Vision, Modeling, and Visualization (2015)
D. Bommes, T. Ritschel, and T. Schultz (Eds.)

Temporal Coherence Predictor for Time Varying Volume Data Based on Perceptual Functions

Tom Noonan, Lazaro Campoalegre and John Dingliana

Graphics Vision and Visualisation Group, Trinity College Dublin, Ireland.

Abstract

This paper introduces an empirical, perceptually-based method which exploits the temporal coherence in consecutive frames to reduce the CPU-GPU traffic size during real-time visualization of time-varying volume data. In this new scheme, a multi-threaded CPU mechanism simulates GPU pre-rendering functions to characterize the local behaviour of the volume. These functions exploit the temporal coherence in the data to reduce the sending of complete per frame datasets to the GPU. These predictive computations are designed to be simple enough to be run in parallel on the CPU while improving the general performance of GPU rendering. Tests performed provide evidence that we are able to reduce considerably the texture size transferred at each frame without losing visual quality while maintaining performance compared to the sending of entire frames to the GPU. The proposed framework is designed to be scalable to Client/Server network based implementations to deal with multi-user systems.

Categories and Subject Descriptors (according to ACM CCS): I.3.3 [Computer Graphics]: Time-varying data—Parallel Processing Volume Rendering

1. Introduction

Efficient rendering and storage have been the main issues in most areas of time-varying visualization research. In many scientific simulations, exploiting spatial and temporal coherence is a means of avoiding increasing computation cost and reducing display time. Decreasing the time required to transfer a sequence of volumes to the rendering engine is still a considerable challenge. All these improvements must be driven without removing fine features from the handled dataset.

In this paper we design a block-wise approach for visualizing animated volumes. We introduce an empirical, perceptually-based method for exploiting the temporal coherence in consecutive frames to improve performance during the real-time visualization of time-varying volume data.

We propose a technique, where a multi-threaded CPU simulates GPU pre-rendering functions characterizing the local behaviour of the volume datasets, avoiding the sending of complete per frame datasets to the GPU in a real-time volume rendering scheme. Our main contributions are:

- A CPU-GPU Framework for time-varying volume data visualization well suited for volume data from physics simulation based volumes.
- A parallel, predictive and perceptually-based CPU mechanism to improve performance in the visualization of animated volume data with minimal loss of visual quality.
- A scheme that allows a significant reduction of the volume data size uploaded to the GPU, without any assumptions of pre-processing.

2. Previous Work

Many compression techniques are frequently used in time-varying volume data visualization to reduce the loading time and memory consumption on both CPU and GPU. Bernardon et al. [BCCS06] proposed a CPU-GPU compression solution for encoding unstructured grids that allows adaptive time-varying volume visualization. A real-time de-compression and visualization scheme is proposed in [GS01], which uses Wavelets and exploits temporal coherence to achieve interactive frame rates. Lum et al. [LMC02] designed a lossy compression mechanism that uses the texture capability of graphics cards by implementing a palette-based decoding algorithm. Liao et al. [KLW*08] developed

© The Eurographics Association 2015.

DOI: 10.2312/vmv.20151255

delivered by EUROGRAPHICS DIGITAL LIBRARY www.eg.org diglib.eg.org

a hierarchical multi-resolution framework using an octree. They use a predictor based on motion-compensation-based during the octree compression to reduce data size. Westerman [Wes95] designed a technique that discriminates singular points of spatially localized time evolutions to improve the rendering.

Spatial data structures have also been a field of research for scientists in this area. Shen et al. [SCM99] proposed a new data structure called Time Space Partitioning Tree (TSPT) to coordinate both the spatial and the temporal coherence. Wang et al. [WGLS05] proposed a parallel multiresolution rendering framework for large-scale time-varying data visualization using the Wavelet-based time-space partitioning tree, see also [DCS09]. The problem using four-dimensional octrees is that sometimes it is difficult to locate regions with only temporal but not spatial coherence [Ma03].

Temporal coherence based approaches are becoming useful tools for the visualization of animated volumes. Younesy et al. [YMC05], exploit the temporal coherence concept by introducing a novel data structure called Differential Time-Histogram table (DTHT) that stores voxels that are changing between time-steps or during transfer function updates. Fang et al. [FMHC07] developed a time activity curve (TAC) to identify temporal patterns while in [JWSK07], the detection of important regions is achieved by studying the local statistical complexity. Wang et al. [WYM08], compute an importance curve for each data block after applying conditional entropy. Curves are then used to evaluate the temporal behaviour of blocks.

A more recent approach, [JEG12], uses functional representation of time-varying datasets to develop an efficient encoding technique taking into account the temporal similarity between time steps. Akiba et al. [AFM06] proposed a technique that uses time histograms for simultaneous classification of time-varying data by partitioning time histograms into temporally coherent equivalence classes.

