Visualization of Big SPH Simulations via Compressed Octree Grids

Florian Reichl, Marc Treib, and Rüdiger Westermann
Computer Graphics & Visualization Group
Technische Universität München
Munich, Germany
Email: reichlf@in.tum.de, {treib,westermann}@tum.de

***Abstract*—Interactive and high-quality visualization of spatially continuous 3D fields represented by scattered distributions of billions of particles is challenging. One common approach is to resample the quantities carried by the particles to a regular grid and to render the grid via volume ray-casting. In large-scale applications such as astrophysics, however, the required grid resolution can easily exceed 10K samples per spatial dimension, letting resampling approaches appear unfeasible. In this paper we demonstrate that even in these extreme cases such approaches perform surprisingly well, both in terms of memory requirement and rendering performance. We resample the particle data to a multiresolution multiblock grid, where the resolution of the blocks is dictated by the particle distribution. From this structure we build an octree grid, and we then compress each block in the hierarchy at no visual loss using wavelet-based compression. Since decompression can be performed on the GPU, it can be integrated effectively into GPU-based out-of-core volume ray-casting. We compare our approach to the perspective grid approach which resamples at run-time into a view-aligned grid. We demonstrate considerably faster rendering times at high quality, at only a moderate memory increase compared to the raw particle set.**

***Keywords*-SPH; volume rendering; data compression;**

I. Introduction

In particle-based simulation techniques such as smoothed particle hydrodynamics (SPH), a volume covering a continuous field is reconstructed from a discrete set of particles carrying physical quantities. Since the particles are scattered irregularly over the 3D spatial domain, the reconstruction process requires determining at every spatial domain point the set of particles having influence on this point, and weighting their contributions to form the resulting value. Rendering SPH data can be done via volume ray-casting, by determining at every sampling point along the rays the influencing particle set and accumulating the respective values. This, however, requires huge numbers of search queries per ray to determine these sets locally.

A different approach is to resample the particle data into a grid in a pre-process and use cell-wise interpolation for reconstructing the values along the rays during rendering. This approach results in a view-independent volume representation and does not require any overlap queries at runtime. If the data is resampled to a uniform grid, 3D texture-based volume rendering on the GPU can be used.

This resampling grid introduces a significant memory overhead, especially when the particle density dictates a high resolution to capture all simulation details. For instance, in the gas dynamics simulation addressed in the current work—the Millennium Run—more than 10 billion particles were used to trace the evolution of the matter distribution in a cubic region of the universe. The spatial resolution of this simulation corresponds to an effective uniform grid size of about $100{,}000^3$.

To overcome the memory overhead of a uniform grid in object space, a non-uniform discretization of the view frustum was introduced by Fraedrich et al. [1]: the perspective grid. In every frame, the particles affecting the data values at points in the view frustum are resampled into this grid. This work also relies on a particle level-of-detail (LoD) structure to effectively reduce the number of resampled particles with decreasing sampling frequency. This structure represents the particle set at ever coarser resolutions with fewer and fewer particles, and it is computed in a pre-process based on specific particle merging rules. The perspective grid is very memory-efficient, but it requires resampling large sets of particles in every frame; this process vastly dominates the rendering times and limits the performance significantly.

A. Contribution

In this work we shed light on the question whether the performance limitations of the perspective grid approach can be overcome by resampling to a uniform grid in a pre-process and using volume ray-casting on this grid. While this seems possible because at runtime any resampling of the particle data to a grid can be avoided, it is unclear whether bandwidth limitations due to the tremendous memory requirement of a uniform grid can be overcome.

