EXPLOITING SPATIO-TEMPORAL COHERENCY IN TIME-VARYING VISUALIZATION OVER NETWORK ENVIRONMENTS

Lazaro Campoalegre, Tom Noonan and John Dingliana
Graphics Vision and Visualization Group , Trinity College Dublin, Ireland

ABSTRACT

We propose an efficient client-server interaction system for multi-user visualization of real-time time-varying volumetric simulations. The approach is designed to reduce data transfer over network environments, while enabling users to interact independently with the visualization. Exploiting spatio-temporal coherence and parallel processing on Graphical Processing Units (GPU's) on both server and client, a predictive mechanism on the server evaluates, on the fly, whether blocks of data can be extrapolated with sufficient accuracy based on previous frames of simulation. If so, a simple instruction is sent to the client to extrapolate the block, in lieu of transmitting actual volume data of the block. A comparative analysis indicates that we can achieve visualization performance, for time-varying datasets, on par with previous techniques whilst considerably reducing the size of data sent through the network. Perceptual evaluations using an established metric indicate that the visual quality is not noticeably reduced.

KEYWORDS

Time-varying data, Volume Rendering

1. INTRODUCTION

Effective collaborative research is a major challenge in scientific visualization. Scientists in various fields often want to analyze, discuss and evaluate data interactively at their own locations, but real-time interaction, in particular, has always been difficult in distributed visualization environments. Remotely accessing large modern-day datasets requires provision of robust systems that guarantee acceptable image quality and low latency. Despite continual advances in computing power, the use of high performance graphics workstations is still considerably restricted due to the cost of the most modern hardware. Widespread deployment of high-speed wireless networks have increased interest in remote rendering, which offers some evident advantages. It provides the potential for rich rendering experiences to thin clients such as mobile devices with limited computing resources, shared by multiple clients and is a simple but effective solution. However the minimization of interaction latency still remains a challenge. Exploiting spatial and temporal coherence is a potential means of avoiding increasing computational cost and reducing display time. Decreasing the time required to transfer a sequence of volumes to the rendering engine is still a considerable challenge. Any such improvements must be achieved without removing fine visual details, which could be relevant to the task being supported by the visualization.

Our main contribution is a novel approach for multi-user remote rendering of time-varying volumetric data. We propose a robust, perceptually-based predictive mechanism that exploits spatiotemporal coherency to reduce the bandwidth of data transferred over the network during real-time visualization of volume data, with minimal loss of visual quality. Our scheme is buffer-based, allowing us to deal with time-varying datasets with an arbitrary number of timesteps. We exploit the parallel computing capabilities of modern Graphics Processing Units (GPUs) to support a GPU-GPU client-server framework, which allows several users to simultaneously and independently interact with visualizations of complex real-time volumetric simulations.

ISBN: 9978-989-8533-66-1 © 2017

2. PREVIOUS WORK

Remote visualization has become a topic of significant interest in recent years. Various client-server approaches have been proposed to compensate for limitations in low performance devices or to reduce costs. We discuss, here, the previous works most relevant to our solution and, for more extensive reviews, we refer the reader to the recently published surveys on interactive remote rendering systems by Shi & Hsu (2015), and compressed GPU based volume compression techniques by Balsa & Vázquez (2012).

Several commercial applications have already been developed for remote visualization of volume data. For instance, Vizserver (Ohazama, 1999), is a system that allows OpenGL programs to run in a client-server environment without modification and almost no restrictions on the visualization. However the number of users that are allowed to visualize and interact independently with a dataset simultaneously under the same graphics pipeline is limited. GLX (Kilgard, 1996) is an OpenGL extension for the X server system that employs command streaming to allow 3D graphics rendering applications to display on a remote X server.

Bethel et al. (2000) proposed a system for large-volume scientific data visualization, where a server performs the rendering and transmits resultant images to clients. Similarly, in the remote visualization framework proposed by Engel et al. (2000), a subset of volume data output from a local slicing tool is transferred to a high-end server, where it is rendered off-screen and transmitted back as compressed images to the client for display. Qi & Tyler (2005) proposed a 3D medical visualization system that progressively transmits losslessly encoded data, which is reconstructed on-the-fly in the rendering client.

Sharp et al. (2010) claim to be the first to present a system for multi-client volume rendering that leverages a server GPU environment. They achieve efficient remote rendering of medical volume data, using a transfer function-aware, GPU-based, empty space-skipping algorithm. In the COVRA system, proposed by Gobbetti et al. (2012), a block-based multiresolution compression domain is employed that allows very large volume datasets to be compressed offline and decompressed on-demand in real-time. They achieve a high compression ratio while maintaining good visual results. Bajaj et al. (2002) proposed a parallel volume rendering system, which uses the GPU of the client to edit the transfer functions (TF) and color maps, while the server comprises a multi-PC framework for rendering high quality images on demand.