Due to the considerable pre-processing stages involved in their pipelines, none of the aforementioned techniques can be classified as real-time solutions.

3. Overview

Our scheme is a synchronized mechanism that involves both CPU and GPU, see Figure 1. It starts by reading from disk an entire time-varying volume dataset, composed of k time steps , $k \in [1,n]$, where n is the amount of time steps in the dataset. We subdivide each time step volume into blocks in the CPU. This subdivision allows the design of a block-wise technique where each block can be processed independently thus making the scheme suitable for parallel environments. After the subdivision, we proceed by sending the first two volumes corresponding to the two first time steps V_0 and V_1 to the GPU. For subsequent time steps, we avoid transferring complete volumes by employing a predictive extrapolation of blocks on the GPU. The premise is that, due to temporal coherence, an extrapolation function based on previous frames will sufficiently approximate the behaviour of a significant percentage of blocks and will be more efficient than transferring the actual contents of the block.

On the CPU, we employ a multi-threaded scheme that starts by finding, in parallel, extrapolated blocks B^i_{New} for each time step $k \mid i \in [1, \frac{Xres}{sBlock} \cdot \frac{Yres}{sBlock} \cdot \frac{Zres}{sBlock}]$, where $Xres$, $Yres$ and $Zres$ are the spatial resolution of each time step volume in each dimension and $sBlock^3$ the size of each independent block. The extrapolated blocks B^i_{New} are computed by applying a linear extrapolation function $E(B_{k-1}, B_k)$, where B_{k-1} and B_k are the corresponding co-spatial blocks (blocks with the same spatial location) of two consecutive time step volumes $k-1$ and k respectively.

The CPU extrapolation is a simulation of the GPU extrapolation. The results of CPU extrapolation are used as parameters of a *similarity function* that evaluates whether a block can or can not be suitably extrapolated in the GPU. A key ingredient of our approach is the method to decide whether to extrapolate or copy a certain block from the original dataset. We use the results of the perceptually-based similarity function $S(B_k, B_{New})$ as explained in Section 5.

The GPU follows the same scheme: if a block B_k is received, the algorithm directly performs an update of the time step volumes on the GPU memory. In the case where the GPU receives the instruction to extrapolate, an extrapolated block B^i_{New} is computed otherwise the block is transferred from dataset before the time step volume is updated and rendered

4. Extrapolation Based Predictor Function

The main objective is to make the CPU responsible for announcing to the GPU whether to render the extrapolated volume blocks B^i_{new} or wait for the new ones from the CPU memory at each frame. By analysing the continuity and the predictable behaviour of physically based scientific simulations, we try to approximate this behaviour as linear within short periods of time. After the subdivision of each per frame volume in blocks B of ($8 \times 8 \times 8$ or $16 \times 16 \times 16$), we apply a linear extrapolation to each voxel inside each block, by forcing the second derivatives (Laplacian) to be 0. Hence, we use the well known Linear Extrapolation Equation: $d_{k+1} = 2 \cdot d_k - d_{k-1}$ as the above mentioned *predictive function*. Note that d_{k-1}, d_k and d_{k+1} are consecutive values of the volume voxel d corresponding to the time step volumes $k-1$, k and $k+1$ respectively.

Algorithm 1, shows the pseudo code of the CPU stage implementation of this temporal coherence predictor scheme. We start by sending a copy of the two first frames of the volume dataset to the GPU memory to perform an initial rendering as explained in Section 6. We also use these two consecu-