We address this problem in two ways: Firstly, we construct an octree grid which adapts locally to the resolution necessary to capture all simulation details. In this way, the memory requirement of the grid structure can be reduced considerably. Furthermore, standard GPU LoD ray-casting can be employed to lower the memory requirement per frame. Secondly, we embed a compression layer into our out-of-core volume ray-casting system to further reduce the octree's size in memory and to reduce bandwidth requirements when streaming the data from disk. Because we

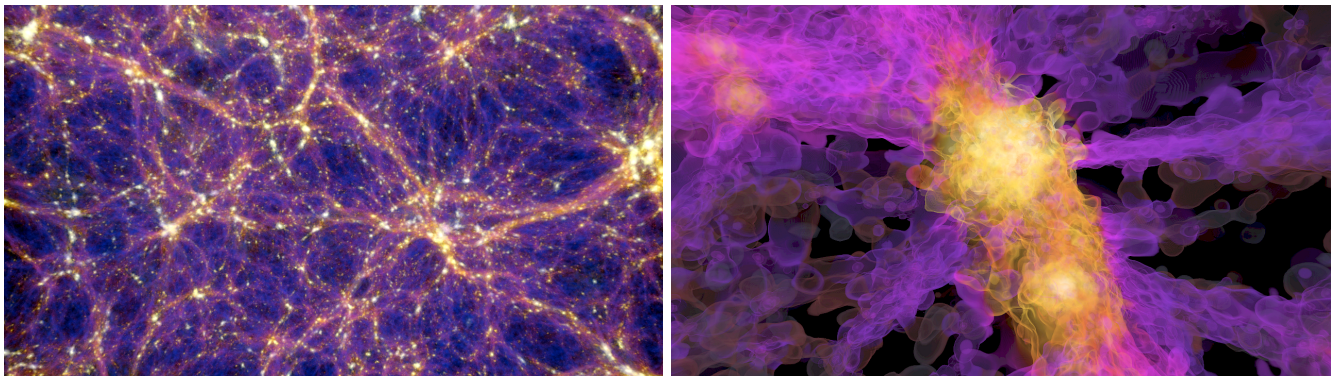

Figure 1. Cosmological structures in the Millennium Run at different resolutions. Rendering to a 1200×800 viewport took 213 ms and 79 ms per frame, respectively, using multiresolution volume ray-casting. The working sets in GPU memory were 2120 MB and 109 MB, respectively.

use a wavelet-based compression scheme which allows fast decoding on the GPU, very large parts of the data set can be cached on the CPU and bandwidth limitations caused by CPU-GPU data transfer can be reduced.

Based on the aforementioned concepts we have designed a system which first converts large SPH particle sets into a compact adaptive multiresolution representation and then renders the data from this representation on the GPU.

The remainder of this paper is organized as follows: First, we review existing work in the context of the Millennium Run and SPH visualization as well as general high-resolution volume rendering in section II. Section III gives a high-level overview of our system, while sections IV and V detail the efficient implementation of the necessary preprocess and our GPU-based compression scheme. We then describe our volume rendering system in section VI. Results and comparisons to existing approaches are provided in section VII, before we conclude the paper with a short discussion and outlook into the future in section VIII.

II. Related Work

A. The Millennium Run

The centrepiece of our evaluations is a cosmological N-body/SPH simulation known as the Millennium Run, presented in detail in the work of Springel et al. [2], [3]. The Millennium Run stands as a representative for the class of gas dynamics simulations where gas expansion and contraction is simulated via SPH. Application areas of such simulations range from chemically reacting flows to molecular dynamics and cosmological structure formation. The Millennium data set consists of 10^{10} particles with varying radii of influence, reaching a spatial dynamic range of 10^5 per dimension in a 3D simulation domain.

SPH has been introduced by Monaghan [4], and since then a vast body of literature related to this field has been published. A comprehensive review of this literature is beyond the scope of this paper.

B. SPH Visualization

The majority of previous approaches for rendering SPH data has been focussed on rendering iso-surfaces in such data, i.e. to render the liquid surface in fluid simulations. Most commonly this is performed by resampling particle quantities to a Cartesian proxy grid [5], [6], and then using iso-surface extraction or direct volume rendering. To avoid the memory consumption of a proxy grid, SPH data can be rendered directly by evaluating the SPH kernels at sampling points along the view rays [7], [8]. An iso-surface extraction technique that works directly on the particle set was introduced in [9].

