Vision, Modeling, and Visualization (2014)
J. Bender, A. Kuijper, T. von Landesberger, H. Theisel and P. Urban (Eds.)

Real-time deblocked GPU rendering of compressed volumes

Fabio Marton[1], José A. Iglesias Guitián[1,2], Jose Díaz[1] and Enrico Gobbetti[1]

[1]Visual Computing Group, CRS4, Pula, Italy – http://www.crs4.it/vic/
[2]Universidad de Zaragoza, Spain

Abstract

The wide majority of current state-of-the-art compressed GPU volume renderers are based on block-transform coding, which is susceptible to blocking artifacts, particularly at low bit-rates. In this paper we address the problem for the first time, by introducing a specialized deferred filtering architecture working on block-compressed data and including a novel deblocking algorithm. The architecture efficiently performs high quality shading of massive datasets by closely coordinating visibility- and resolution-aware adaptive data loading with GPU-accelerated per-frame data decompression, deblocking, and rendering. A thorough evaluation including quantitative and qualitative measures demonstrates the performance of our approach on large static and dynamic datasets including a massive 512^4 *turbulence simulation (256GB), which is aggressively compressed to less than* 2 *GB, so as to fully upload it on graphics board and to explore it in real-time during animation.*

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

1. Introduction

Volume compression tightly coupled with adaptive GPU-based direct volume rendering has been shown to be an effective solution to explore large static and dynamic volumetric datasets in local and distributed settings [BRGIG*14]. Adopting a compression-domain adaptive rendering approach, capable to maintain data in compressed format at all stages of the rendering pipeline, while loading only the data required for a particular view, makes it possible to minimize latency and overcome hard GPU memory size limitations, especially for massive, time-varying, or multi-volume visualization.

In such a compressed volume rendering architecture, decompression should ideally be *transient* and *local*, that is, a fully reconstructed volume should never be produced, and reconstruction should be performed in parallel for different volume areas. Current methods achieve these goals through independent coding of small volume blocks, using asymmetric compression schemes designed to provide fast decoding at run-time at the expense of increased (but high quality) encoding time [BRGIG*14]. Thanks to efficient parallel GPU implementations, modern block-based compression-domain GPU direct volume renderers are capable to achieve impressive performance in terms of rendering quality and speed. However, the independent decoding of volume blocks leads, especially at moderate and low bit rates, to visible discontinuities between adjacent blocks. Such aggressive compression are especially important for massive time-varying datasets, so as to fit data on GPU in order to perform real-time visualization during real-time simulation playback.

The term *deblocking* refers to approaches for improving visual quality by smoothing the sharp edges which may appear between blocks when block coding techniques are used. While in the literature there are many techniques presented for deblocking images or videos, no method has been, so far, applied to volume rendering. Deblocked volume rendering significantly differs from image/video processing: data is presented through a complex rendering computation, rather than just directly mapping it to the output; access to data blocks is view-dependent and not performed in a fixed streaming fashion; most methods use fixed-size encoding with variable errors, rather than predetermined quantization thresholds in conjunction with variable-rate encoders.

In this paper, we introduce a specialized architecture which efficiently performs high quality shaded rendering of massive static and dynamic datasets by closely coordinating visibility- and resolution-aware adaptive data loading with GPU-accelerated per-frame data decompression, deblocking, and rendering. The architecture is based on the concept of deferred filtering [KLF, FAM*05, WYM10, GIM12], i.e., a multi-pass approach in which portions of data are reconstructed in native formats into a temporary buffer before being processed for rendering using GPU hardware filtering capabilities. Our novel contributions are the following:

- we introduce the first compression-domain GPU volume rendering architecture that reduces blocking artifacts through post-process deblocking performed at rendering time; the method does not require any modification in the encoding method and uses a specialized decompress-filter-and-render approach with pluggable compression methods and deblocking filters;
- we generalize deferred filtering for GPU volume rendering to multi-resolution settings combining levels of detail with adaptive loading and visibility culling;
- we propose a novel and general deblocking filter based on projecting voxel scalar values onto *rational Bézier* curves capable to reduce compression artifacts across block boundaries while preserving existing features in the volume; the filter works without knowledge of the compression technique.

As demonstrated by our qualitative and quantitative evaluation (Sec. 6), our method is capable to guarantee real-time performance on standard graphics PCs, while improving visual quality for massive static and dynamic datasets. In particular we show that aggressive compression of massive time varying datasets makes it possible to fully store data on the graphics board for interactive exploration of animated sequences.

2. Related Work

Our architecture extends and combines state-of-the-art results in compression, filtering, and GPU volume rendering. In the following, we only discuss the approaches most closely related to ours. We refer the reader to very recent surveys [BRGIG*14, BHP14] for more information.