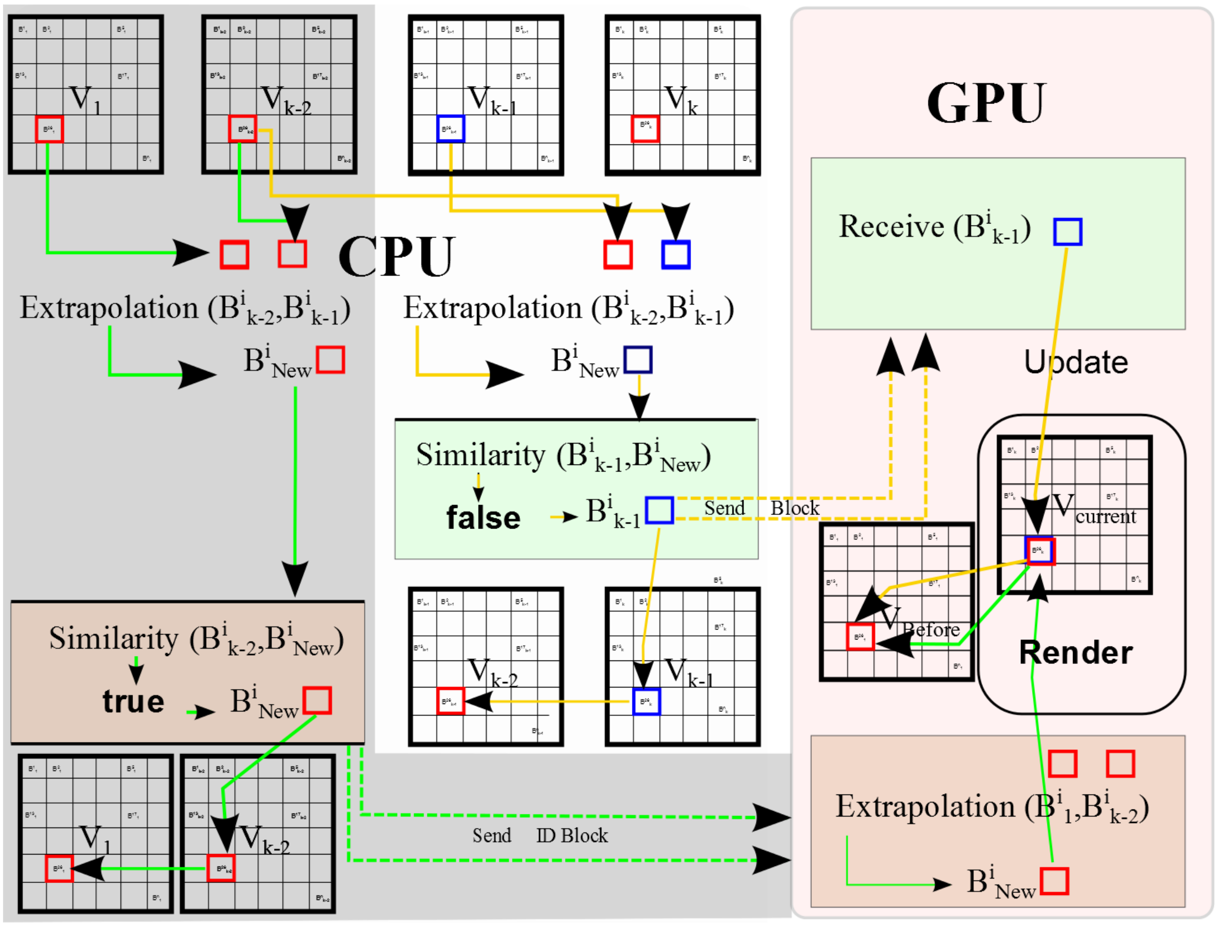
CPU
GPU
V_1
V_{k-2}
V_{k-1}
V_k
Extrapolation (B^i_{k-2},B^i_{k-1})
B^i_{New}
Similarity (B^i_{k-1},B^i_{New})
false
B^i_{k-1}
Send Block
Similarity (B^i_{k-2},B^i_{New})
true
Send ID Block
Receive (B^i_{k-1})
Update
$V_{current}$
V_{Before}
Render
Extrapolation (B^i_1,B^i_{k-2})

Figure 1: *Overview of the proposed Approach*

tive volume frames for starting the pre-rendering simulation in the CPU multi-threaded scheme.

The predictive mechanism starts at frame $k = 2$, where blocks B^i_{new} are calculated in parallel by extrapolating from the two consecutive co-spatial blocks $B^i_{current}$ and B^i_{before}. After this, the same multi-threaded implementation computes a similarity function (see Section 5) that evaluates whether the extrapolation function correctly simulated the temporal coherence between consecutive frames or not. If similarity exits ($S = true$) between the extrapolated block B^i_{new} and the block corresponding to the current frame k, B^i_k, then the algorithm sends a signal with the block identifier (i) to the GPU. Otherwise, if the result of the *similarity function* is false ($S = false$), the corresponding CPU thread sends the block B^i_k to the GPU side. According to the *similarity function* results, the algorithm updates the two consecutive volume frames in the CPU memory. This is done by assigning $B^i_{before} \leftarrow B^i_{current}$ and, $B^i_{current} \leftarrow B^i_{new}$ or $B^i_{current} \leftarrow B^i_k$ whether the *similarity function* is ***true*** or ***false*** respectively.

5. Perceptual Similarity

As mentioned in Section 4, the *similarity function* S, represented in equation 1, evaluates the CPU extrapolation E_C and drives the GPU extrapolation E_G of each block B^i_{new}. The function computes the Root Averaged Square Weighted value between two consecutive co-spatial voxels and compares this to an empirical similarity threshold value ε. The weight (or importance) is calculated using the $H(x_k)$ function corresponding to the frequency each voxel $v_k \in B^i_{current}$ of a particular density x_k appears in the volume.