To also render the fluid body, order-independent splatting of transparent particle sprites has been used [10]–[13]. This approach is fast because it does not require any time-consuming evaluation of the SPH kernels. Yet it can only produce an approximate visual representation because it blends together particle footprints in screen space and does not reconstruct a spatially continuous 3D field in object space. Volume attenuation can only be simulated when the particle set is sorted in every frame, but even in this case the technique is not able to faithfully represent surface-like structures as shown in [1].

To achieve high-quality rendering of large SPH data sets, Fraedrich et al. [1] introduced the perspective grid. This methods employs GPU resampling of particle quantities into a 3D grid [8]. Yet, instead of using a uniform grid discretizing the object space, it uses a perspective grid discretizing the view frustum. Thus, the grid moves with the camera position, requiring only those particles contributing to points in the view frustum to be resampled. Even though a LoD particle hierarchy can reduce the number of particles to be resampled in every frame significantly [13], the per-frame cost of resampling is still too high to allow for interactive rates for simulations as large as the Millennium Run.

C. Volume Rendering

A number of GPU-based volume ray-casting systems for large data sets have been proposed over the last years. These systems employ texture mapping hardware for volume interpolation and make use of the GPU's design for massively parallel workloads to traverse many view rays in parallel [14]. The most recent approaches have shown the efficiency of GPU volume ray-casting when paired with hierarchical octree data structures for LoD rendering and GPU-CPU out-of-core strategies for handling data sets too large to be stored on the GPU [15]–[18].

An important topic related to our method is volumetric data compression using transform coding. The recent survey by Balsa Rodriguez et al. [19] provides a focused treatment of compression techniques used in the context of volume visualization. We use the wavelet-based method described by Treib et al. [20], [21] for the compression of 3D scalar fields, which decodes the data directly on the GPU from a Huffman-encoded wavelet coefficient stream.

III. Overview

Our proposed visualization system for very large SPH data takes as input a set of particles, each given by a position in the 3D simulation domain, a smoothing length specifying the particle's support, and additional scalar attributes like density. In a preprocess, the simulation domain is partitioned adaptively into blocks of different size using octree subdivision, and each block is discretized into a 16^3 grid. As a criterion for steering the subdivision process we use the particles' smoothing lengths: It indicates the size of the structures that are resolved by a single particle, which is up to 100,000$\times$ smaller than the domain extent in the Millennium Run. We subdivide the domain so that the cell size of a grid is equal to the smallest smoothing length of all particles influencing the grid's values.

The preprocess generates an adaptive octree structure, where the leaf nodes represent the blocks at different resolution levels. The particle attributes are then resampled into the grids at these nodes, and the grids at the inner nodes are generated bottom-up via averaging the data values at the grids of the respective child nodes. Upon construction of its parent grid, the grid values are compressed and written to disk. The preprocess is illustrated in figure 2.

GPU ray-casting is then used to visualize the octree data structure. The blocks are traversed on the GPU in front-to-back order. If the compressed representation of the data values of the current block is not residing in CPU memory, it is loaded from disk and cached in RAM. The values are then streamed to the GPU, where a CUDA kernel is executed to decompress the data. Our system leverages GPU texture memory to take advantage of hardware-supported texture filtering. Streaming to the GPU works asynchronously, meaning that the transmission stalls neither the CPU nor the GPU.

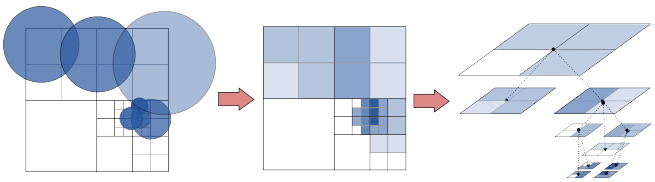

Figure 2. Schematic overview of the preprocessing pipeline: The particle domain is adaptively subdivided based on the smoothing length and distribution of the particles binned to it, until a predefined brick size is reached. Bricks are arranged in an octree data structure, where the data at inner nodes is created by averaging the respective child node data.

IV. Preprocess

The preprocess required to convert the particle data into the adaptive octree hierarchy is specifically tailored to exploit the highly parallel nature of modern CPUs and CPU clusters as well as GPUs. It is divided into two major steps: In a first step, the simulation domain is subdivided into cubic subdomains of fixed spatial extent, with a side length of $\frac{1}{64}$ of the domain in each dimension. We will refer to these subdomains as *superbricks*. All particles are binned on disk into their respective superbricks depending on their position and smoothing length. Particles at superbrick borders are duplicated in this process.