A popular strategy is to exploit temporal coherence in improving the efficiency of processing time-varying datasets. For instance Shen et al. (1999) propose a data structure called the Time-Space Partitioning tree to capture spatial and temporal coherence to improve rendering speed for large-scale out-of-core rendering of time-varying fluid dynamics simulation. Gao et al. (2005) extended upon this by employing adaptive retrieval of regions based on visibility for distributed collaborative visualization. More recently Jang et al. (2012) used a functional representation of animated volumes to develop an efficient encoding technique that takes into account the temporal similarity between timesteps. Akiba et al. (2006) designed a technique that uses time histograms for simultaneous classification of time-varying data. Younesy et al. (2005) exploit temporal coherence using a Differential Time-Histogram Table that stores voxels which change between timesteps or during transfer function updates, while Fang et al. (2007) employ a Time Activity Curve to identify temporal patterns. Janicke et al. (2007) implemented a technique that detects important regions based on local statistical complexity. After applying conditional entropy, Wang et al. (2008) compute importance curves that are used to evaluate the temporal behavior of blocks. Noonan et al. (2015) exploited block-wise temporal coherence to improve streaming time-varying volume data from CPU to GPU on a single workstation.

Although interesting practical solutions have been proposed in recent years, there remain a number of open problems that merit further investigation. Transmitting the whole volume model to the client device ensures the best interaction capabilities but is still a challenging problem. Some previous works have achieved promising results but typically restrict size of models or are prone to network latency, limiting interaction capabilities. Compression is a good solution for volume models whose size exceeds current hardware capabilities but performance during decompression is an area in need of improvement. A recent trend is to move decompression to the final stages of the graphics pipeline and potential on the GPU, resulting in better exploitation of available bandwidth.

In this paper, we address some of the open problems of previous approaches to deliver a simultaneous multi-user visualization of time-varying volume data by taking advantage of the spatio-temporal coherence of consecutive timesteps in volumetric animation to reduce the network load. Our scheme involves the GPUs of both server and client in a synchronized framework where the server simulates the client's pre-rendering functions, allowing the extrapolation of local datasets on the client side, reducing the explicit transfer of data.

3. OVERVIEW

The framework discussed in this paper is extended from an approach we previously proposed (Noonan et al. 2015) for improving performance in the streaming of time-varying volume datasets from CPU to GPU on a single workstation. That is, we exploit a similar strategy of data reduction through prediction, but extend upon it to support a wholly different use-case, namely that of a networked client-server mechanism capable of facilitating simultaneous multi-user interactions with time-varying volume simulations. In particular, we support real-time interaction, which we define as the ability to process the data on-the-fly with no requirement for preprocessing the entire dataset. We provide a number of novel improvements to the core of the previous approach including added consideration for spatial coherency that results in improved visualization quality. We also exploit the GPU on both server and client, specifically we employ the NVIDIA CUDA platform[1], which provides gains in terms of scalability, performance and the allowable size of datasets that can be handled. Unlike our previous approach, which requires all simulation timesteps to be preloaded into memory, our new mechanism is buffer-based and reads, sequentially, a subset of timesteps on the fly, thus it is not limited to dealing with datasets that can wholly fit in CPU memory. Finally, we provide an added mechanism for dealing with null blocks to further improve data reduction.

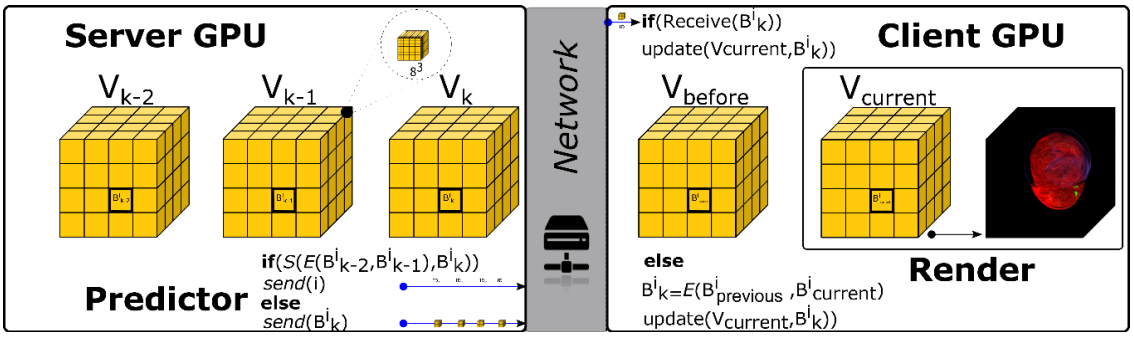

Figure 1. Overview of the proposed approach: volumes are transmitted in a block-wise manner from server to client. Data transmission is reduced when it is determined that the client can approximate, using a predictor function (E), a given block with sufficient accuracy, determined by a similarity function (S)

Figure 1 provides an overview of our framework, which comprises a synchronized mechanism that involves both server and client GPUs. Our scheme reads, from disk, subsequent timesteps, V_k, of a time-varying volume dataset composed of n timesteps, where $k \in [1, n]$. We start by sending the first two volumes corresponding to the first two timesteps V_0 and V_1 to the server-GPU, subdivided into blocks B^i of $8 \times 8 \times 8$ voxels, and then subsequent volumes are transferred, in a block-wise manner, to the client.

