Light-Field DVR on GPU for Streaming Time-Varying Data

David Ganter†, Martin Alain†, David Hardman†, Aljoscha Smolić†, Michael Manzke†

GV2, Trinity College Dublin, Ireland

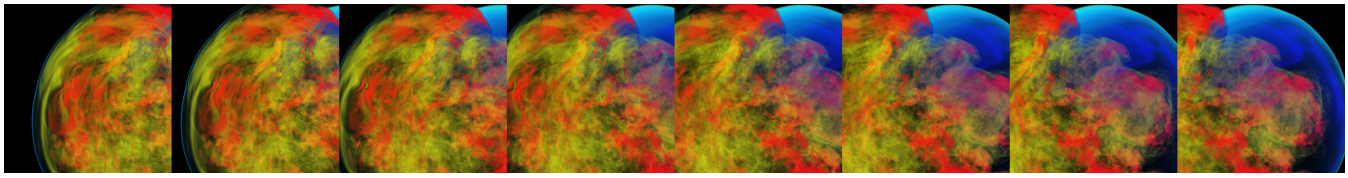

Figure 1: *An 8x1 sub-section from a light-field of* 512^2 *pixel sub-images of a supernova volume dataset. [Blo07]*

Abstract

Direct Volume Rendering (DVR) of volume data can be a memory intensive task in terms of footprint and cache-coherency. Ray-guided methods may not be the best option to interactively render to light-fields due to feedback loops and sporadic sampling, and pre-computation can rule out time-varying data. We present a pipelined approach to schedule the rendering of sub-regions of streaming time-varying volume data while minimising intermediate sub-buffers needed, sharing the work load between CPU and GPU. We show there is significant advantage to using such an approach.

CCS Concepts

•***Computing methodologies*** → ***Rendering;*** *Parallel algorithms; Graphics systems and interfaces;*

Keywords: Direct Volume Rendering, Light Fields, GPU, Data Visualisation, Time Varying Volume Data, Cache Coherency, Object-Order, Sort-Last Rendering

1. Introduction

For many streaming volume data applications such as 4D MRI, 4D ultrasound or CFD simulations, direct volume rendering (DVR) is a prevalent method. It is desirable for users to view these datasets in real-time while changing the view-point or transfer-function. Parallax via light-field (LF) displays can give a greater understanding of the data [RZDdW14], and further perceptual enhancement and eye-strain prevention can be achieved by near-eye light-field displays [LL13]. When rendering to a single-view, rays are generally closely coupled and sample data with good spatial locality. This can change when rendering to a LF display when rays from different views sample more sporadically. Work has been done on displaying volume data to a LF display [IGM10, AGIG*08] which work well when data is static or updates without a latency requirement [HBJP12], but are not ideally suited for time-varying (TV) data. TV DVR has also been researched [SZ03, ZEP09, NCD15] but few papers focus on rendering to a LF display. Approaches that do [Rui09, RZDdW14, BVL*16] are either quite simple or resort to down-sampled data or image-based techniques.

We present a method to make efficient use of existing GPUs to schedule the rendering of regions of streaming TV volumes to all views of a LF in one pass, exploiting the on-chip L2 cache. We focus on the performance of LF DVR at practical screen and data resolutions and dimensions, projecting sub-volumes that fit in the L2 cache to all views before progressing to the next sub-volume, enhancing cache coherency. We exploit sub-volumes close in render order to minimise the amount of intermediate sub-buffers, reducing the complexity and memory-footprint of standard sort-last rendering. We present this as a pipeline for splitting the view-dependent working-set determination and rendering between the CPU and GPU. We evaluate our method, showing significant performance gain in the render kernel in comparison to octree-based empty-space-skipping as a result of improved L2 cache utilisation.