In a second step, for each of the superbricks we create an octree structure—a *subtree*—as explained in detail in section IV-A. The subtrees store an adaptive multiresolution representation of the 3D scalar fields encoded in the particle attributes. Each subtree is then inserted into the final octree at the respective position given by the superbrick's coordinates. The level of the root node of each subtree is determined by the extent of each superbrick—level 6 in our case, with 0 being the coarsest level. All bricks are compressed using a fast wavelet-based GPU compression scheme, which will be detailed in section V.

After each subtree has been built and inserted into the final tree, the coarsest levels are created from the finer levels by recursively merging 8^3 subtree root bricks into a single brick and averaging the corresponding voxel values.

A. Subtree Creation

With the domain divided into reasonably-sized superbricks, each subtree can be created individually, either on a single PC or distributed on a larger cluster. In combination with the LoD metric proposed in this section, the required particle data as well as the output volume data is usually small enough to be processed completely in-core. However, it is worth noting that this process is built upon the same out-of-core octree data structure as our rendering system described in section VI, and thus scales well to systems with limited memory or data sets with very large superbricks.

The subtree creation is based on the observation that the particle distribution and smoothing lengths exhibit very high variance over the entire domain. This allows us to adapt

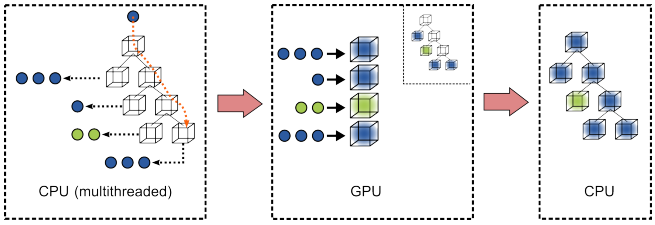

Figure 3. Subtree creation: A particle's position and smoothing length indicate the leaf node at which the particle is stored. The tree is traversed top-down until this leaf node is reached and the particle is stored; missing nodes along the path are generated. The particles of each leaf node are then sampled into the discrete grid, and coarser levels of detail are created bottom-up. Green nodes were created via subdivision but do not contain any particles of correct resolution, and they are deleted finally.

the effective sampling resolution on a per-brick basis: The distance between two grid points is chosen to be less or equal to the smallest smoothing length h, assuring that no particle contributions get "lost". Given the desired grid size of a brick b and the domain extent d, the required octree level l can then be determined as

$$l = \left\lceil \log_2 \left(\frac{d}{h \cdot b} \right) \right\rceil$$

with $l = 0$ being the coarsest level with a grid size of b.

For each particle, the subtree is traversed top-down until the according level is reached. All intermediate nodes are created during traversal. To create the hierarchy, each non-leaf node is always subdivided into eight children. Child nodes which do not contain any particles at the required level are marked as *oversampled* and will not be inserted into the final octree. With each octree leaf node, a list of particles overlapping this node is stored. This whole process is parallelized over the particle data to maximize performance. An illustration is provided in figure 3.

Once the nodes have been created, an empty buffer is allocated for each leaf node to store the grid values at this node, and the particle list for the node is transferred to the GPU. Particle resampling is then performed in a CUDA kernel with one thread associated to each grid vertex—due to the small average number of particles per node, this gathering approach exhibited superior performance to particle scattering as it can be implemented without any write-conflicts. Each thread block first loads a portion of the particle list into shared memory to speed up the process. For each grid vertex, the contribution of each particle is then determined by evaluating a cubic spline kernel based on the particle's smoothing length.

Once all leaves of an octree have been created, coarser levels are generated by recursively merging each node's children, where the value of each voxel is averaged from the respective child voxels. All non-oversampled bricks are then compressed and stored within the final octree on disk.