A study of the visual quality during the rendering of the animated volume allowed the selection of the correct ε values used to discriminate whether the *similarity function* is ***true*** or ***false*** in the tested datasets.

We compared three similarity functions based on different error metrics, computed using the density values of two consecutive voxels with the same spatial location. The first one computes the *Maximum Difference(**Max Diff**)*, the second one performs the *Root Mean Square(**RMS**)* and fi-

Algorithm 1: CPU Multi-thread Extrapolation

```
$V_{before} \leftarrow V_0$;
$Send(V_0)$;
$V_{current} \leftarrow V_1$;
$Send(V_1)$;
for $k \leftarrow 2$ to $k \leftarrow n$ do
    for $B_k \leftarrow B_0^i$ to $B_k \leftarrow B_n^i$ do
        $B_{new} = Extrapolation(B^i_{current}, B^i_{before})$;
        if $Similar(B^i_{new}, B^i_k)$ then
        $Send(O.K, i)$;
        $B^i_{before} \leftarrow B^i_{current}$;
        $B^i_{current} \leftarrow B_{new}$ ;
        else $Send(B^i_k)$;
        $B^i_{before} \leftarrow B^i_{current}$;
        $B^i_{current} \leftarrow B^i_k$ ;
    end
end
```

nally, we calculate the *Root Averaged Square* (see equation 1), weighted by the $H(x_k)$ function as is explained before ***(RASH)***.

$$S = \sqrt{H(x_k) \cdot \frac{(x_k - y_k)^2}{n}} \quad (1)$$

Our next step, during the previous study was computing the HDR-VDP−2 (High Dynamic Range-Visual Difference-Predictor-2), see [MKRH11] between each pairs of frames k_{BF} and k_{PP} corresponding to the same time step, where k_{BF} is a rendered frame using the ***Brute Force*** technique and k_{PP} represents the frame rendered after applying our ***Temporal Coherence Predictive*** scheme. We denominate ***Brute Force*** to the standard rendering process which follows the loading of entire non-compressed frames to the GPU.

We selected the HDR-VDP−2 because it is a recent perceptual metric for measuring the quality degradation between processed and reference images. It is based on a new visual model for all luminance conditions, computed from contrast sensitivity measurements.

6. GPU Extrapolation and Rendering

Accessing and transferring data from the main memory across the graphics bus is relatively slow compared to the direct access of graphics memory. This fact limits the size of the volume that can be interactively rendered. Hence, the loading of the volume data into the graphics card video memory has a special importance for hardware accelerated volume rendering techniques. Our GPU algorithm overcomes this issue by extrapolating new blocks from the two frame volumes present on the GPU memory.

Algorithm 2 shows the pseudo code of our CUDA implementation for the GPU stage. The CPU sends, in parallel, block identifiers or data blocks according to the *similarity function* results. If a new block B^i_k is received, the two frame volumes present in the GPU memory are updated. When any one of the CPU threads send a block identifier i, it means that the extrapolation predictor announces to the GPU to extrapolate the corresponding block. Then, the GPU extrapolated block B^i_{new} is updated into the texture volume $V_{current}$ to be rendered. This updating process also involves: $B^i_{before} \leftarrow B^i_{current}$, where $B^i_{before} \in V_{before}$ and $B^i_{current} \in V_{current}$ as is explained in Algorithm 2.

Algorithm 2: GPU CUDA implementation

```
$V_{before} \leftarrow V_0$;
$receive(V_0)$;
$V_{current} \leftarrow V_1$;
$receive(V_1)$;
if $(B^i_k$ $received)$ then
$B^i_{before} \leftarrow B^i_{current}$;
$B^i_{current} \leftarrow B^i_k$ ;
else
$B^i_{new} = Extrapolation(B^i_{current}, B^i_{before})$;
$B^i_{before} \leftarrow B^i_{current}$;
$B^i_{current} \leftarrow B_{B^i_{new}}$ ;
```

7. CPU performance improvements

Although the proposed scheme is a multi-threaded implementation for dealing with a block based volume representation, computation times were higher than the Brute Force implementation for the tested time variant volume models. We found the extra computational cost of having to extrapolate and compare every voxel to outweigh the gains from the reduction in bandwidth as far as performance is concerned.

The general performance could further be improved by reducing the amount of voxels which must be extrapolated and compared by the CPU. This can be achieved through the use of a mask that selects a subset of voxels to sample in each 8x8x8 block. However the ideal distribution and frequency of samples in this mask needs to be balanced against the reduction in visual quality. The optimal selection of this mask is left for future study.

Frame rates were in the same order of the Brute Force implementation. This scheme exploits parallelism in both the CPU and GPU, and due to the use of CUDA in our implementation we were able to update the upcoming blocks into the current texture directly, without further waste of composing time.

