., HESSELINK L.: Vector quantization for volume rendering. *Workshop on Volume Visualization* (1992), 69–74. 6, 8, 9

[NH93] NING P., HESSELINK L.: Fast volume rendering of compressed data. In *Proceedings of IEEE Visualization* (1993). 6, 9

[NIH08] NAGAYASU D., INO F., HAGIHARA K.: Two-stage compression for fast volume rendering of time-varying scalar data. In *Proceedings of International Conference on Computer graphics and interactive techniques in Australasia and Southeast Asia (GRAPHITE)* (2008), ACM, pp. 275–284. 8, 12, 13

[NLP*12] NYSTAD J., LASSEN A., POMIANOWSKI A., ELLIS S., OLSON T.: Adaptive scalable texture compression. In *Proceedings of High Performance Graphics* (2012), Dachsbacher C., Munkberg J., Pantaleoni J., (Eds.), Eurographics Association, pp. 105–114. 7, 13, 15

[NS01] NGUYEN K. G., SAUPE D.: Rapid high quality compression of volume data for visualization. *Computer Graphics Forum 20*, 3 (2001), 49–57. 5, 10, 11

[PK09a] PARYS R., KNITTEL G.: Giga-voxel rendering from compressed data on a display wall. In *Proceedings of Winter School of Computer Graphics* (2009), pp. 73–80. 6, 9, 15

[PK09b] PARYS R., KNITTEL G.: Interactive large-scale volume rendering. In *Proceedings of High End Visualization Workshop* (2009). 6, 9, 13, 15

[RBE10] RUBINSTEIN R., BRUCKSTEIN A., ELAD M.: Dictionaries for sparse representation modeling. *Proceedings of the IEEE 98*, 6 (2010), 1045 –1057. 6, 9, 10

[RK09] RUITERS R., KLEIN R.: Btf compression via sparse tensor decomposition. In *Computer Graphics Forum* (2009), vol. 28, Wiley Online Library, pp. 1181–1188. 10

[Rod99] RODLER F. F.: Wavelet based 3d compression with fast random access for very large volume data. In *Proceedings of Pacific Conference on Computer Graphics and Applications* (1999), IEEE Computer Society Press, pp. 108–. 5, 7, 10, 11, 12

[SBN11] SHE B., BOULANGER P., NOGA M.: Real-time rendering of temporal volumetric data on a gpu. In *Proceedings of the International Conference on Information Visualisation* (2011), IEEE Computer Society, pp. 622–631. 8, 12

[SE10] SKRETTING K., ENGAN K.: Recursive least squares dictionary learning algorithm. *IEEE Transactions on Signal Processing 58*, 4 (2010), 2121–2130. 10

[SHC*09] SMELYANSKIY M., HOLMES D., CHHUGANI J., LARSON A., CARMEAN D. M., HANSON D., DUBEY P., AUGUSTINE K., KIM D., KYKER A., LEE V. W., NGUYEN A. D., SEILER L., ROBB R.: Mapping high-fidelity volume rendering for medical imaging to cpu, gpu and many-core architectures. *IEEE Transactions on Visualization and Computer Graphics 15*, 6 (Nov. 2009), 1563–1570. 15

[She06] SHEN H.-W.: Visualization of large scale time-varying scientific data. *Journal of Physics 46*, 1 (2006), 535–544. 5, 7, 8, 11, 12, 13, 14

[SIM*11] SUTER S. K., IGLESIAS GUITIÁN J. A., MARTON F., AGUS M., ELSENER A., ZOLLIKOFER C. P., GOPI M., GOBBETTI E., PAJAROLA R.: Interactive multiscale tensor reconstruction for multiresolution volume visualization. *IEEE Transactions on Visualization and Computer Graphics 17*, 12 (December 2011), 2135–2143. 1, 6, 8, 11, 13, 15, 16

[SMB*03] SCHELKENS P., MUNTEANU A., BARBARIEN J., GALCA M., I NIETO X. G., CORNELIS J.: Wavelet coding of volumetric medical datasets. *IEEE Transactions Medical Imaging 22*, 3 (2003), 441–458. 7

[SP07] STRÖM J., PETTERSSON M.: Etc2: texture compression using invalid combinations. In *Proceedings of ACM SIGGRAPH/EUROGRAPHICS symposium on Graphics hardware* (2007), pp. 49–54. 7, 13, 15