On the server-GPU, we employ a CUDA-based *prediction mechanism* that computes, in parallel, linearly extrapolated blocks, B^i_{new}, for each timestep k, based on the two previous timesteps. Then, a *similarity function* evaluates whether the extrapolation result is a sufficiently accurate approximation of the actual data of the timestep B^i_k. If the extrapolation result is not sufficiently similar to the actual data, a full block is transmitted to the client, which performs an update of the timestep volumes on the client-GPU's memory with the newly received data. Otherwise, an instruction is sent to the client to approximate the block B^i_{new} in lieu of transmitting the actual data. In this case, the client executes an identical extrapolation scheme to the server, before the timestep volume is updated and rendered.

In Section 4 and 5, we describe in detail the *prediction mechanism* and *similarity function* respectively. Section 6 discusses our networking framework for visualization on multiple clients.

[1] NVIDIA CUDA webpage: http://www.nvidia.com/object/cuda_home_new.html

4. GPU-BASED PREDICTIVE DATA REDUCTION

At run-time, the server is responsible for instructing the client whether to receive new volume blocks or to extrapolate blocks, B^i_{new}, where it has determined that such an extrapolation can accurately approximate the local behavior of the simulation. In this manner, we exploit computational resources to make significant savings on data transfer. As a key improvement over the approach presented in our previous work (Noonan et al. 2015), we exploit parallel computing capabilities of GPUs on both the server and client. Importantly, this enables real-time processing of large time-varying volume datasets, which in turn allows us to support a networked client-server framework facilitating remote collaborative visualization.

Assuming some degree of spatio-temporal coherence between subsequent timesteps, our approach tries to approximate the behavior of scientific simulations as linear within short periods of time. We apply a linear extrapolation to each voxel inside each block, by forcing the second derivatives (Laplacian) to be 0, as follows: $d_k = 2 \cdot d_{k-1} - d_{k-2}$, where d_{k-2}, d_{k-1} and d_k are consecutive values of the voxel d in corresponding timestep volumes V_{k-2}, $V_{k-1}$and V_k respectively.

The predictive mechanism starts at timestep $k = 2$, where blocks B^i_{new} are calculated in the server GPU by extrapolating from the two consecutive co-spatial blocks $B^i_{current}$ and $B^i_{previous}$. After this, a *similarity function, S* (detailed in the next section), evaluates whether the extrapolation result, B^i_{new}, is a sufficiently accurate approximation of the actual data of the next timestep, B^i_k. If so, the server sends a signal with the block identifier, i, to the client-GPU. Otherwise the actual block B^i_k is transmitted to the client. Then we update the two consecutive volume timesteps held in the server's GPU memory.

The manner in which we manage data transmission between server and client is a key ingredient of the networked visualization approach. On the client, we compute $B^i_{new} = 2 \cdot B^i_{current} - B^i_{previous}$, at each step following initial setup and update $B^i_{previous} \leftarrow B^i_{current}$. Then we update $B^i_{current} \leftarrow B^i_{new}$ if a block identifier (i) is received from the server, or $B^i_{current} \leftarrow B^i_k$ if a block (B^i_k) is received. Performing the extrapolation on a block-wise basis allows us to reduce storage on the client-GPU to just two full frame volumes, which allows us to support a buffer based mechanism that works regardless of the number of timesteps in the dataset.

We further improve performance by avoiding extrapolation of null regions in the volume datasets by evaluating whether all the voxels in B^i_k are empty. If so, the algorithm proceeds by assigning a *null* identifier, transmitted using only 1 byte, to avoid the redundant transmission or extrapolation of the block on the client GPU. When a block with a null identifier is received by the client, the algorithm examines the $null$ identifier of the preceding co-spatial block. If this is also $null$, no update of $B^i_{current}$ is performed. Otherwise we assign 0 to each voxel in $B^i_{current}$, thus substituting the extrapolation function by a more efficient assignment function.