Adaptive volume rendering for massive datasets requires the combination of visibility and level-of-detail culling, for removing the data not required for a particular image, with out-of-core compressed data management techniques. This typically leads to adaptive loading from compressed data representations organized into space-partitioned multi-resolution blocks [HBJP12, Eng11, GMI08]. In this context, GPU decompression during rendering is of great importance to save storage space and bandwidth at all stages of the processing and rendering pipelines. This requires, however, support for on-demand, fast and spatially independent decompression on the GPU [FM07]. The simplest hardware-supported fixed-rate block-coding methods (e.g., OpenGL VTC [Cra] or per-block scalar quantization [YNV08, IGM10]) support general random access, but have limited flexibility in terms of supported data formats and achievable compression. This led to the development of a variety of more elaborated techniques, all based on the concept of independent decoding of volume blocks [BRGIG*14]. While a few methods employ the GPU only to accelerate decoding, storing data in decompressed form for further processing [WSKW05, MRH10, SIM*11], the most advanced techniques [VKG04, FM07, WYM10, GIM12, TBR*12] interleave decompression with rendering stages in order to decrease memory needs. Such *deferred filtering* solutions already demonstrated their suitability for high-quality rendering, as, by decoupling data decoding from data sampling using a multi-pass approach, they can harness the power of hardware filtering operations for multi-sampling and/or high-quality shading. With the possible exception of the COVRA architecture [GIM12], these methods have so far been limited to single-resolution slice-based rendering. In this work, we introduce a deblocking stage and generalize the method to multi-resolution settings, incorporating adaptive loading, levels of detail and visibility culling.

Deblocking solutions are meant to improve visual quality by reducing the block artifacts caused by independent block coding while preserving sharp edges. In the context of volume rendering, solutions have been presented only for removing local discontinuities when interpolating among uncompressed blocks at different level of details [WWH*00, GS04, LLY06, BHMF08], however deblocking has not been tackled when dealing with compressed volumes.

The image- and video-processing literature presents a wide variety of approaches that can be classified into in-loop and post-loop filtering, depending on where and how the deblocking operation is performed [LLL07]. Only post-loop techniques, which perform deblocking on the presented image, can achieve deblocking without the original image and video, and do not require modification of encoding/decoding methods. For this reason, they are the most appropriate for incorporation in a volume rendering architecture capable to support multiple encoding methods. A number of post-loop methods, however, work directly in the compression domain [LY04, WZF04, ADF04] and are therefore usable only in conjunction with specific encodings. Most generic techniques perform instead a filtering operation over adjacent decoded blocks, usually conceptually arranged in three main phases, first performing edge-detection over adjacent block boundary values, then classifying discontinuities as compression artifacts or as genuine high-frequency signals, and finally smoothing out only the edges marked as artifacts. Many approaches employ advanced knowledge on per-block quantization errors to differentiate between true and false edges [LJL*03, ASD05, WB08], and are therefore hard to apply to volumetric data with fixed-rate but variable error encodings, while other techniques perform statistical image analysis phases to discriminate between features and artifacts [LW03, KVS04, KCJ07]. While these methods would be applicable, their cost is non-negligible when applied to 3D datasets. Moreover, all these image/video filters are typically implemented in an architecture based on streaming access in specific sequential order, while volume rendering requires view-dependent traversals. In this paper, we use a generic

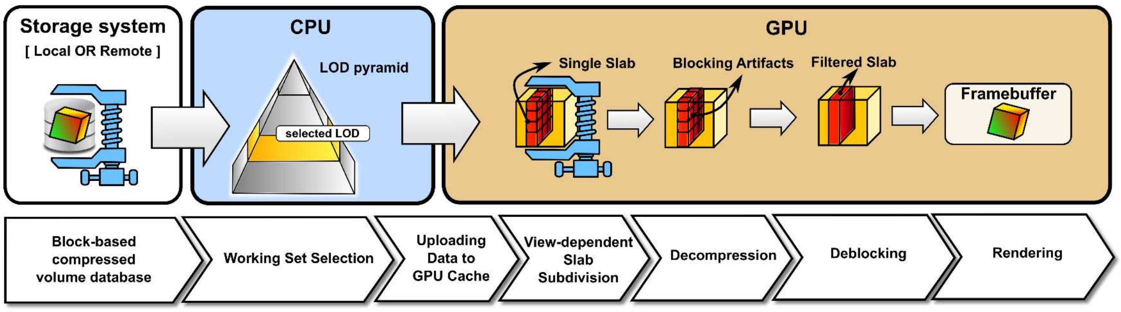

Figure 1: ***Architecture overview.*** *Our algorithm accesses an octree of compressed bricks. At run-time, an adaptive loader selects the most appropriate LOD and incrementally uploads blocks to GPU memory. At each frame, the GPU working set is subdivided into a set of slabs orthogonal to the main view direction. Each slab is decompressed, filtered, rendered, and accumulated in a front-to-back order to produce the final image.*

post-loop approach in which deblocking is integrated in an adaptive renderer traversing data in a view-dependent way. The filter employed in this paper is a low-complexity solution based on separable filtering with *rational Bézier* curves.

3. Method Overview

Our deferred filtering architecture performs at each frame decoding, filtering and rendering of compressed volume data, stored in a volume octree of coarse bricks, which are subdivided into smaller compressed blocks (see Fig. 1). Different block-based compression methods and deblocking filters can be plugged into the code without affecting the rest of the pipeline. In particular, we do not require to store any additional information in the compressed data to perform filtering. The framework supports real-time deferred deblocking and rendering on time varying data using a straightforward approach based of independent coding (or loosely dependent through dictionary sharing) of timesteps.

At each frame, we select a global desired level of detail depending on the current average projected voxel size of the volume dataset. As volume datasets (as opposed to, e.g., terrains) have limited spatial extents, and in most use cases orthogonal projections or narrow-angle perspectives are typically used in order to reduce volume distortions, it is reasonable to consider that a single volume sampling rate can be defined for rendering the entire volume. This assumption allows us to simplify the rendering pipeline, using a single resolution per frame, avoiding intra-level data filtering.