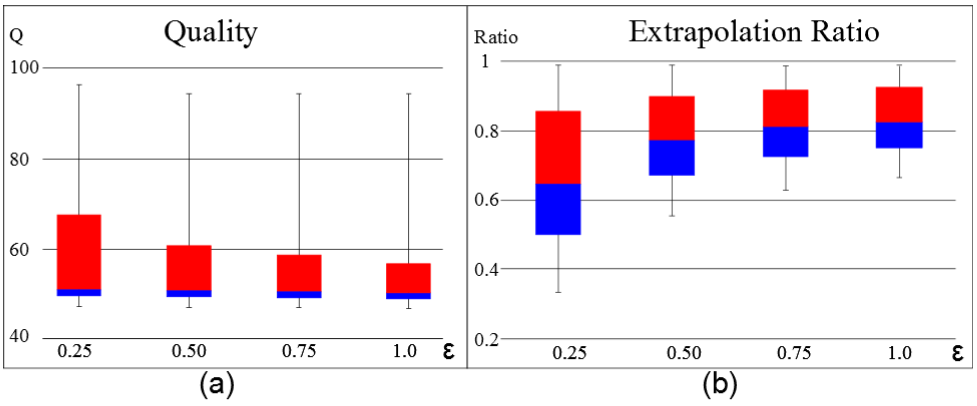

Figure 2: *Behaviour of the HDR-VDP−2 Quality value using different ε values (a). Behaviour of the Extrapolation Ratio using different ε values (b) Both cases represent the results of testing the smoke simulation dataset using the RASH based similarity function.*

8. Results and discussion

As presented in Section 2, many of the previous methods are classified as non real-time solutions due to their use of pre-processing implying that they do not really process data on-the-fly. Thus we compare our approach with a brute-force implementation, which we define as one that uses no pre-processing of the data and contributes no reduction in the bandwidth required. Such brute-force solutions are still widely used in real-time simulations like the one we perform in this paper.

We performed tests with two datasets using three different similarity functions (*RMS*, *Max Diff* and *RASH*) in our temporal coherence framework. Results vary according to the dataset behaviour. We first choose empirically a range of ε values and then restricted that range according to the quality measure (Q) from the HDR-VDP−2 (see Figure 2). This quality measure represents the similarity between reference and test images as is explained in Section 5. A $Q = 100$ Value indicates that both the reference and the test images are equal, while $Q = 0$ represents that these images are completely different. Thus, we decided to use ε values which guarantee a $Q > 50$ value, for each frame and all the possible similarity functions.

Figure 2 shows the results of studying a *smoke simulation* based on the method presented in [FSJ01] with a resolution of $100 \times 100 \times 100$ voxels and 500 time steps. In this case we used the *RASH* based *similarity function*. Quartiles in the graph of Figure 2-(a), represent the HDR-VDP−2 quality value Q for all the frames of the time varying volume data and four different ε values.

As is noted, most of the time step quality values are over the mean, which is always higher than 50. Quartiles in the graph of Figure 2-(b) represent the ratio between the total number of blocks and the amount of extrapolated ones for each frame. The ε values are the same used in the experiment showed in Figure 2-(a). The distance from the mean to the maximum and minimum values provides evidence of a low variation of the extrapolation ratio across time step volumes.

As expected, the visual quality decreases when the extrapolation percentage increases as we are predicting more voxels by our linear extrapolation approximation. The visual quality tends to be similar for $0.5 < \varepsilon < 1.0$. Note that in these cases, Q is around 50 and above with an extrapolation ratio always over 0.5. This fact demonstrates that we are able to reduce at least 50% of the texture size for each frame before loading it into the GPU, by selecting correct ε values.

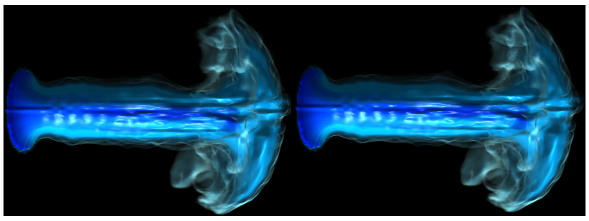

Figure 3: *Image quality results. Rendering of the time step 155 using Brute Force (Right). Rendering of the time step 155 using the RASH based similarity function with* $\varepsilon = 0.25$ *(Left).*

A measure of the temporal predictor effectiveness is shown in Figure 3. Images in this figure show the results of rendering the time step 155 of the smoke simulation by copying the entire volume frame into the GPU (Brute Force), see Figure 3-Right, and by sending only 13% of the volume sub-blocks to the GPU memory (Figure 3-Left). In this case we are extrapolating 87% of the blocks on the GPU. Images in Figure 4 represent a map of the probabilities of detecting differences per pixels [MKRH11]. The Image in Figure 4-Right