5. SPATIO-TEMPORAL SIMILARITY FUNCTION

In previous work (Noonan et al. 2015), we proposed a *similarity function* that evaluates temporal coherency of blocks, determining whether they should be extrapolated or transmitted from server to client. This is done by computing the Root Averaged Squared difference between consecutive co-spatial voxels and comparing it to an empirical threshold value, ε. Whilst the temporal coherency component accounts for how a voxel changes from one frame to the next, if the value in the voxel moves within the block, the overall block may not be significantly changed. Thus, we improve upon this by additionally accounting for spatial similarity within blocks by using a weighting factor, W^i_k (denoted for convenience as W), in the *similarity function*, S (see Equations (1) and (2)).

$$S = \begin{cases} true & if \sqrt{\frac{1}{m} \cdot \sum_{v=0}^{m} W \cdot (x_v - x'_v)^2} < \varepsilon \\ false & otherwise \end{cases} \quad \text{(Erro! Marcador não definido.)}$$

$$W = b \cdot Max[(x_v - x_u)^2] + 1 \quad (2)$$

In these equations, x_v and x'_v are co-spatial voxels corresponding to the two consecutive timesteps; m is the block size in voxels (typically $8 \times 8 \times 8$); and $x_u$is one of six neighbors of the voxel x_v in the same timestep.

The parameter b, which we call the *spatial parameter*, assigns a weight that captures the influence of the spatial distribution of voxels at each block during the simulation. The ε variable is analogous to a compression factor and adjusting it leads to varying degrees of accuracy in the final result. In practice, the value can be subjectively chosen for a given dataset or use-case but the ideal value of ε varies across different datasets and transfer functions. Our CUDA implementation on both server and client allows us to evaluate both temporal and spatial behavior of blocks without decreasing the performance in the simulation. A study of the influence of W on the visual result and the extrapolation behavior is presented in Section 7.

6. DATA TRANSMISSION FOR NETWORKED MULTI-USER VISUALIZATION

As previously discussed, our objective is to facilitate a networked multi-user system where the server transmits, to the client, only non-null blocks that cannot be extrapolated in the client-GPU. These are sent as part of packets that are classified as either *initialization* packages, *empty* or packages containing only *data*. An initialization packet contains the required information for setting up a 3D texture as well as managing the volume animation on the client side, that is, the resolution, number of timesteps, timestep identifier and bytes per voxel. Packets that contain data include the block ID and the voxel information. An empty packet serves as an identifier for null blocks.

On the client, a receiving function collects the information to generate the blocks of data within a 3D texture corresponding to the current timestep of the volumetric simulation. The block update within the current texture is performed directly within the client-GPU memory, each time a new block is fully received or extrapolated on the client side.

We use TCP (Transmission Control Protocol), which offers error detection ensuring the correct delivery of our blocks of data in the transmitted packages. TCP also includes a flow control that determines when data needs to be re-sent, and stops the flow of data until previous packets are successfully transferred. The result is a robust client-server scheme that guarantees interactive rendering in both server and multiple clients and allows unrestricted, independent simultaneous interaction of a large number of users with large scale datasets.

7. RESULTS AND DISCUSSION

For generality of results, we tested our approach using different types of data, namely:

- A *Supernova* simulation dataset made available by Dr. John Blondin at the North Carolina State University through US Department of Energy's SciDAC Institute for Ultrascale Visualization, Figure 2 (left), with a resolution of $432 \times 432 \times 432$ voxels and 60 times steps.
- A *Smoke Simulation*, Fig. 2 (middle), which is simulated based on the method presented by Fedkiw et al. (2001) with a resolution of $100 \times 100 \times 100$ voxels and 500 timesteps.
- A *4D MRI lung* dataset, Fig. 2 (right) (von Siebenthal, 2008) that contains dynamic sagittal image slices of free breathing. The model is composed of 200 timesteps with $50 \times 224 \times 224$ resolution.

All images in this section were rendered at a resolution of 817×534, using our own implementation of a GPU volume ray-caster based on the approach by Hadwiger et al. (2006).

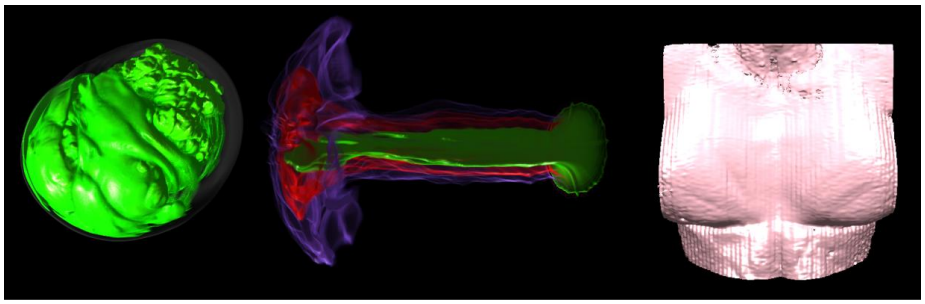

Figure 2. Volumetric datasets used in our experiments

Multi-user interaction: Figure 3 shows images from simultaneous multi-user interactions with the *Super Nova*, where users on four remote clients independently view synchronized data streamed from a server, running the simulation. Each client employs a different transfer function that can be manipulated independently along with other viewing parameters, such as rotation, zooming. The transfer functions represent different density ranges and were chosen to study the extrapolation behavior across a variety of functions.