Once the level is selected, we perform adaptive refinement, incrementally uploading visible bricks to the GPU, while discarding bricks not visible because of transfer function or viewing frustum culling. Since filtering operations (for interpolation, shading, and deblocking) require access to neighboring samples to produce an output value, we adopt the deferred filtering approach of decompressing data into a temporary decoding buffer, and perform deblocking as a filtering stage before the final rendering and compositing stages. In order to manage datasets whose decoded size exceeds the available GPU memory, we subdivide the volume into thick slices, called *slabs*, orthogonal to the main viewing direction. For each slab, we perform decoding, deblocking and rendering before accumulating results in front-to-back order into the output frame buffer. The decoded and filtered buffer dimensions are thus proportional to the width of the deblocking filter. As the final deblocked data can be accessed using texture fetches and contains enough boundary elements around the currently rendered samples, the final rendering stage can perform sampling and gradient computation with trilinear interpolation to rapidly produce high quality shaded image.

A specific deblocking filter based on rational Bézier approximations is presented in Sec. 4, while an optimized GPU-accelerated implementation of the approach is presented in Sec. 5.

4. Deblocking Filter

Blocking artifacts are due to signal discontinuities at the borders of adjacent blocks. In order to reduce them, the signal close to the border must be modified without introducing new intra-block discontinuities, which would produce new artifacts in the resulting visualization. The filter must thus be block-aware and data-dependent. As we have seen in Sec. 2, in all previous methods for image and video processing, the filtering strength is higher in homogeneous regions of the image, and lower wherever high gradient variations are present. Thus, local features are preserved and blocking artifacts removed. However, in volume rendering, because of shaded semi-transparent rendering through compositing, small artifacts in uniform areas are barely perceivable, whereas artifacts at high gradient regions are clearly visible, because of gradient changes emphasized by the shading process. For this reason, we can afford a lower filtering strength in homogeneous regions and a higher, but not excessive, strength wherever high gradient variations are present near block boundaries. This section presents a new deblocking filter for volume datasets, which takes into account the fact that compression artifacts are present across block boundaries.

Rational Bézier filtering. Blocking artifacts are due to signal

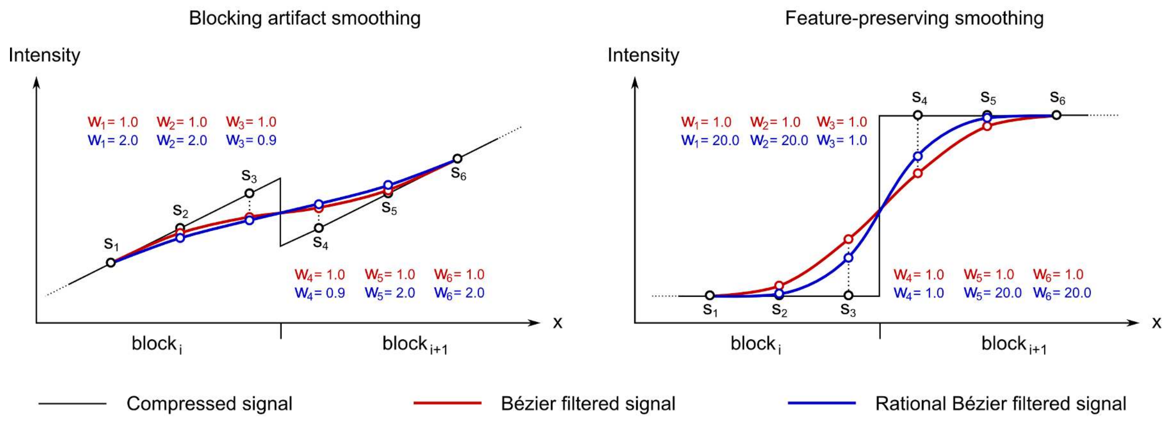

Figure 2: ***Bézier vs rational Bézier filtering.*** *Filtering the compressed signal discontinuity across the boundary of two adjacent blocks in the x axis direction. The filtered values of the samples close to the boundary are computed by projecting them onto a Bézier or a rational Bézier curve. Both approaches produce a smooth transition accross the border, but the rational Bézier one provides more accurate approximations to the uncompressed signal thanks to adding adjustable weights to the control points.*

discontinuities at the borders of adjacent blocks. In order to reduce them, the signal close to the border must be modified without introducing new intra-block discontinuities, which would produce new artifacts in the resulting visualization. To explain the proposed filter easily, let's consider a set S of aligned data samples at both sides of the border in a given axis direction (see Figure 2). Being the intensity information of the compressed dataset the signal to filter, we propose to compute the rational Bézier curve defined by the samples of S (*i.e.* control points), and project each sample onto the curve to obtain the filtered value. Thus, the new intensity of a sample $s \in S$ is computed with the following equation:

$$I_f(t_s) = \frac{\sum_{i=0}^{n} \binom{n}{i} t_s^i (1-t_s)^{n-i} I(s_i)\omega_i}{\sum_{i=0}^{n} \binom{n}{i} t_s^i (1-t_s)^{n-i}\omega_i} \qquad (1)$$

where n is the number of samples of S, $I(y)$ is a function that returns the compressed intensity of a given sample y, $t_s \in [0,1]$ is the position of the sample s within S (being $t_s = 0$ when s represents the first control point of the curve, and $t_s = 1$ when it represents the last one), and ω_i is a weight associated to the ith control point. The reason for using rational Bézier curves instead of the Bézier ones is that by adding weights, closer approximations to the original uncompressed signal can be obtained wherever a feature is detected (see Figure 2), that allows us to preserve them after filtering. To do this, a feature detection step is needed to determine which samples s_i represent a local feature. Similarly to [FRDSDF12] we compute the gradient magnitude $\|\nabla s_i\|$ at each sample position to modulate filtering strength. From these values we compute the weights of Equation 1: $\omega_i = e^{\lambda(1-\|\nabla s_i\|)}$ where λ is a user-defined parameter that controls the intensity of the filtering. In order to preserve local details in homogeneous regions of the volume, an exponential function is used to weigh more features with low gradient magnitudes. Thus, a closer approximation to the uncompressed signal is obtained, what results in a subtle filtering. On the contrary, lower weights are assigned to regions with high gradient magnitudes, which provide smoothing enough to remove the artifacts while preserving the main features. This filter mix the requirements of being block position aware and data dependent.