V. GPU Data Compression

To reduce the total data size and, consequently, the disk bandwidth required during rendering, all bricks are compressed using a discrete wavelet transform (DWT) followed by entropy coding. We employ the GPU for both compression and decompression, as it has been shown to achieve superior throughput compared to CPUs [20]. Our GPU compression scheme for volumetric data is similar in spirit to the one proposed by Treib et al. [21]: First, the input floating point values are quantized with a user-defined quantization step. A 2-level 3D reversible integer DWT using the CDF 5/3 wavelet is then performed on each brick by applying the lifting scheme [22]. More than 2 levels do not significantly improve the compression rate. Finally, the wavelet coefficients are compressed using Huffman coding. The decompression performs the inverse operations in reverse order.

In order to efficiently use the parallel processing power of the GPU, our compression layer processes multiple bricks in parallel (up to 256 in the current implementation). The brick-wise 3D DWT is implemented in a CUDA kernel. For each brick, we launch one thread block consisting of 256 threads which performs the following operations:

- Cooperatively load one brick consisting of 16^3 16-bit integer values (corresponding to 8 KB of memory) from global into shared memory.
- Perform the first-level 16^3 DWT. In this step, the threads are grouped into a 16×16 grid, i.e. each thread processes one row of data along each dimension.
- Shuffle the low-pass and high-pass coefficients from interleaved order (due to the lifting scheme) into subband order: Each thread loads 16 values from shared memory into registers and, after a block sync, stores them to their target positions. In this way, no additional "staging buffer" in shared memory is required.
- Perform the second-level 8^3 DWT, with the threads grouped into an $8\times8\times4$ grid.
- Shuffle the 8^3 coefficients into subband order.
- Store the final result to global memory.

This approach is very efficient because each element is read from and written to global memory exactly once. The decoder writes directly into the volume ray-caster's 3D texture using 3D surface writes to avoid an extra `memcpy`.

When using larger bricks (e.g. 32^3), a brick is too large to fit into shared memory at once. In this case, we fall back to a three-pass algorithm, i.e. we run three CUDA kernels sequentially:

1) Each thread block loads a $32\times32\times1$ slice from global to shared memory, does a DWT in the X and Y dimension, and stores the result back to global memory.
2) Each thread block loads a $32\times1\times32$ slice, does a DWT in the Z dimension, and stores the result back to global memory.

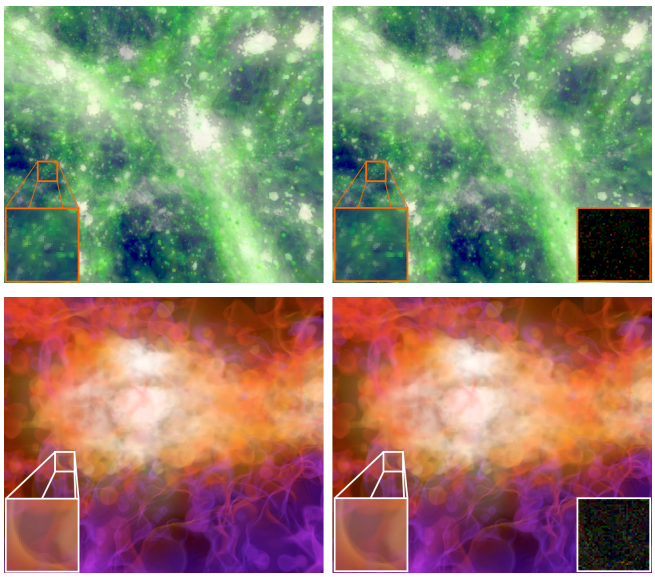

Figure 4. Quality comparison between the uncompressed (left) and compressed (right) data. The difference image is scaled by a factor of 20.

3) Each thread block loads a $16\times16\times16$ block of low-pass coefficients, does a DWT in all 3 dimensions, and stores the result back to global memory.

For the final entropy coding stage, we employ a Huffman coder as in [20]. However, we skip the preceding run-length encoding step, as it does not improve the compression rate in our case.