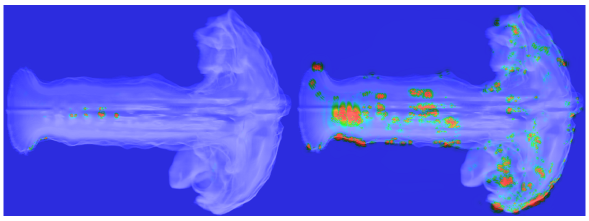

Figure 4: *Map of probabilities of detecting differences. Rendering of the time step 155 using the RASH based similarity function with* $\varepsilon = 0.25$ *(Right). Rendering of the time step 155 using the RASH based similarity function with* $\varepsilon = 0.75$ *(Left).*

shows these results by rendering the studied dataset using the *RASH* based *similarity function* with $\varepsilon = 0.25$. Figure 4-Left shows the result of changing *epsilon* to $\varepsilon = 0.75$.

Visual results achieved with the *RASH* and the *Max Diff* similarity functions are shown in figures 5 and 7 respectively. Images in figures 6 and 8 show the map of probabilities of detecting differences between the source and the resultant images for this two similarity functions.

The mean quality measure Q achieved in the case of the *RASH similarity function* was $Q = 55\%$ with a mean extrapolation ratio of 0.62%. For the *Max Diff similarity function* these values were $Q = 75\%$ and an extrapolation ratio of 0.5%.

The frame rates achieved with our *Temporal Predictor*, were in the same order than the ones we computed for the *Brute Force* scheme, with the advantage that we reduce at least 50% the traffic of data from the CPU to the GPU. This fact validates the current framework to be extensible to Client/Server network based architectures as well as parallel schemes and multi-user based systems.

Our technique does not improve the rendering stage itself but the pre-rendering functions that we parallelise during the simulation on both the CPU and GPU. Table 1 shows a comparison of the pre-rendering functions for the smoke simulation between our technique and the brute-force implementation. Note that the time-step processing-rates for the temporal coherence predictor are in the same order of the pre-rendering stages in the brute-force approach. This analysis allows us to assure that there is not a major decrease of the overall pipeline performance due to temporal coherence pre-rendering functions. That is, we assure an overall improvement because we guarantee the same order of the performance for the pre-rendering and rendering functions while improving the CPU-GPU bandwidth requirements.

The image in Figures 10 and Figures 9, show the rendering after applying a clip plane to the smoke simulation to distinguish the $8 \times 8 \times 8$ blocks extrapolated in frames 90 and 442 respectively. Blocks in red are the ones which are copied

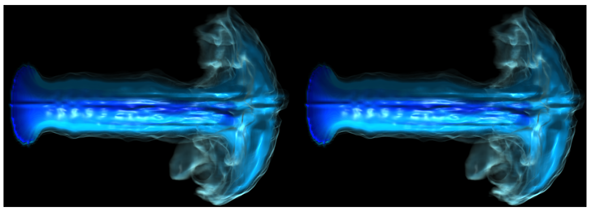

Figure 5: *Image quality results. Rendering of the time step 155 using the RASH based similarity function with* $\varepsilon = 0.25$ *(Left) and* $\varepsilon = 0.75$*(Right)*

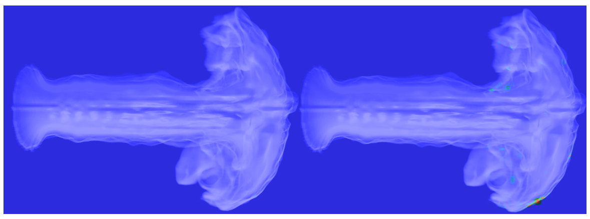

Figure 6: *Map of probabilities of detecting differences. Rendering of the time step 155 using the RASH based similarity function with* $\varepsilon = 0.25$ *(Right) and* $\varepsilon = 0.75$ *(Left).*

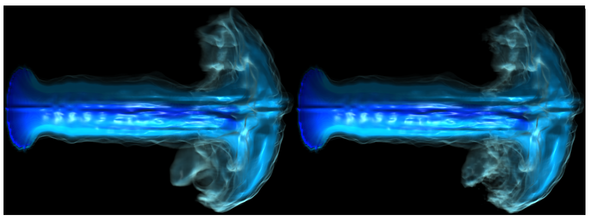

Figure 7: *Image quality results. Rendering of the time step 155 using the Max Diff based similarity function with* $\varepsilon = 1$ *(Left) and* $\varepsilon = 7$*(Right)*