Deblocking process. Having the original dataset represented as a regular volumetric scalar field $V = f(x)$ where $x \in \mathbb{R}^3$, we subdivide V into a set of blocks B. Then, each block is compressed and decompressed individually. In order to apply our deblocking filter, we first perform the feature detection step, which computes the gradient magnitude for each voxel of V with central differences method. After that, the rational Bézier filter is applied sequentially along the three main axis directions of the volume. This filtering step can be summarized as follows: given two adjacent blocks b_i and b_{i+1} of B in a determined axis direction, the transition across their boundary is filtered by computing all the rational Bézier curves that traverse it, where each curve is defined by all the axis-aligned voxels of b_i and b_{i+1} at a certain point of the boundary (note that we propose a 1D filter applied over a set of axis-aligned samples. Therefore, it must be performed for every row of voxels in the x direction, for every column in the y direction, etc.). Then, since blocking artifacts appear at block boundaries, the voxels closer to the border (half the voxels of each block) are modified according to the computed curves. Thus, discontinuities between b_i and b_{i+1} are effectively smoothed. The final intensity $I_f(t_v)$ of a given voxel v, is computed as the average of its three filtered values (one for each axis).

5. GPU-accelerated implementation

Our GPU-accelerated rendering architecture generalizes current adaptive multiresolution renderers by integrating deblocking through a deferred filtering approach (See Fig. 1).

The rendering process combines adaptive CPU data loaders with a slab based renderer fully handling decompression, deblocking, and rendering on the GPU.

Working-set selection closely follows the approach of current state-of-the-art adaptive GPU volume renderers [GMI08, CNLE09, GIM12], with the important difference that level-of-detail selection is performed using a world-space threshold determined on the basis of the current view. This level is determined by projecting to the screen the voxel size from the point of the bounding sphere closest to the camera. All the active bricks are maintained on GPU in a LRU-cache that stores them in compressed form. At the end of refinement, the finest level of all the bricks determines the level of resolution at which rendering will be performed.

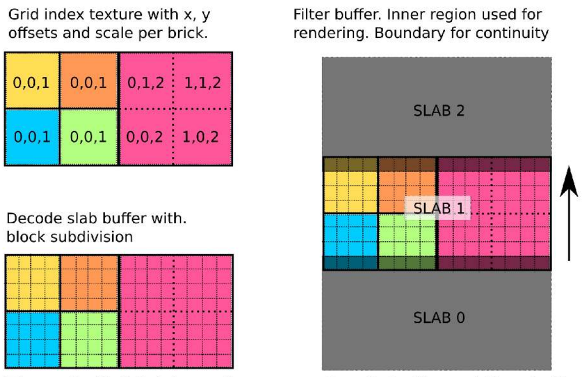
Grid index texture with x, y offsets and scale per brick.
0,0,1
0,0,1
0,1,2
1,1,2
0,0,1
0,0,1
0,0,2
1,0,2
Decode slab buffer with. block subdivision
Filter buffer. Inner region used for rendering. Boundary for continuity
SLAB 2
SLAB 1
SLAB 0

Figure 3: ***Slab bricks.*** *Index texture, decode buffer and filter buffer. Each color identifies a different brick. Pink brick is one level coarser. Numbers represent x,y offset and scale to map data into the proper output position. Only highlighted data is used for rendering, while the two boundary block slices are used for filtering and for continuity among adjacent slabs.*

Deferred filtering approach. At rendering time, data is traversed by subdividing the current working set into axis-aligned thick slabs of compressed blocks, treatable within the available GPU resources. For deblocking, each slab needs two memory buffers to store the decoded blocks and their filtered versions. Having the buffer of filtered data in a 3D-texture allows us to exploit hardware trilinear interpolation and to compute high quality shading effects. Rendering proceeds front-to-back on a slab-by-slab basis. For each slab, the active bricks are decoded, using an index texture created at each frame by the adaptive loader and containing for each non-empty brick the 3D index of the data position inside the compressed brick cache. This texture is used to identify which data need to be decoded and also for empty space skipping during rendering. Decompression from the three main axis have to deal with different transformations to write data into the proper position of the decode buffer, thus there are three different kernels to perform decoding. To smoothly change among adjacent LODs the decoding process handles a transient situation with bricks at different levels of detail, expanding all the data at the rendering resolution using data duplication when needed. A texture with same dimension of the index texture contains offset and scale information which is used to properly identify which data has to be decompressed in a particular output position, see Fig. 3 top-left. After decoding, a flat representation of the current volume slab is available for filtering. The deblocking filter needs, for each block, to access neighbors in the 3 directions. At the same time, rendering needs neighboring information also for trilinear filtering and gradient computation. For this reason, decoding, deblocking, and rendering are interleaved in a pipeline fashion, ensuring that the deblocking operation has access to one slice of fully decoded neighboring blocks and rendering has access to two slices of fully deblocked voxels along the viewing direction, see Fig. 3. Thanks to this deferred filtering approach, rendering can be performed using an accelerated raycasting traversal that uses the index texture for empty space skipping and hardware filtering for accessing deblocked data.