2. Background & Related Work

Using DVR [Lev88] a volume is rendered by integrating over rays, accumulating colour and opacity, generally discretised as a Riemann sum. It is common to divide the volume into bricks

† {surname}{firstinitial}@scss.tcd.ie (e.g. ganterd@scss.tcd.ie)

and use a data-structure for a reduced working-set to be determined and efficiently traversed during rendering [CNLE09,IGM10, KTW*11, FSK13]. Approaches can use the CPU as the sole renderer [Kni00, KTW*11], or use a ray-guided feedback loop between the CPU and GPU [GMIG08, FSK13, HBJP12, HAAB*18]. This feedback loop prevents the pipe-lining of streaming data. Liu et al. [LCDP13] note that the CPU is a better candidate for tree traversal and thus, determining the working set. These methods require pre-processing making them unsuitable for TV data, and do not take explicit advantage of cache-coherency. TV data adds pressure to memory storage and bandwidth, and approaches either simplistically transfer the full volume per-frame [ZEP09], use dynamic-resolution [Rui09, RZDdW14] or use brick extrapolation [NCD15], introducing artefacts. As a side note, we currently consider compression [BGIG*14] to be a stage prior to our pipeline, thus out of scope. Sort-last (SL) approaches render bricks out-of-order or in parallel to sub-buffers, compositing in-order to the final image [Hsu93,MPHK94], presenting a trade-off of concurrency versus memory footprint and complexity. Samanta et al. [SFLS00] presented a hybrid sort-first / sort-last polygonal rendering approach for a cluster. Our approach draws inspiration and applies a similar method to GPU DVR.

The light-field (LF) was introduced in [LH96] as an efficient image-based rendering (IBR) method. A LF aims to capture all rays passing through a given space, commonly parametrised using two parallel planes, discretised as a collection of sub-aperture images arranged on a 2D grid. We choose to use this parametrisation as it can be used to interface multiple rendering methods [LH96, IMG00] and displays [WLHR12, HLW15]. Current LF DVR methods generally use the full volume [Rui09, RZDdW14, KPE*12, BVL*16] and are either application oriented or use IBR techniques. Ray-guided approaches have also been proposed for rendering large static volumes to LF displays [IGM10], but suffer the same set-backs as previously mentioned.

A shared 2MB L2 cache is shared by the streaming-multiprocessors in Nvidia's Pascal architectures, which we exploit in our approach by enhancing spatio-temporal sampling coherence. Concurrency is limited by registers or memory usage per-thread, and while it is easy to consider the DVR as a set of simultaneously progressing rays, in reality the volume is rendered in thread blocks. This means rays in one tile may exit the volume before the next set of thread blocks begins, leading to the L2 cache being thrashed before the next set of rays start. Our approach limits render kernels to complete a volume brick before progressing, achieving enhanced cache coherency.

3. Pipelined Brick-Based LFDVR

The core concepts of our method are the following: Projecting one sub-L2-cache-sized brick to all LF sub-aperture images before progressing to the next brick for more cache-efficient sampling access by forcing all GPU thread blocks to remain in the same volume region. Since the render order of bricks may be different for each sub-aperture image, each brick renders to an intermediate buffer (sub-buffer). We exploit bricks that can use the same sub-buffers as their neighbours to reduce memory footprint and compositing complexity. We pipeline asynchronous tasks on the CPU and the GPU. The pipeline is structured as the following steps:

1. **CPU** Empty-space-skipping (ESS) information generation.
2. **CPU** Per-view view-dependent render-order list determination.
3. **CPU** Per-view brick sub-buffer amalgamation (minimisation).
4. **Both** Working-set data transfer.
5. **GPU** Per-brick render kernels.
6. **GPU** Sub-buffer compositing kernels to final image.

Simple ESS data-structure generation has been covered in the literature extensively, and the rendering method itself is relatively standard, thus we only expand upon view-list generation and the sub-buffer minimisation.