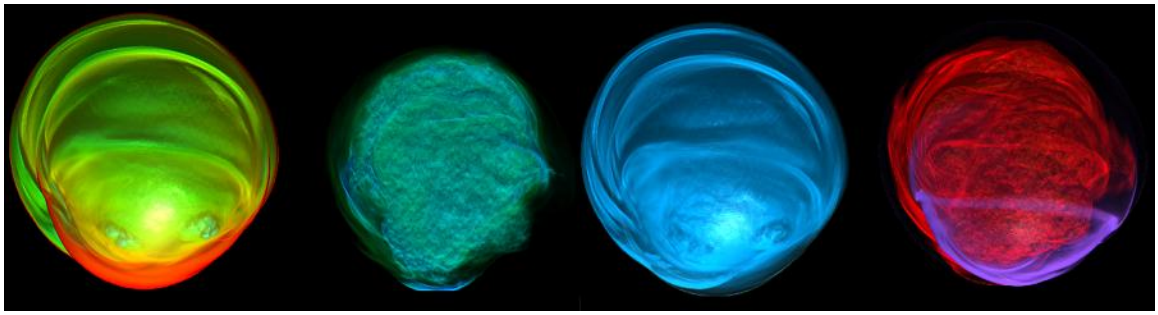

Figure 3. Multi-user visualizations of the Supernova using different transfer functions

Reduction in Data Transmission: We characterize the performance of our scheme by the extrapolation ratio, E_r, where $0 \leq E_r \leq 1$. This is defined, for any particular simulation, as the ratio of the total number of extrapolated volume blocks across all the simulation frames to the total number of rendered blocks.

In general, our approach achieves similar results to those reported by Noonan et al. (2015), which is generally around 50% reduction of data transmission for typical viewing conditions, when compared to the full volume data. However, since, many volume datasets consist of a significant subset of empty blocks (null regions) which might simply be skipped, we report, E_r as the reduction achieved purely in the non-null regions.

The graph in Figure 4(a) shows a study of the visual quality and extrapolation behavior of the three simulations mentioned previously across discrete values of the *spatial parameter*, b. Quality scores, Q, are calculated using the High Dynamic Range Visual Difference Predictor (HDR-VDP2) (Mantiuk et al. 2011), which is an established metric for perceptual similarity.

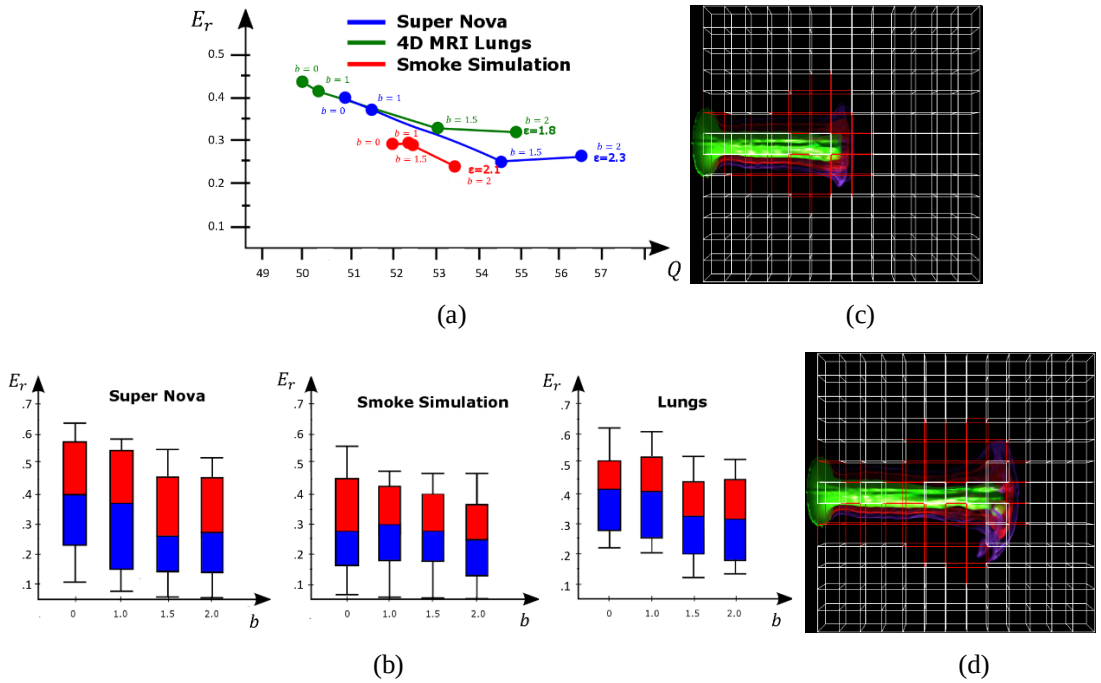

Figure 4. (a) Study of visual quality and extrapolation ratio (Er) for test datasets using different values of the spatial parameter b; (b) Study of achieved by our technique for the three datasets (c) Cross-section of the smoke simulation at timesteps 76; blocks marked in white are either extrapolated or empty (d) similar visualization of timestep 122