Bézier filter implementation. The filter is implemented with two GPU kernels. The first one computes and stores the ω_i weights in a texture. The second one performs filtering, making use of shared memory to minimize the number of fetches, with each thread writing one output voxel. The GPU blocks executing this kernel have the same size of the volume compressed blocks (b_i). Each GPU block is centered on the junction of 8 adjacent compressed blocks, and computes the filtered values around this position. To filter along X each thread fetches two values (and two weights) one from b_i and one from b_{i+1}, store them into shared memory and use them to perform X filtering. The same process is repeated along Y and Z. To speed up this process, the voxel values shared by the three filtering steps are fetched only once. The results of the three filtering steps are incrementally averaged to produce the filtered intensity $I_f(t_v)$.

6. Results

An experimental software library has been implemented on Linux using C++, OpenGL and NVIDIA CUDA 5.5. The out-of-core octree structure is implemented on top of Berkeley DB. The choice of the compression technique is orthogonal to the presented deferred filtering architecture. In this paper we decided to test two compression approaches to show its general validity: the de-facto standard Hierarchical Vector Quantization (HVQ) method [SW03] and a recent real-time decoding technique [GIM12] achieving state-of-the-art results in terms of compression quality.

We have tested our system with a variety of high resolution models and settings. In this paper, we discuss the results obtained on three datasets: a micro-CT scan of a Veiled Chameleon specimen ($1024 \times 1024 \times 1080$, 16bit/sample: 2.1GB; courtesy of Digital Morphology Project, the CTLab and the University of Texas), a 60 time steps time-varying Supernova simulation ($432^3 \times 60$, float: 18GB; courtesy of Dr. John Blondin at North Carolina State University through

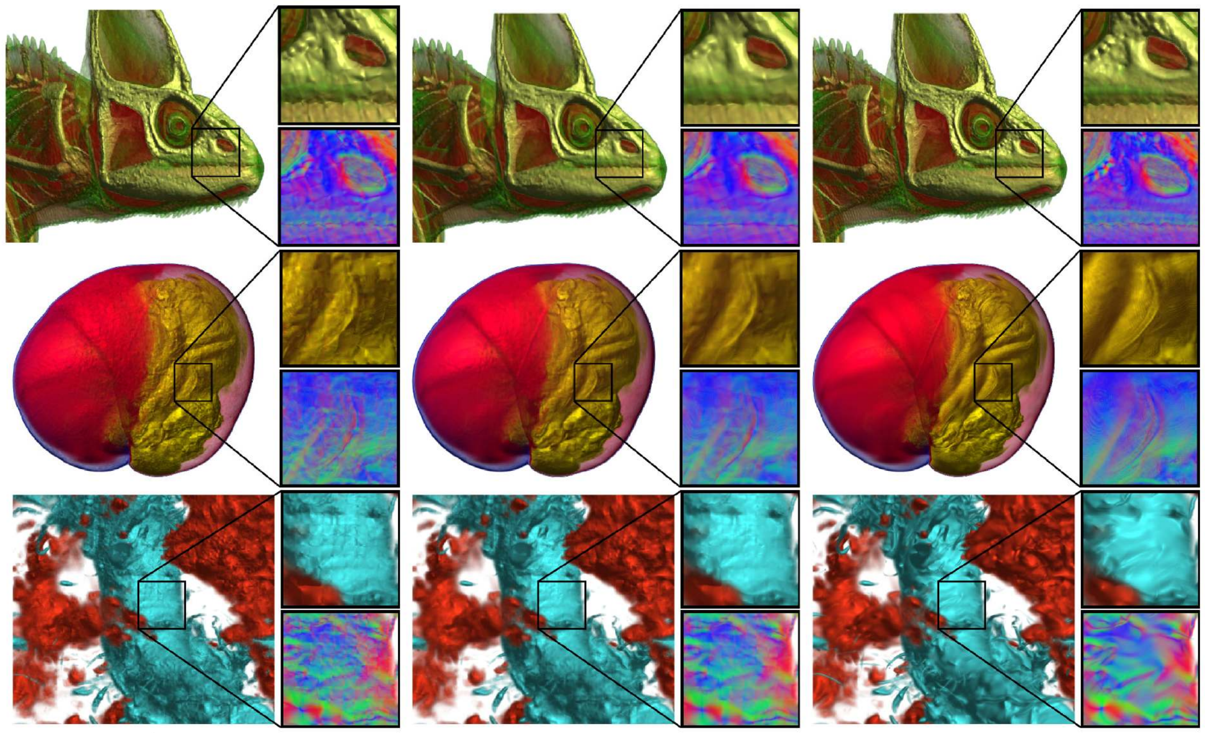

Figure 4: ***Deblocking quality.*** *Visual comparison of deblocking results for the three presented datasets. Compression using sparse coding with block size $B = 8$, dictionary size $D = 1024$, and sparsity $K = 6$ (Chameleon, Supernova) and $K = 4$ (Turbulence). From left to right: compressed without deblocking, compressed with deblocking and original datasets. On colored inset images it is possible to appreciate the compression artifacts at block boundaries and how they are reduced in the deblocked version. Artifacts are emphasized in inset image with gradient coded as color. Inset images have been used to compute SSIM and MS-SSIM values.*

SciDAC Institute for Ultrascale Visualization) and a 512 time steps time-varying Turbulence simulation (512^4, float: 256 GB, central crop of the even time steps of a 1024^4 dataset; courtesy of the Johns Hopkins Turbulence Database initiative [LPW*08]). All the tests have been performed on an Intel 3.5 GHz Core I7 PC with a NVIDIA GTX 780 with 3GB of video memory. The decompressed size of two of the three datasets exceeds the available GPU RAM, which shows the scalability of the proposed approach.