View-List Generation: To generate a per-view render list we first determine the overall order which bricks will be scheduled. We order these by distance from the camera plane centre point. Bricks that are considered empty are *not* placed on this list, thus are now excluded. We then generate the per-view view-dependent render lists in parallel on the CPU. These lists represent the order in which bricks need to be rendered *for a particular view* to produce the correct result. These lists are determined by using a flood-fill algorithm starting from the closest brick in a view frustum.

Sub-Buffer Minimisation: In algorithm 1 we outline our method to minimise the amount of sub-buffers. We exploit the fact that neighbouring bricks will be close enough in the overall render order that they may share a buffer maintaining the front-to-back compositing order. The algorithm loops through all bricks for a given view. For each brick that does not have an assigned buffer (all of them in the beginning) a new buffer is created with the dimensions of the brick's projection bounds. We then recursively traverse the forward descendants of the brick (i.e bricks which are further from the view-point by manhattan distance) testing for existing assigned buffers. If there is no assigned buffer for a descendant *and* it is later in the render order, it is assigned the current buffer, and the buffer dimensions are expanded to accommodate the descendant's projection bounds. With all sub-buffer information generated we *now* allocate the space for these sub-buffers on the GPU.

Algorithm 1 Sub-buffer minimisation algorithm.

```
function MinimiseBuffersForView(View v)
    List brickList = view.brickListInOrder
    for all Brick b in brickList do
        if !b.HasBufferForView(v) then
            Buffer s = new Buffer();
            RecursiveAddBuffer(b, v, s);
function RecursiveAddBuffer(Brick b, View v, Buffer s)
    if b.HasBufferForView(v) then
        return;
    b.BufferForView(v) = s;
    s.AccommodateBrickProjectionBounds(b);
    for all Brick n in b.forwardNeighbours do
        if n.renderIndex > b.renderIndex then
            RecursiveAddBuffer(n, v, s);
```

Compositing Kernel: In algorithm 1 we generated a list of sub-buffers, which we can trivially order from front-to-back. Using this

list we divide the final-image screen-space into tiles, which we refer to as screen-tiles. Each of these screen-tiles is the work item of a CUDA thread block. Each thread block traverses the ordered list from front-to-back, testing each sub-buffer in the ordered list for contribution to it's target screen-tile. If it does, the brick-tile is composited to the final image, otherwise it is skipped. This method ensures that a correct final image is composited.

4. Implementation & Evaluation

Our system used an Nvidia GTX1080 GPU and Intel Xeon E5-1620 v3 CPU. Experiments involved a LF camera plane doing a full rotation in 50 steps around a volume. Results presented were obtained using a 512^3 floating-point (FP32) volume of a supernova [Blo07]. Similar results excluded for brevity were also obtained using datasets exceeding the L2 cache size. We expect bricks that have a data size of sub-L2 size (2MB) to perform best, meaning approximately 80^3 voxels for FP32 data. Taking into account other access requirements for other rendering kernels, we estimate brick sizes of sub 72^3 voxels to have superior cache utilisation. Considering computational overhead and complexity of small bricks, we further focus our expected performant range to between 32^3 to 72^3 voxels. Our experiments prove that this is the case. Since our application area is relatively unresearched we have little to compare against. Ray-guided approaches require too much off-line pre-processing for streaming volumes, and multi-view streaming dataset DVR approaches are either quite simple or introduce artefacts via IBR techniques. We compromise by implementing both a naive ray-caster and an ESS octree ray-caster as comparisons.

View-Count, Brick Size & Target Resolution: We vary the brick size from 32^3 up to 512^3, and the LF view-count from 1 view to 32^2 views, timing the render kernels of the ray-casters independently. Note that the view-count is a sub-division of the target screen resolution. The volume is also rendered with shading, introducing more GPU computation and intensity on the cache. Figure 2 shows the results at a target resolution of $4K^2$. For the expected performant range of brick sizes our approach outperforms octree ESS as much as x2. While the graphs omit view-counts 1, 2^2, 4^2 for brevity, we still discuss the results: While our approach performs well in a many-view small-brick scenario, smaller view-counts and larger brick sizes reveal a computational overhead of our approach and we no-longer gain temporal-locality advantage. We also examined the effect of changing the target screen-buffer resolution parameter between $1K^2$, $2K^2$ and $4K^2$ for all view counts with varying brick sizes. We observed that for a relatively small resolution size of $1K^2$ our approach slightly under-performs in relation to the ESS approach due to overhead, with this drop exaggerated with a smaller than 32^3 brick size. However, with a brick-size of approximately 64^3 we outperform ESS regardless of resolution.