As previously mentioned, the ideal value of ε varies across different datasets. In order to assure acceptable visual quality, we empirically chose an ε value for each simulation, so that the visual quality of 99% of the timesteps of the simulation were at least $Q = 50$, which was taken as an indicator of good visual similarity. Note that for each simulation there is a significant influence of the parameter, b, on the visualization, that is, when other parameters are constant, incrementing b leads to considerable improvement in visual quality, Q, across the simulation.

The quartiles in Figure 4(b) show the mean E_r (in non-null regions) across all frames of the Super Nova, Smoke Simulation and MRI Lungs for four empirically tested values of the *spatial parameter*, b. Although the mean E_r values generally fall under 0.50, the extent of the upper half of the box plots indicate that a considerable amount of blocks are extrapolated in a high percentage of the timesteps at each studied simulation. The ε used here are as in Figure 4(a). Note also that there is generally a decrease of E_r when increasing b.

The images in Figure 4(c) and (d) show cross-sections of the rendering of the smoke simulation to distinguish the $8 \times 8 \times 8$ blocks extrapolated in timesteps 76 and 122 respectively. Blocks outlined in red are transmitted to the client, while the white blocks are either extrapolated or recorded as null blocks of the volume. Note that a considerable amount of blocks in the non-*null* region are also extrapolated.

Figures 5 and 6 show the rendering of the Smoke Simulation and Supernova respectively, in order to visually demonstrate the influence of the *spatial parameter*, b, on the visual quality of the images as described in Figure 4(a). In both cases, we selected a fixed ε value whilst increasing the parameter b. Note that the visual quality Q is considerably better in the cases where $b = 2.0$ (see figures 5(left) and 6(left)).

Table 1 shows how average frame processing times are affected by the spatial parameter, b, when applied to a number of time-varying datasets. Timings include network transfer time over local wifi and were captured on a server equipped with 16GB RAM and an Nvidia GeForce GT 750M with 2GB RAM, and a single local client with identical specifications. Note that there exists a clear increase of the frame processing time when b is increased.

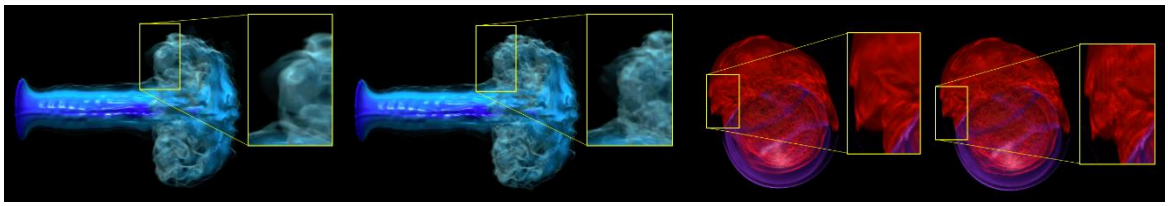

Figure 5. Influence of the spatial parameter, b, on image quality for the smoke simulation: b=2.0, Q=55.62 (left); and b=0.5, Q=50.20 (right). In both cases ε =1.5

Figure 6. Influence of b for Supernova: b=2.0, Q=58.38 (left); and b=0.5, Q=53.18 (right). In both cases ε =1.8

Comparative Analysis: Table 2 presents a feature comparison between our technique and popular approaches in the literature for network-based visualization of volume datasets. The rows of the table indicate, respectively, the size of datasets used in the comparison; suitability for multi-user visualization; whether the approach is real-time, in other words whether it can visualize data on-the-fly without pre-processing the full frame sequence; support for time varying volume data; and whether the volume data resides on the client side. In addition, performance indicators are provided in terms of average *frames per seconds* (FPS) and the effective compression ratio (equivalent of E_r).