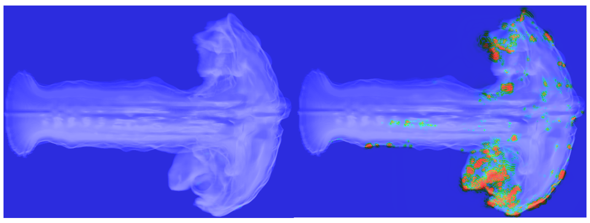

Figure 8: *Map of probabilities of detecting differences. Rendering of the time step 155 using the Max Diff based similarity function with* $\varepsilon = 1$ *(Right) and* $\varepsilon = 7$ *(Left).*

to the GPU, while the white blocks are extrapolated from the information of the previous cospatial frames as explained in section 6. Note that a considerable amount of blocks in the non *null* region are also extrapolated.

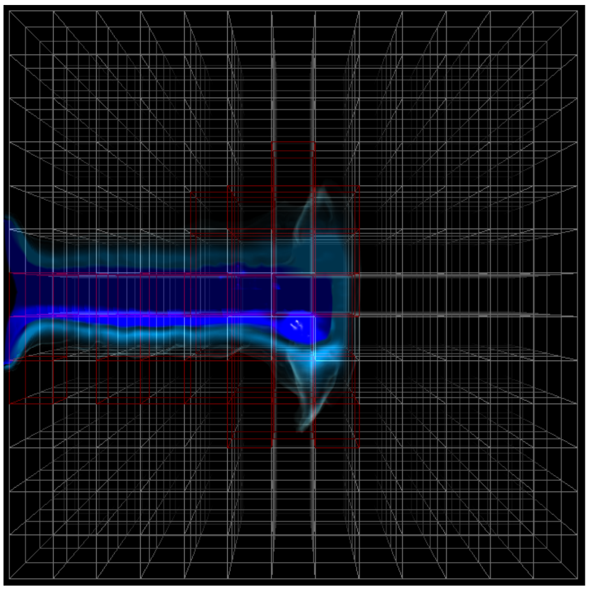

Figure 9: *Rendering of the smoke simulation at frame 90. The image shows a cut planar for clarification. The white blocks in the block subdivision are the ones which are extrapolated in the GPU.*

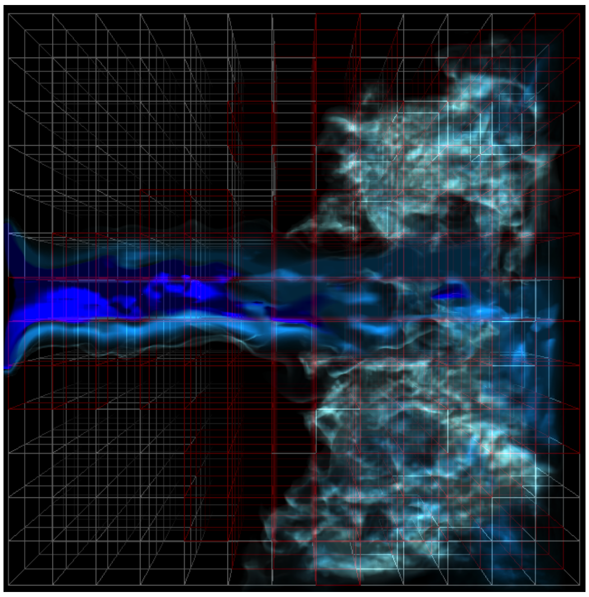

Figure 10: *Rendering of the smoke simulation at frame 442. The image shows a cut planar for clarification. The white blocks in the block subdivision are the ones which are extrapolated in the GPU.*

Limitations: As we mention in section 9 we would like to extend the predictive functions to a finer filter scheme. Our current implementation is only able to deal with physics simulations that can be approximated by analysing the behaviour of the partial difference equations, thus a generalisation of this proposal is still needed. We also need to perform a rigorous user test for perceptual evaluation of the required parameters. A unique method to adjust the parameters with different datasets is still required

	Without Rendering	With Rendering
Brute Force	0.135ms	29.635ms
Ep = 0.25	0.1695ms	38.485ms
Ep = 0.50	0.1624ms	37.527ms
Ep = 0.75	0.1604ms	36.544ms
Ep = 1.00	0.1577ms	36.068ms

Table 1: *A comparison of the impact that our technique has on average frame times. Here we compare Brute Force against results from our RASH similarity function.*

9. Conclusions and Future Work

We have proposed a new framework for time-varying-volume data visualization, well suited for physics simulation based volumes. Our scheme reduces the volume data size uploaded to the GPU by performing a *predictive function* that exploits the local temporal coherence among consecutive frames. Tests performed provide evidence that we are able to reduce the texture transfer bandwidth by at least 50% at each frame with a minimal loss of the visual quality.