[SW03] SCHNEIDER J., WESTERMANN R.: Compression domain volume rendering. In *Proceedings of IEEE Visualization* (2003), pp. 293–300. 5, 6, 8, 9, 12, 13, 14, 15

[SZP10] SUTER S. K., ZOLLIKOFER C. P., PAJAROLA R.: Application of tensor approximation to multiscale volume feature representations. In *Proceedings of Vision, Modeling and Visualization* (2010), pp. 203–210. 6, 8, 9

[VKG04] VIOLA I., KANITSAR A., GRÖLLER M. E.: Gpu-based frequency domain volume rendering. In *Proceedings of spring conference on Computer graphics* (2004), ACM, pp. 55–64. 5, 13, 14

[WCCA04] WANG Z., CHUI C.-K., CAI Y., ANG C.-H.: Multidimensional volume visualization for pc-based microsurgical simulation system. In *Proceedings of VRCAI* (2004), Brown J. R., Cai Y., (Eds.), ACM, pp. 309–316. 12

[Wes94] WESTERMANN R.: A multiresolution framework for volume rendering. In *Proceedings of the Symposium on Volume Visualization (VVS)* (1994), ACM, pp. 51–58. 5, 14

[Wes95] WESTERMANN R.: Compression domain rendering of time-resolved volume data. In *Proceedings of IEEE Visualization* (1995), IEEE Computer Society Press. 8, 12

[WF08] WEISS K., FLORIANI L.: Modeling and visualization approaches for time-varying volumetric data. In *Proceedings of International Symposium on Advances in Visual Computing, Part II* (2008), Springer-Verlag, pp. 1000–1010. 8

[WGLS05] WANG C., GAO J., LI L., SHEN H.-W.: A multiresolution volume rendering framework for large-scale time-varying data visualization. In *Proceedings of Volume Graphics* (2005), Eurographics Association, pp. 11–19. 8, 12, 14

[WM08] WANG C., MA K.-L.: A statistical approach to volume data quality assessment. *IEEE Transactions on Visualization and Computer Graphics 14*, 3 (may-june 2008), 590–602. 5

[WQ05] WU X., QIU T.: Wavelet coding of volumetric medical images for high throughput and operability. *IEEE Transactions on Medical Imaging 24*, 6 (june 2005), 719–727. 5, 7, 10, 11, 14

[WS04] WANG C., SHEN H.-W.: *A Framework for Rendering Large Time-Varying Data Using Wavelet-Based Time-Space Partitioning (WTSP) Tree*. Tech. rep., Department of Computer and Information Science, The Ohio State University, 2004. 12, 14

[WS09] WOODRING J., SHEN H.-W.: Multiscale time activity data exploration via temporal clustering visualization spreadsheet.

IEEE Transactions on Visualization and Computer Graphics 15, 1 (Jan. 2009), 123–137. 12

[WSKW05] WETEKAM G., STANEKER D., KANUS U., WAND M.: A hardware architecture for multi-resolution volume rendering. In *Proceedings of the ACM SIGGRAPH/EUROGRAPHICS conference on Graphics hardware* (2005), pp. 45–51. 5, 7, 10, 11, 12, 13, 14

[WWH*00] WEILER M., WESTERMANN R., HANSEN C., ZIMMERMANN K., ERTL T.: Level-of-detail volume rendering via 3d textures. In *Proc.IEEE symposium on Volume visualization* (2000), pp. 7–13. 16

[WWLM11] WANG Y.-S., WANG C., LEE T.-Y., MA K.-L.: Feature-preserving volume data reduction and focus+context visualization. *IEEE Transactions on Visualization and Computer Graphics 17* (2011), 171–181. 7

[WWS03] WOODRING J., WANG C., SHEN H.-W.: High dimensional direct rendering of time-varying volumetric data. In *Proceedings of IEEE Visualization* (2003), pp. 55–. 8

[WWS*05] WANG H., WU Q., SHI L., YU Y., AHUJA N.: Out-of-core tensor approximation of multi-dimensional matrices of visual data. *ACM Transactions on Graphics 24*, 3 (July 2005), 527–535. 8

[WXC*08] WU Q., XIA T., CHEN C., LIN H.-Y. S., WANG H., YU Y.: Hierarchical tensor approximation of multi-dimensional visual data. *IEEE Transactions on Visualization and Computer Graphics 14*, 1 (Jan. 2008), 186–199. 15