Table 1. Influence of b on performance

b	Rendering Time (ms)		
	Smoke Simulation	Supernova	4DMRI Lungs
0	19.5	24	20.14
0.5	23.21	27.36	24.75
1	25.18	34.73	30.46
1.5	28.11	37.19	31.8
2	34.5	39.08	35.22

Table 2. Feature comparison with other published techniques

Approach	Gobetti et. al. (2012)	Sharp et. al. (2010)	Bajaj et. al. (2002)	This Paper
Dataset	Supernova (432^3)	512^3	512^3	Supernova (432^3)
Multi-user	×	✓	×	✓
Real-time	×	✓	✓	✓
Time-variant data	✓	×	×	✓
Volume-on-client	✓	×	×	✓
FPS	10	15	10-12	6-11
Compression ratio	0.98	-	0.9977	0.5

Note that Gobetti et al. (2012) achieve interactive frame rates for very large datasets, is suitable for volume animations and considerably reduces traffic over the network. Their limitations lie in the time needed for pre-processing and lack of support for simultaneous users independently interacting with the visualization. Support for a wide range of users interacting in real-time is provided by Sharp et al. (2010) with a transfer-function aware medical image application. However, since the rendering is performed on the server, their approach requires a very complex server-side mechanism composed of multiple GPUs for simultaneously rendering images on-demand from several different users. The approach also involves the CPU in the task of managing the transfer functions received from each user simultaneously. Thus the reduction in data transmission is dependent on the selected transfer function at each client. The thin-client proposed by Bajaj et al. (2002) achieves interactive exploration of volume data, but is not suitable for multi-user systems and has not been designed for time-varying volume data. The compression ratio in the related column in Table 2 is approximated based on the ratio of the transmitted image size versus the volume data and the FPS (frames per second) figures for our approach are based on a simulation with 8 simultaneous users interacting independently with the supernova visualization over the network.

Our approach overcomes a number of limitations of these previous techniques. We guarantee simultaneous interactive visualization of large scale datasets, while considerably reducing the data sent over the network. Although we do not achieve ratios as high as Gobetti et al. (2012), we have the advantage of not requiring a pre-processing stage. In the case of the supernova dataset ($432^3 \times 60$, float 18 GB), the average size of a frame transmitted to the client GPU was $\sim$158 MB.

As stated, our system mainly extends upon the technique proposed by Noonan et al. (2015) but in contrast to this previous approach, we exploit GPUs on the client, which facilitates the system to run in real-time for time-varying data of arbitrary timesteps; in other words our buffer-based approach is not limited to data that fits into GPU memory. In fact our approach is well suited for in-situ visualization with multiple clients, where a stream of volume data may be rendered on the fly as it is simulated or acquired.

8. CONCLUSIONS

We presented a novel framework for time-varying-volume data visualization designed for bandwidth-constrained network environments. In our block-wise scheme, a server-GPU simulates client-GPU pre-rendering functions characterizing the local behavior of the volume dataset, exploiting this to avoid transmission of complete per-frame volumes to the client-GPU. Results indicate that we are able to considerably reduce the transfer bandwidth, exploiting both temporal and spatial coherency.

The technique requires no pre-processing and allows multiple users, over the network, to have simultaneous, personalized independent interactions with the simulation data. Analysis of the output using an established perceptual metric suggest that the visual quality is not noticeably reduced. The resulting frame rates provide evidence of the viability of our technique compared to popular approaches described in the literature. In our framework, the 3D data reside in both server and client GPUs, allowing each client to independently change viewing parameters as well as the transfer function.

Due to the linear extrapolation component, the technique is ideally suited for large scale models where a first order approximation can be made of the dynamic behavior of a significant proportion of the dataset. The proposed method assumes that multiple users will access the data in a sequential manner, in other words, it does not particularly aid volume data visualization if users want to look at random frames, and the server side cost will increase if users access different timesteps simultaneously. However, the system would be well suited for instance for in-situ visualization where the data is being viewed *on-the-fly* as it is generated by a simulation or acquired using some sensor technology. We also assume that the rendering client will be equipped with a GPU, however mobile devices are becoming increasingly equipped with graphical computing capabilities, and although our current implementation is implemented in CUDA, we believe it can be ported to other parallel computing platforms such as OpenCL.

The results presented in the paper are based on a block size of 8^3 voxels. We also tested the system with 16^3 blocks but found negligible difference in performance as the gains seemed to be counter-balanced by overhead costs of processing larger blocks. In future work we plan to investigate the use of adapted block sizes corresponding to the temporal volume behavior. We plan to replace the Linear Extrapolation function with a more complex scheme based on the use of localized filters as predictive functions. Using our approach alone, the size of individual frame of volume data needs to be within the maximal size allowed by the available GPU RAM but we believe our solution could be coupled with other strategies for downsampling or adaptively accessing only parts of the volume that are visible within the viewport at a particular time. We plan to integrate and test our system with such approaches in the future. We also plan to conduct perceptual evaluations based on user tests to further study the quality of the results. With increasing adoption of mobile technologies, we would like to test the effectiveness of our technique on such devices and adapt our algorithms to run optimally on clients with reduced hardware capabilities.

ACKNOWLEDGEMENT

This research has been conducted with the financial support of Science Foundation Ireland (SFI) under Grant Number 13/IA/1895.