Render List, Buffer Minimisation & Compositing: Timing the individual components of our pipeline, we observed that the sections overlap well without the CPU becoming the bottleneck in most cases. The exception to this is when the brick size drops to about 32^3, when the computational overhead on the CPU rises dramatically, as seen in figure 3. This, coupled with the overhead on the GPU, is reason to keep brick-sizes in our approach in the sweet-spot of approximately 64^3 for FP32 data.

L2 Cache Statistics & Scalability: To confirm our base assumption of better utilisation of the GPU's L2 cache, we used the profiler 'nvprof' to query the metric 'l2_tex_hit_rate'. While we attempted this with a $4K^2$ screen-buffer, nvprof reported metric overflow errors. Compromising, we profiled with a $2K^2$ screen-buffer with 32^2 views and a brick size of 64^3 FP32 data, testing the naive, octree ESS, and our brick-based ray-casters. We observed a substantial hit-rate improvement from an ~50% hit rate for the octree ray-caster versus naive, to an ~87% hit-rate for our brick-based ray-caster. In addition to the GTX1080, we also tested rendering performance a Quadro K2200, observing the same performance gain for performant brick sizes, showing that our approach scales with shader cores.

5. Conclusion & Future Work

We have presented an approach in the relatively untapped area of LF DVR of streaming TV data. We have shown that by forcing increased spatio-temporal sampling we benefit substantially when rendering streaming TV volumes to a LF. By limiting regions of the volume to a certain size and using them only once, we can exploit temporal-coherency in the shared L2 cache on the GPU and gain a performance increase of up to 2x in contrast to an octree ESS approach when multiple views are presented. Our approach is compatible with streaming TV data without need for extensive pre-processing. In the future, we believe that we can further exploit frame-to-frame coherency both for view-list generation and memory allocation, perhaps leveraging light-weight easy-to-update data-structures to accelerate the working-set determination.

Acknowledgements

This publication has emanated from research conducted with the financial support of Science Foundation Ireland (SFI) under the Grant Numbers 13/IA/1895 and 15/RP/2776.

References

[AGIG*08] AGUS M., GOBBETTI E., IGLESIAS-GUITIÁN J. A., MARTON F., PINTORE G.: GPU accelerated direct volume rendering on an interactive light field display. *Computer Graphics Forum 27*, 2 (2008), 231–240. 1

[BGIG*14] BALSA RODRÍGUEZ M., GOBBETTI E., IGLESIAS-GUITIÁN J. A., MAKHINYA M., MARTON F., PAJAROLA R., SUTER S. K.: State-of-the-art in compressed GPU-based direct volume rendering. *Computer Graphics Forum 33*, 6 (2014), 77–100. 2

[Blo07] BLONDIN J.: Supernova Modelling, 2007. URL: http://vis.cs.ucdavis.edu/VisFiles/pages/supernova.php. 1, 3

[BVL*16] BATTIN B., VALETTE G., LEHURAUX J., REMION Y., LUCAS L.: A premixed autostereoscopic OptiX-based volume rendering. In *Proceedings of the 2015 International Conference on 3D Imaging 2015 - IC3D 2015* (2016), vol. 31 of *SIGGRAPH '11*, pp. 3–10. 1, 2