[WYM08] WANG C., YU H., MA K.-L.: Importance-driven time-varying data visualization. *IEEE Transactions on Visualization and Computer Graphics 14*, 6 (2008), 1547–1554. 8

[WYM10] WANG C., YU H., MA K.-L.: Application-driven compression for visualizing large-scale time-varying data. *IEEE Computer Graphics and Applications 30*, 1 (Feb 2010), 59–69. 7, 8, 12, 13, 14

[XWCH03] XIONG Z., WU X., CHENG S., HUA J.: Lossy-to-lossless compression of medical volumetric data using three-dimensional integer wavelet transforms. *IEEE Transactions on Medical Imaging 22*, 3 (March 2003), 459–470. 7

[Yan00] YANG C.-K.: *Integration of Volume Visualization and Compression: A Survey*. Tech. rep., SUNY SB, 2000. 1

[YGKM08] YOON S., GOBBETTI E., KASIK D., MANOCHA D.: *Real-time Massive Model Rendering*, vol. 2 of *Synthesis Lectures on Computer Graphics and Animation*. Morgan and Claypool, August 2008. 2

[YL95] YEO B.-L., LIU B.: Volume rendering of DCT-based compressed 3D scalar data. *IEEE Transactions on Visualization and Computer Graphics 1*, 1 (1995), 29–43. 5, 7, 8, 10, 11

[YNV08] YELA H., NAVAZO I., VAZQUEZ P.: S3dc: A 3dc-based volume compression algorithm. *Computer Graphics Forum* (2008), 95–104. 13, 15

[ZFW12] ZHAO L.-P., FANG M., WEI Y. W.: An efficient compressed volume rendering algorithm based on gpu. *Advanced Materials Research 433 - 440* (2012), 5448–5452. 15

[ZYX*11] ZHAO L.-P., YUE G., XIAO D.-G., ZHOU X., YU X., YU F.: A content-based classified hierarchical vector quantization algorithm for volume compression. *Journal of Software 6*, 2 (2011), 322–330. 15

Appendix A: STAR Authors

Marcos Balsa Rodríguez Marie Curie Early Stage Researcher in the Visual Computing group of the CRS4 research center. He holds a MS degree from the Polytechnic University of Catalonia (UPC) in Spain. His research interests cover scientific visualization, massive model rendering, parallel programming, volume rendering and mobile programming.

Enrico Gobbetti Director of Visual Computing at the CRS4 research center. His research spans many areas of computer graphics and is widely published in major journals and conferences. Volume compression, processing, streaming, and rendering are among his researched topics. Enrico holds an Engineering degree (1989) and a Ph.D. degree (1993) in Computer Science from the Swiss Federal Institute of Technology in Lausanne (EPFL).

José A. Iglesias Guitián Researcher in the Visual Computing group at the CRS4 research center. Holds a Masters degree in Computer Science (2006) from University of A Coruña, Spain, and a PhD degree in Electronics and Computer Engineering (2011) from University of Cagliari, Italy. His current research interests are in volume visualization, compressed data representations, GPU algorithms. He recently joined the Graphics and Imaging Lab at the University of Zaragoza as a Marie Curie Postdoc Researcher.

Maxim Makhinya Researcher at the Visualization and Multimedia Lab at the University of Zurich, Switzerland. He holds a Ph.D. degree (2012) in computer science from the University of Zurich. His research interests include volume visualization and parallel data rendering.

Fabio Marton Senior researcher in the Visual Computing group at the CRS4 research center. He holds a Laurea (M.Sc.) degree (1999) in Computer Engineering from the University of Padua, Italy. His current research interests include out-of-core data processing, multiresolution modeling and time-critical rendering.

Renato Pajarola Head of the Visualization and MultiMedia Lab and Professor at the Department of Informatics at the University of Zurich, Switzerland. His research spans a wide range of topics from interactive 3D computer graphics and geometry processing to high-performance scientific visualization. He received a Dipl. Inf-Ing ETH engineering as well as a Dr. sc.-techn. degree from the Swiss Federal Institute of Technology (ETH) Zürich in 1994 and 1998 respectively.

Susanne K. Suter A Ph.D. student in computer science at the Visualization and Multimedia Lab at the University of Zurich, Switzerland. Her research covers large volume visualization from compact and compressed datasets, feature extraction as well as the application of mathematical frameworks to visualization. She holds a M.Sc. degree (2005) in computer science from the University of Zurich.