To justify the use of lossy compression, figure 4 shows images created from compressed and uncompressed data, where the particle quantities are stored as 32-bit floating point values. This coincides with the precision of the input data. As can be seen, the compression introduces no visual artefacts. For bricks of size 16^3, decompression is performed at an average output rate of 2.4 GB/s, yielding a decompression time of 3.18 μs per brick.

VI. Multiresolution Volume Rendering

Rendering is performed using an out-of-core ray-guided ray-caster running on the GPU. For an in-depth description of similar systems, we refer the reader to [17], [15] and [16]. The whole system is outlined in figure 5.

The back-end of our system is an out-of-core octree structure which stores the volume data and the LoD pyramid. All data is organized in *pages* a few megabytes in size to minimize the impact of disk seek times on the data transfer rate. Bricks are grouped into pages during the preprocess in a breadth-first manner, and each page is treated as one entity in terms of disk transfer and main memory caching.

Whenever a brick needs to be rendered, the corresponding page is requested and asynchronously loaded from disk if it is not yet available in the LRU *page cache* in main memory. After loading has completed, it is added to the page cache and all required bricks are inserted into the decompression queue in GPU memory. Whenever this queue is full or rendering of a new frame starts, decompression is performed, and the decompressed bricks are added to an LRU *brick cache*. We call the bricks available in this GPU cache *resident*.

The octree structure of all resident bricks is maintained in GPU memory, where each node contains—if available—a pointer to its eight children, a pointer to the volume data, the scalar range of the contained data and two flags indicating whether volume data and/or children are available on hard disk. This structure is traversed in a single pass on the GPU, where each brick that contains data in the range of a user-defined transfer function is rendered using transfer function preintegration [23] and quadrilinear filtering, using data from the brick's parent.

In contrast to the mentioned previous systems, our preprocess refines only those octants of a brick which require a higher resolution due to the determined per-particle LoD. Always creating all 8 children at once would increase the total number of bricks by more than a factor of 3. Whenever a child node is not available during rendering, we fall back to rendering the corresponding octant of its parent brick. To make this possible, the data manager ensures that a brick's volume data is always uploaded to the GPU before descending further.

The decisions to descend or render are driven by a user-defined desired maximum error in screen space. In addition—as is common when rendering this type of astrophysical data—a focus distance along with a region of interest can be defined to reduce the amount of displayed information. We also employ a variant of β-acceleration [24]: Depending on the currently accumulated opacity α, a ray may decide to not further descend down the hierarchy if the screen space error induced by rendering the current brick is less then $e_s \cdot (1+\alpha \cdot k)$, where e_s is the desired screen space error and k is a user-defined factor ≥ 0. In our experiments, choosing $k = 1.0$ increased the rendering performance by

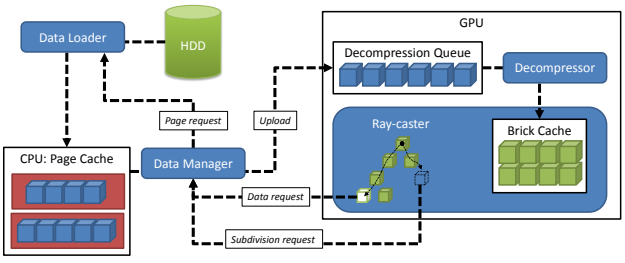

Figure 5. Rendering system overview: Bricks of the octree are fetched asynchronously from the hard drive and cached in main memory. Required bricks are uploaded to the GPU as soon as they are available. Once a fixed number of compressed bricks reside in GPU memory, they are decompressed and cached. The octree is traversed in a single-pass compute shader which signals the missing volume data and child bricks to the CPU.

about 20%, with minimal effect on the output quality.

Due to the asynchronous loading and caching of bricks, brick misses can happen during ray traversal in the following situations:

1) A brick's children need to be rendered, but their nodes have not been inserted into the GPU octree.
2) A brick needs to be rendered, but its data is not resident.