We tested our approach using strong compression rates (under 1-bit/voxel). This allowed us to test the use case of real-time exploration of time-varying datasets during animation playback with pre-loaded compressed data in GPU memory. The Chameleon and Supernova datasets have been compressed using a dictionary size of $D = 4096$ for HVQ, and a dictionary size of $D = 1024$ with sparsity $K = 6$ for sparse coding. The larger Turbulence dataset was, instead, compressed with $D = 2048$ for HVQ and $D = 1024$ with sparsity $K = 4$ for sparse coding. In all cases, block size was set to $B = 8$ voxels. The value of the deblocking filtering parameter λ has been identified through a parameter space search to get a reasonable trade-off among artifacts filtering and features preservation. For all the tests presented in this paper we have found $\lambda = 3$ to be a good compromise value.

Method	**Metric**	**Cham** 2.1 GB $1024^2 \times 1080$	**Super** 18 GB $432^3 \times 60$	**Turb** 256 GB 512^4
K-SVD	bps	0.25	0.07	0.20
	SSIM com.	0.67	0.69	0.77
	SSIM com.+deb.	0.72	0.76	0.84
	MS-SSIM com.	0.81	0.75	0.76
	MS-SSIM com.+deb.	0.83	0.80	0.84
HVQ	bps	0.63	0.17	0.71
	SSIM com.	0.69	0.67	0.82
	SSIM com.+deb.	0.78	0.76	0.89
	MS-SSIM com.	0.85	0.72	0.86
	MS-SSIM com.+deb.	0.89	0.78	0.90

Table 1: ***Datasets and image comparison results.*** *Dataset resolutions, bits-per-sample (bps) and perceptual metric values for two compression methods, HVQ and K-SVD. SSIM and MS-SSIM values are related to inset images presented in Fig. 4 (sparse coding) and Fig. 5 (HVQ) and are presented for the compressed version without deblocking (com.) and with deblocking (com.+deb.)*

Interactive performance.. All rendering tests have been performed on a 800×450 pixels viewport with a screen tolerance of 1 pixel. Frame rates are generally above 10 fps, ranging between 6 fps (Turbulence whole view) and 20 fps (Supernova). Single frame working set size ranges from 12-MVoxels (Supernova closeup), up to 128-MVoxels (Turbulence whole view). Peak throughput performance is achieved for the Turbulence dataset with 768-MVoxels/sec. Filtering quality and rendering performance are also illustrated in an

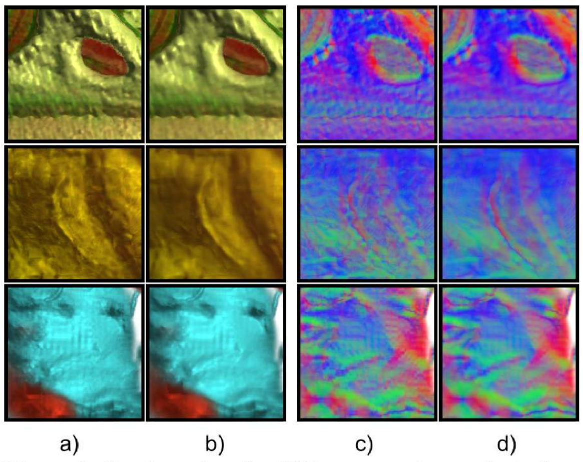

Figure 5: ***Results using the HVQ compression method.*** *Comparison between images of compressed, (a) and (c), and compressed+deblocked versions, (b) and (d), of the three datasets. Dictionary size for HVQ was $D = 4096$ for the first two datasets, $D = 2048$ for Turbulence. Artifacts are emphasized in images in rows (c) and (d) with gradient coded as color.*

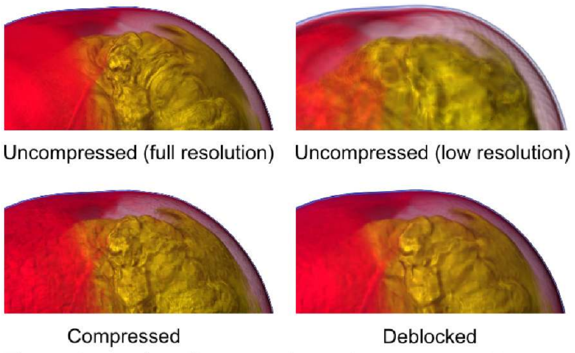

Figure 6: ***Visual quality comparison.*** *The uncompressed lower resolution has the same occupancy as the compressed version. Our deblocked version significantly reduce blocking artifacts while preserving main important features.*

accompanying video. With the proposed system it is possible to interactively inspect massive volumes and perform transfer function changes in real-time, even for time-varying models.