[CNLE09] CRASSIN C., NEYRET F., LEFEBVRE S., EISEMANN E.: GigaVoxels. In *Proceedings of the 2009 symposium on Interactive 3D graphics and games - I3D '09* (2009), I3D '09, p. 15. 2

[FSK13] FOGAL T., SCHIEWE A., KRUGER J.: An analysis of scalable GPU-based ray-guided volume rendering. In *Proceedings of the IEEE Symposium on Large Data Analysis and Visualization 2013 - LDAV 2013* (2013), pp. 43–51. 2

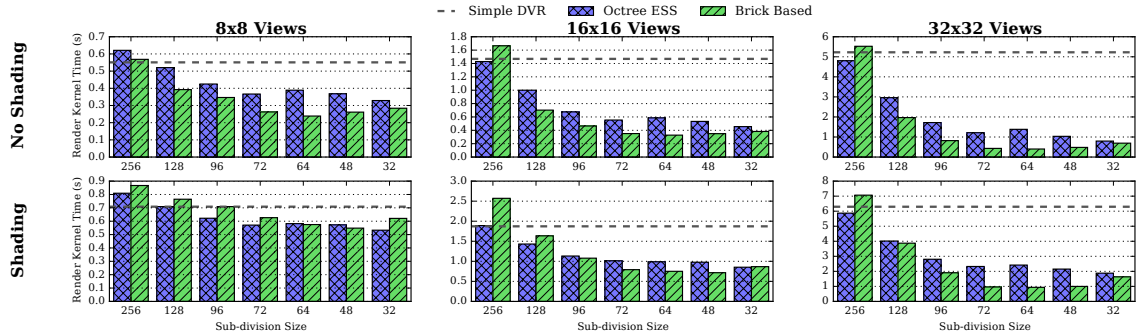

Figure 2: *Render kernel times for multiple configurations of view count and brick size for a target resolution of $4K^2$ rendering a 512^3 floating-point volume with and without shading on the GTX 1080. We see a performance gain when brick sizes drop below the L2 size.*

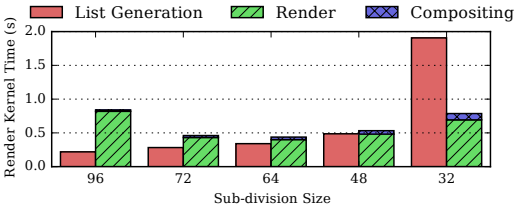

Figure 3: *Individual component timings for rendering without shading to a $4K^2$ buffer of 32^2 views. Render and compositing kernels are stacked as they are sequential. This further confirms our sweet-spot to be a brick-size of approximately 64^3 for FP32 data.*

[GMIG08] 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. *Visual Computer 24*, 7-9 (2008), 797–806. 2

[HAAB*18] HADWIGER M., AL-AWAMI A. K., BEYER J., AGUS M., PFISTER H.: SparseLeap: Efficient Empty Space Skipping for Large-Scale Volume Rendering. *IEEE Transactions on Visualization and Computer Graphics 24*, 1 (2018), 974–983. 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 Transactions on Visualization and Computer Graphics 18*, 12 (2012), 2285–2294. 1, 2

[HLW15] HUANG F.-C., LUEBKE D., WETZSTEIN G.: The light field stereoscope. *ACM SIGGRAPH 2015 Emerging Technologies on - SIGGRAPH '15 34*, 4 (2015), 1–1. 2

[Hsu93] HSU W. M.: Segmented ray casting for data parallel volume rendering. In *Proceedings of the 1993 symposium on Parallel rendering - PRS '93* (1993), vol. d of *PRS '93*, pp. 7–14. 2

[IGM10] IGLESIAS-GUITIÁN J. A., GOBBETTI E., MARTON F.: View-dependent exploration of massive volumetric models on large-scale light field displays. In *Visual Computer* (2010), vol. 26, pp. 1037–1047. 1, 2