To handle these cases, we follow the procedure suggested in [17]. In the first case, a *subdivision request* is generated and the brick is rendered instead of its children. In the second case, a *data request* is generated, and instead of the requested brick the corresponding octant of its parent is rendered in the current frame. Requests are written to a GPU buffer which provides a slot for each resident brick. These write operations can be performed in a non-atomic way, since in each frame the buffer is first cleared to zero and then set to a constant value for each request. The request buffer is read back to the CPU in every frame. Finally, the CPU updates the LRU caches in CPU and GPU memory as requested, and the process restarts. To further reduce the latency introduced by low disk bandwidth, whenever the data loader is not busy waiting for data, the data in a spherical region [25] is prefetched.

VII. Results

We have evaluated the performance and memory consumption of our Direct3D 11 rendering system and the preprocessing pipeline on a single desktop PC equipped with an Intel Xeon X5560 processor (quad-core, 2.80 GHz), 24 GB of main memory and an NVidia GeForce GTX Titan with 6 GB of video memory. Timings and working set sizes always refer to a viewport of 1200×800 pixels with a desired screen-space pixel error of 1.0, though some images have been cropped for layout reasons. Two data channels were sampled from the particle data: Dark matter density and density-weighted velocity dispersion, both quantized using a quantization step of 0.01 after logarithmic rescaling, thus each channel can be stored using 2 bytes per voxel. Our variant of β-acceleration was enabled with $k = 1.0$. All I/O was performed on a RAID-0 configuration of two hard drives with a maximum throughput of 190 MB/s.

A. Preprocessing

Preprocessing times for one timestep of the Millennium Run simulation are given in table I. The size of each brick was chosen to be 16^3 voxels, including an overlap of one voxel at the borders to ensure correct interpolation between bricks. The timings include all data transfers between main memory and GPU memory.

The selected brick resolution has proven to be a good trade-off between rendering speed and memory requirements: Small bricks result in a better adaption to the local feature sizes. They can thus approximate larger regions with coarser resolutions, greatly reducing the overall memory requirement. On the other hand, they increase the overhead of overlaps—for bricks of resolution 16^3 with one voxel of overlap, only 67% of the voxels contain non-redundant information—and reduce the effectiveness of per-brick compression. They also diminish rendering performance due to the octree traversal overhead and the more incoherent layout of the volume data in GPU memory. Table II compares compressed and uncompressed file sizes as well as frame rendering times for different brick sizes. Rendering performance is given for the screenshot in figure 1 (left). As can be seen, a brick resolution of less than 16^3 voxels even increases the total file size as the effectiveness of the compression decreases drastically.

To show the effectiveness of adaptive subdivision, table III shows statistics for the different octree levels, starting from the "superbrick level" 6 up to the finest level.

B. Rendering

We compare the rendering quality and performance to particle splatting [13] and perspective grid ray-casting [1]. In all comparisons, we will refer to the presented system as multi-resolution direct volume rendering (MRDVR). For MRDVR, we present the rendering times as well as the required GPU memory for the visible uncompressed bricks, which we denote as the *working set size*.

While the existing approaches have been evaluated on an older GeForce GTX 280, experiments have shown a speedup

Table I
Preprocessing times for one timestep of the Millennium Run.

Action	Time (h:m)
Superbrick creation	00:34
Octree creation	04:41
Particle resampling	19:40
Subtree merging	00:44
Compression	07:28
Disk I/O	02:13
Total	35:20

Table II
Influence of different brick sizes on file size and rendering speed. For comparison, the size of the original particle data is 225 GB (for reference, [13] required 198 GB). *Frame* gives the rendering times for figure 1, left.

	File size			
Brick size	Raw (GB)	Comp (GB)	Ratio	Frame (ms)
8^3	1,655.56	540.47	0.33	276
16^3	2,916.85	512.61	0.18	213
32^3	7,967.81	998.34	0.13	168

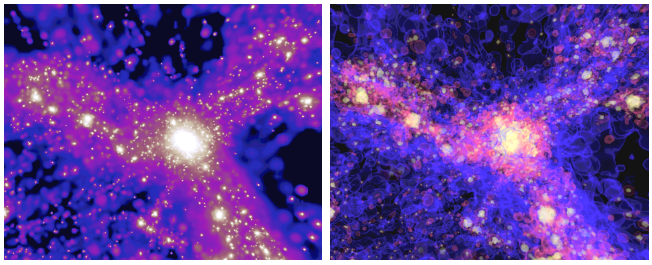

Figure 6. Comparison between unordered splatting (left, 18 ms) and MRDVR (right, 122 ms). Unordered splatting can not reconstruct fine, surface-like structures, and the lack of occlusion hinders depth perception.