REFERENCES

Akiba, H., Fout, N., & Ma, K.-L. (2006). Simultaneous Classification of Time-varying Volume Data Based on the Time Histogram. *Proceedings of the Eighth Joint Eurographics / IEEE VGTC Conference on Visualization* (pp. 171-178). Eurographics Association.

Bajaj, C., Park, S., & Thane, A. (2002). Parallel multi-PC volume rendering system. *CS \& ICES Technical Report, University of Texas at Austin, 2*.

Balsa, M., & Vázquez, P. (2012). Practical Volume Rendering in Mobile Devices. In *Advances in Visual Computing* (pp. 708-718). Springer.

Bethel, W., Tierney, B., Lee, J., Gunter, D., & Lau, S. (2000). Using high-speed WANs and network data caches to enable remote and distributed visualization. *Supercomputing, ACM/IEEE 2000 Conference*, (pp. 28-28).

Engel, K., Ertl, T., Hastreiter, P., Tomandl, B., & Eberhardt, K. (2000). Combining local and remote visualization techniques for interactive volume rendering in medical applications. *Proceedings of the conference on Visualization'00*, (pp. 449-452).

Fang, Z., Möller, T., Hamarneh, G., & Celler, A. (2007). Visualization and Exploration of Time-varying Medical Image Data Sets. *Proceedings of Graphics Interface 2007* (pp. 281-288). ACM.

Fedkiw, R., Stam, J., & Jensen, H. W. (2001). Visual simulation of smoke. *Proceedings of the 28th annual conference on Computer graphics and interactive techniques*, (pp. 15-22).

Gao, J., Huang, J., Johnson, C. R., & Atchley, S. (2005, Oct). Distributed data management for large volume visualization. *VIS 05. IEEE Visualization, 2005.*, (pp. 183-189). doi:10.1109/VISUAL.2005.1532794

Gobbetti, E., Guitián, I. a., & Marton, F. (2012). COVRA: A compression-domain output-sensitive volume rendering architecture based on a sparse representation of voxel blocks. *Computer Graphics Forum, 31*, pp. 1315-1324.

Hadwiger, M., Kniss, J. M., Rezk-salama, C., Weiskopf, D., & Engel, K. (2006). *Real-time Volume Graphics.* Natick, MA, USA: A. K. Peters, Ltd.

Jang, Y., Ebert, D. S., & Gaither, K. (2012). Time-varying data visualization using functional representations. *Visualization and Computer Graphics, IEEE Transactions on, 18*, 421-433.

Janicke, H., Wiebel, A., Scheuermann, G., & Kollmann, W. (2007). Multifield Visualization Using Local Statistical Complexity. *IEEE Transactions onVisualization and Computer Graphics, 13*, 1384-1391.

Kilgard, M. J. (1996). *OpenGL programming for the X Window System.* Addison Wesley Longman Publishing Co., Inc.

Mantiuk, R., Kim, K. J., Rempel, A. G., & Heidrich, W. (2011). HDR-VDP-2: A Calibrated Visual Metric for Visibility and Quality Predictions in All Luminance Conditions. *ACM Trans. Graph., 30*, 40:1--40:14.

Noonan, T., Campoalegre, L., & Dingliana, J. (2015). Temporal Coherence Predictor for Time Varying Volume Data Based on Perceptual Functions. In D. Bommes, T. Ritschel, & T. Schultz (Ed.), *Vision, Modeling \& Visualization.* The Eurographics Association.

Ohazama, C. (1999). OpenGL Vizserver. *White Paper, Silicon Graphics Inc*.

Qi, X., & Tyler, J. M. (2005). A progressive transmission capable diagnostically lossless compression scheme for 3D medical image sets. *Information Sciences, 175*, 217-243.

Sharp, T., Robertson, D., & Criminisi, A. (2010). *Volume Rendering on Server GPUs for Enterprise-Scale Medical Applications.* Tech. rep. 72, Microsoft Research, Cambridge, UK, Microsoft Research, Cambridge, UK.

Shen, H. W., Chiang, L. J., & Ma, K. L. (1999, Oct). A fast volume rendering algorithm for time-varying fields using a time-space partitioning (TSP) tree. *Visualization '99. Proceedings*, (pp. 371-545).

Shi, S., & Hsu, C.-H. (2015). A Survey of Interactive Remote Rendering Systems. *ACM Computing Surveys, 47*, 57.

von Siebenthal, M. (2008). *Analysis and modelling of respiratory liver motion using 4DMRI.* Ph.D. dissertation, Citeseer.

Wang, C., Yu, H., & Ma, K.-L. (2008). Importance-driven time-varying data visualization. *IEEE Transactions on Visualization and Computer Graphics , 14*, 1547-1554.

Younesy, J., Moller, T., & Carr, H. (2005). Visualization of time-varying volumetric data using differential time-histogram table. *Volume Graphics, 2005. Fourth International Workshop on*, (pp. 21-224).