Deblocking filter quality assessment. For evaluating our work we show a visual comparison of the three datasets discussed in the paper (see Fig. 4). To better convey the visual impact of blocking artifacts, we emphasize the effects of compression by showing the gradient quality. Using our filter significantly reduce existing visual artifacts caused by block-based compression methods. We also performed a quantitative evaluation of our results by running the Structural Similarity (SSIM) [WBSS04] and the Multi-Scale SSIM (MS-SSIM) [WSB03] perceptual metrics, which are better suited than traditional signal fidelity measures like SNR and PSNR to evaluate blockiness. Existing calibration reports (DB-LIVE [SSB06], DB-TID2008 [PLE*08]) support our choice to use both metrics for image quality assessment when blocking artifacts are present. Because of SSIM is a grayscale metric, we compute the SSIM of each color channel of our images and use the geometric mean as an overall distortion measure, as it is done by Rajashekar et al. in [RWS09]. The SSIM quality index values obtained for all volume rendered image insets of Fig. 4 and Fig. 5 are always higher for images from deblocked volumes than those from the just compressed versions. With SSIM we got $[5-7\%]$ of gain for the K-SVD method and of $[6-11\%]$ for HVQ. By using the MS-SSIM, the gain is of $[2-8\%]$ for K-SVD and of $[4-6\%]$ for HVQ. We obtained that deblocked volumes exhibit better perceptual image qualities. All values are reported in Table 1. The effectiveness of our deblocking filter coupled with standard and state-of-the-art block-based compression methods is also confirmed through an image-based comparison that show images of the Supernova dataset in its uncompressed version, a downsampled lower-resolution version, a decompressed version without deblocking and the corresponding deblocked version (Fig. 6). The lower resolution version was computed averaging voxel values and has the same memory footprint as the compressed and deblocked ones. Being Q the image quality, we can observe in Fig. 6 that $Q_{lowres} < Q_{compressed} < Q_{deblocked} < Q_{fullres}$.

7. Conclusions and Future Work

We have presented the first GPU volume rendering architecture working on compressed data that improves visual quality by reducing blocking artifacts at rendering time. The method supports high quality shaded rendering from general block-compressed data formats, extending deferred filtering to support adaptive multi-resolution data loading, levels of detail and visibility culling. Our decompress-filter-and-render approach supports different compression methods and deblocking filters. We proposed a novel deblocking filter based on rational Bézier approximations. The method works without a-priori knowledge of the employed compression technique and does not require to modify data encoding. We have shown how massive static and dynamic datasets, including a 512^4 time varying simulation, can be decoded, filtered and rendered in real-time on commodity graphics platforms. Results show improved visual quality measured using SSIM and MS-SSIM metrics. Despite we obtain interactive frame rates, our filter is still computationally intensive, thus our future work will concentrate on improving filtering speed.

Acknowledgments. Authors would like to thank Belen Masiá for helpful discussions on perceptual metrics. This work is partially supported by the EU FP7 Program under the DIVA (290277) and GOLEM (251415) projects. We also acknowledge the contribution of Sardinian Regional Authorities.

References

[ADF04] ALTER F., DURAND S., FROMENT J.: Deblocking DCT-based compressed images with weighted total variation. In *Proc.*

ICASSP (2004), pp. iii–221–4. 2

[ASD05] Averbuch, Schclar, Donoho: Deblocking of block-transform compressed images using weighted sums of symmetrically aligned pixels. *IEEE TIP 14*, 2 (2005), 200–212. 2

[BHMF08] Beyer J., Hadwiger M., Möller T., Fritz L.: Smooth mixed-resolution GPU volume rendering. In *Proc. IEEE/EG Symp. on Volume and Point-Based Graphics* (2008), pp. 163–170. 2

[BHP14] Beyer J., Hadwiger M., Pfister H.: A survey of GPU-based large-scale volume visualization. *Proc. EuroVis)* (2014). 2

[BRGIG*14] Balsa Rodriguez M., Gobbetti E., Iglesias Guitián J., Makhinya M., Marton F., Pajarola R., Suter S.: State-of-the-art in compressed GPU-based direct volume rendering. *Computer Graphics Forum 33* (2014). 1, 2

[CNLE09] Crassin C., Neyret F., Lefebvre S., Eisemann E.: Gigavoxels: Ray-guided streaming for efficient and detailed voxel rendering. In *Proc. I3D* (2009), pp. 15–22. 5

[Cra] Craighead M.: Gl_nv_texture_compression_vtc. OpenGL Extension Registry. 2

[Eng11] Engel K.: CERA-TVR: A framework for interactive high-quality teravoxel volume visualization on standard PCs. In *Proc. LDAV'11* (2011), pp. 123–124. 2

[FAM*05] Fout N., Akiba H., Ma K.-L., Lefohn A., Kniss J.: High-quality rendering of compressed volume data formats. In *Proc. EG/IEEE Symp. on Visualization* (2005). 2

[FM07] Fout N., Ma K.-L.: Transform coding for hardware-accelerated volume rendering. *IEEE TVCG 13*, s6 (2007). 2

[FRDSDF12] Francisco N. C., Rodrigues N. M. M., Da Silva E. A. B., De Faria S. M. M.: A generic post-deblocking filter for block based image compression algorithms. *Image Commun. 27*, 9 (2012), 985–997. 4

[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*, 3/4 (2012), 1315–1324. 2, 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 (2008), 797–806. 2, 5

[GS04] Guthe S., Strasser W.: Advanced techniques for high quality multiresolution volume rendering. *Computers & Graphics 28* (2004), 51–58. 2