The frame rates achieved with our *Temporal Predictor*, were in the same order than the ones we computed for the *Brute Force* scheme, with the advantage that we reduce at least 50% the traffic of data from the CPU to the GPU.

In future work we expect to investigate the use of adapted block size according to the temporal volume behaviour. We want to replace the Linear Extrapolation function with a more complex scheme based on the use of local filters as predictive functions. Increasing quality measures by using an analysis function that discriminates with exactness the behaviour of each frame, should be a future improvement. We need this analysis function to be fast enough, to be computed in real-time on the GPU side. The results of this analysis function could then be sent as parameters for the *similarity function* to the CPU side.

We also expect to refine the error metric involved in this framework by including perceptual evaluation from user tests. And finally we also realise that the reduction in bandwidth we have achieved will have far greater positive effects in environments which are more inherently bandwidth constrained, such as over a network. Because of this we would also like to extend our scheme to a Client/Server network based architecture to accommodate multi-user systems being able to render time variant datasets over a network.

10. Acknowledgements

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

References

[AFM06] Akiba H., Fout N., Ma K.-L.: Simultaneous classification of time-varying volume data based on the time histogram. In *Proceedings of the Eighth Joint Eurographics / IEEE VGTC Conference on Visualization* (2006), Eurographics Association, pp. 171–178. 2

[BCCS06] Bernardon F. F., Callahan S. P., Comba J. a. L. D., Silva C. T.: Interactive volume rendering of unstructured grids with time-varying scalar fields. In *Proceedings of the 6th Eurographics Conference on Parallel Graphics and Visualization* (2006), pp. 51–58. 1

[DCS09] Du Z., Chiang Y.-J., Shen H.-W.: Out-of-core volume rendering for time-varying fields using a space-partitioning time (spt) tree. In *Visualization Symposium, 2009. PacificVis '09. IEEE Pacific* (April 2009), pp. 73–80. doi:10.1109/PACIFICVIS.2009.4906840. 2

[FMHC07] Fang Z., Möller T., Hamarneh G., Celler A.: Visualization and exploration of time-varying medical image data sets. In *Proceedings of Graphics Interface 2007* (2007), GI '07, ACM, pp. 281–288. 2

[FSJ01] Fedkiw R., Stam J., Jensen H. W.: Visual simulation of smoke. In *Proceedings of the 28th annual conference on Computer graphics and interactive techniques* (2001), ACM, pp. 15–22. 5

[GS01] Guthe S., Strasser W.: Real-time decompression and visualization of animated volume data. In *Visualization, 2001. VIS '01. Proceedings* (Oct 2001), pp. 349–572. 1

[JEG12] Jang Y., Ebert D. S., Gaither K.: Time-varying data visualization using functional representations. *Visualization and Computer Graphics, IEEE Transactions on 18*, 3 (2012), 421–433. 2

[JWSK07] Janicke H., Wiebel A., Scheuermann G., Kollmann W.: Multifield visualization using local statistical complexity. *IEEE Transactions onVisualization and Computer Graphics 13*, 6 (2007), 1384–1391. 2

[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 *IEEE Pacific Visualization Symposium* (2008), pp. 135–142. 1

[LMC02] Lum E. B., Ma K.-L., Clyne J.: A hardware-assisted scalable solution for interactive volume rendering of time-varying data. *IEEE Transactions on Visualization and Computer Graphics 8*, 3 (July 2002), 286–301. 1

[Ma03] Ma K.-L.: Visualizing time-varying volume data. *Computing in Science & Engineering 5*, 2 (2003), 34–42. 2

[MKRH11] Mantiuk R., Kim K. J., Rempel A. G., Heidrich W.: Hdr-vdp-2: A calibrated visual metric for visibility and quality predictions in all luminance conditions. *ACM Trans. Graph. 30*, 4 (2011), 40:1–40:14. 4, 5

[SCM99] Shen H.-W., Chiang L.-J., Ma K.-L.: A fast volume rendering algorithm for time-varying fields using a time-space partitioning (tsp) tree. In *Proceedings of the conference on Visualization* (1999), IEEE Computer Society Press, pp. 371–377. 2

[Wes95] Westermann R.: Compression domain rendering of time-resolved volume data. In *Proceedings of the 6th Conference on Visualization* (1995), IEEE Computer Society. 2

[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 the Fourth Eurographics/IEEE VGTC conference on Volume Graphics* (2005), pp. 11–19. 2

[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. 2

[YMC05] Younesy J., Moller T., Carr H.: Visualization of time-varying volumetric data using differential time-histogram table. In *Volume Graphics, 2005. Fourth International Workshop on* (2005), pp. 21–224. 2