[IMG00] ISAKSEN A., MCMILLAN L., GORTLER S. J.: Dynamically reparameterized light fields. In *Proceedings of the 27th annual conference on Computer graphics and interactive techniques - SIGGRAPH '00* (2000), pp. 297–306. 2

[Kni00] KNITTEL G.: The ULTRAVIS System. In *2000 IEEE Symposium on Volume Visualization, VV 2000* (2000), pp. 71–79. 2

[KPE*12] KWON K.-C., PARK C., ERDENEBAT M.-U., JEONG J.-S., CHOI J.-H., KIM N., PARK J.-H., LIM Y.-T., YOO K.-H.: High speed image space parallel processing for computer-generated integral imaging system. *Optics Express 20*, 2 (2012), 732. 2

[KTW*11] KNOLL A., THELEN S., WALD I., HANSEN C. D., HAGEN H., PAPKA M. E.: Full-resolution interactive CPU volume rendering with coherent BVH traversal. In *Proceedings of the IEEE Pacific Visualization Symposium 2011 - PacificVis 2011* (2011), pp. 3–10. 2

[LCDP13] LIU B., CLAPWORTHY G. J., DONG F., PRAKASH E. C.: Octree rasterization: Accelerating high-quality out-of-core GPU volume rendering. *IEEE Transactions on Visualization and Computer Graphics 19*, 10 (2013), 1732–1745. 2

[Lev88] LEVOY M.: Display of Surfaces from Volume Data. *IEEE Computer Graphics and Applications 8*, 3 (1988), 29–37. 1

[LH96] LEVOY M., HANRAHAN P.: Light field rendering. In *Proceedings of the 23rd annual conference on Computer graphics and interactive techniques - SIGGRAPH '96* (1996), pp. 31–42. 2

[LL13] LANMAN D., LUEBKE D.: Near-eye light field displays. *ACM Transactions on Graphics 32*, 6 (2013), 1–10. 1

[MPHK94] MA K. L., PAINTER J. S., HANSEN C. D., KROGH M. F.: Parallel Volume Rendering Using Binary-Swap Compositing. *IEEE Computer Graphics and Applications 14*, 4 (1994), 59–68. 2

[NCD15] NOONAN T., CAMPOALEGRE L., DINGLIANA J.: Temporal Coherence Predictor for Time Varying Volume Data Based on Perceptual Functions. In *Vision, Modeling & Visualization* (2015), The Eurographics Association, pp. 33–40. 1, 2

[Rui09] RUIJTERS D.: Dynamic resolution in GPU-accelerated volume rendering to autostereoscopic multiview lenticular displays. *Eurasip Journal on Advances in Signal Processing 2009* (2009). 1, 2

[RZDdW14] RUIJTERS D., ZINGER S., DO L., DE WITH P. H.: Latency optimization for autostereoscopic volumetric visualization in image-guided interventions. *Neurocomputing 144* (2014), 119–127. 1, 2

[SFLS00] SAMANTA R., FUNKHOUSER T., LI K., SINGH J. P.: Hybrid sort-first and sort-last parallel rendering with a cluster of PCs. In *ACM SIGGRAPH/Eurographics Workshop on Graphics Hardware - HWWS '00* (2000), pp. 97–108. 2

[SZ03] SHEKHAR R., ZAGRODSKY V.: Cine MPR: Interactive Multiplanar Reformatting of Four-Dimensional Cardiac Data Using Hardware-Accelerated Texture Mapping. *IEEE Transactions on Information Technology in Biomedicine 7*, 4 (2003), 384–393. 1

[WLHR12] WETZSTEIN G., LANMAN D., HIRSCH M., RASKAR R.: Tensor displays. *ACM Transactions on Graphics 31*, 4 (2012), 1–11. 2

[ZEP09] ZHANG Q., EAGLESON R., PETERS T. M.: Dynamic real-time 4D cardiac MDCT image display using GPU-accelerated volume rendering. *Computerized Medical Imaging and Graphics 33*, 6 (2009), 461–476. 1, 2