[HBJP12] Hadwiger M., Beyer J., Jeong W.-K., Pfister H.: Interactive volume exploration of petascale microscopy data streams using a visualization-driven virtual memory approach. *IEEE TVCG 18*, 12 (2012), 2285–2294. 2

[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. 2

[KCJ07] Kim J., Choi M., Jeong J.: Reduction of blocking artifacts for hdtv using offset-and-shift technique. *IEEE TCE 53*, 4 (2007), 1736–1743. 2

[KLF] Kniss J., Lefohn A., Fout N.: Deferred filtering: Rendering from difficult data formats. *GPU Gems 2*, 669–677. 2

[KVS04] Kong H.-S., Vetro A., Sun H.: Edge map guided adaptive post-filter for blocking and ringing artifacts removal. In *Proc. ISCAS'04* (2004), vol. 3, pp. iii–929. 2

[LJL*03] List P., Joch A., Lainema J., Bjontegaard G., Karczewicz M.: Adaptive deblocking filter. *IEEE TCSVT 13*, 7 (2003), 614–619. 2

[LLL07] Liu T.-M., Lee W.-P., Lee C.-Y.: An in/post-loop deblocking filter with hybrid filtering schedule. *IEEE TCSVT 17*, 7 (2007), 937–943. 2

[LLY06] Ljung P., Lundström C., Ynnerman A.: Multiresolution interblock interpolation in direct volume rendering. In *Proc. EUROVIS* (2006), pp. 259–266. 2

[LPW*08] Li Y., Perlman E., Wan M., Yang Y., Meneveau C., Burns R., Chen S., Szalay A., Eyink G.: A public turbulence database cluster and applications to study lagrangian evolution of velocity increments in turbulence. *Journal of Turbulence*, 9 (2008). 6

[LW03] Luo Y., Ward R.: Removing the blocking artifacts of block-based dct compressed images. *IEEE TIP 12*, 7 (2003), 838–842. 2

[LY04] Liew A.-C., Yan H.: Blocking artifacts suppression in block-coded images using overcomplete wavelet representation. *IEEE TCSVT 14*, 4 (2004), 450–461. 2

[MRH10] Mensmann J., Ropinski T., Hinrichs K.: A GPU-supported lossless compression scheme for rendering time-varying volume data. In *Proc. Volume Graphics* (2010), pp. 109–116. 2

[PLE*08] Ponomarenko N., Lukin V., Egiazarian K., Astola J., Carli M., Battisti F.: Color image database for evaluation of image quality metrics. In *IEEE Workshop on Multimedia Signal Processing* (2008), pp. 403–408. 7

[RWS09] Rajashekar U., Wang Z., Simoncelli E. P.: Quantifying color image distortions based on adaptive spatio-chromatic signal decompositions. In *Proc. IEEE ICIP* (2009), pp. 2213–2216. 7

[SIM*11] Suter S., Iglesias Guitián J., Marton F., Agus M., Elsener A., Zollikofer C., Gopi M., Gobbetti E., Pajarola R.: Interactive multiscale tensor reconstruction for multiresolution volume visualization. *IEEE TVCG* (2011). 2

[SSB06] Sheikh H., Sabir M., Bovik A.: A statistical evaluation of recent full reference image quality assessment algorithms. *IEEE TIP 15*, 11 (2006), 3440–3451. 7

[SW03] Schneider J., Westermann R.: Compression domain volume rendering. In *Proc. IEEE Vis.* (2003), pp. 293–300. 5

[TBR*12] Treib M., Burger K., Reichl F., Meneveau C., Szalay A., Westermann R.: Turbulence visualization at the terascale on desktop PCs. *IEEE TVCG 18*, 12 (2012), 2169–2177. 2

[VKG04] Viola I., Kanitsar A., Gröller M. E.: GPU-based frequency domain volume rendering. In *Proc. Spring Conf. on Computer graphics* (2004), pp. 55–64. 2

[WB08] Wong A., Bishop W.: Deblocking of block-transform compressed images using phase-adaptive shifted thresholding. In *Proc. ISM* (2008), pp. 97–103. 2

[WBSS04] Wang Z., Bovik A., Sheikh H., Simoncelli E.: Image quality assessment: from error visibility to structural similarity. *IEEE TIP 13*, 4 (2004), 600 –612. 7

[WSB03] Wang Z., Simoncelli E., Bovik A.: Multiscale structural similarity for image quality assessment. In *Proc. Asilomar Conf. on Signals, Systems and Computers* (2003), vol. 2, pp. 1398–1402. 7

[WSKW05] Wetekam G., Staneker D., Kanus U., Wand M.: A hardware architecture for multi-resolution volume rendering. In *Proc. Graphics hardware* (2005), pp. 45–51. 2

[WWH*00] Weiler M., Westermann R., Hansen C., Zimmermann K., Ertl T.: Level-of-detail volume rendering via 3d textures. In *Proc. IEEE Volume Visualization* (2000), pp. 7–13. 2

[WYM10] Wang C., Yu H., Ma K.-L.: Application-driven compression for visualizing large-scale time-varying data. *IEEE CGA 30*, 1 (2010), 59–69. 2

[WZF04] Wang C., Zhang W.-J., Fang X.-Z.: Adaptive reduction of blocking artifacts in dct domain for highly compressed images. *IEEE TCE 50*, 2 (2004), 647–654. 2

[YNV08] Yela H., Navazo I., Vazquez P.: S3dc: A 3dc-based volume compression algorithm. In *Proc. CEIG* (2008). 2