of both techniques of a factor of about 3 on the target GPU (GeForce GTX Titan). While the compute and rasterization performance has experienced a massive increase over the last years, the limiting factors have shown to be the texture fill rate and memory bandwidth due to the high amount of overdraw and blending, which have only increased by a factor of 4 and 2, respectively. Thus, the particle splatting approach performs with an average of about 30 ms per frame, while the perspective grid ray-caster can only achieve hardly interactive rendering times of over 3 seconds per frame for this data set.

Figure 6 shows the differences in quality and rendering time between the unordered splatting approach and MRDVR. The dominant structures are visible in both approaches, yet MRDVR allows for much more complex transfer functions.

Figure 7 displays the results with and without β-acceleration. Small features in the background are well preserved while the rendering time exhibits a speed-up of about 20%. The working set size is reduced by 30%.

Finally, we performed a trip through the data set and recorded rendering times, working set sizes and memory transfer sizes. The flight features a wide range of speeds as well as close-up and distant views all across the domain.

Table III
PER-LEVEL STATISTICS. AVERAGE *Children* ARE GIVEN PER INTERNAL (NON-LEAF) NODE. *Oversampled* BRICKS ARE NOT CONTAINED IN THE FINAL OCTREE. *# Particles* IS THE NUMBER OF PARTICLES WHOSE SMOOTHING RADII DICTATE SAMPLING ON THE RESPECTIVE LEVEL.

	# Bricks			
Level	Stored	Oversampled	Children	# Particles
6	262,000	0	7.9	2,855,365,100
7	2,074,646	21,290	5.5	2,797,593,157
8	10,487,783	4,789,657	3.2	2,227,115,066
9	22,747,376	33,640,354	2.3	2,282,715,327
10	27,027,644	65,031,897	2.0	2,105,993,312
11	20,636,515	60,094,086	1.9	1,335,442,527
12	9,015,712	28,574,277	1.9	474,714,085
13	1,208,903	3,953,823	–	45,567,944

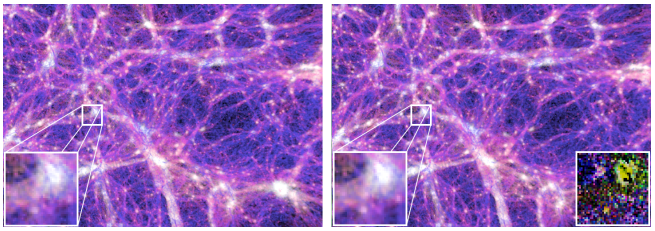

Figure 7. Rendering results without (left, 268 ms) and with (right, 220 ms) β-acceleration. The working set is reduced from 2.44 GB to 1.74 GB. The bottom right shows a difference image, scaled by a factor of 20.

Results can be seen in figure 8. The dominant drops in the working set size are due to rapid viewport changes, causing a large number of GPU cache misses. As coarser levels of detail are being rendered in these cases, the performance improves temporarily at the cost of image quality. However, the missing data is delivered within a few frames.

VIII. CONCLUSION

In this paper we have presented an out-of-core system for the interactive visualization of very large SPH data sets. The key conclusion of our work is twofold: Firstly, by resampling the particle attributes to a regular, yet adaptive hierarchical octree grid and compressing the resampled data using a wavelet-based scheme, only a moderate increase in memory is introduced. Compared to the compressed particle data including a particle LoD hierarchy, a factor of 2 could be demonstrated. Secondly, since any resampling of particle attributes at runtime can be avoided, rendering performance increases significantly. At the same quality, our tests have shown a performance increase of about 10-15$\times$ compared to the perspective grid approach. Compared to order-independent particle splatting, our system is about 5-10$\times$ slower, yet it comes with significantly higher quality and more flexible visualization options like iso-surface rendering or gradient shading.

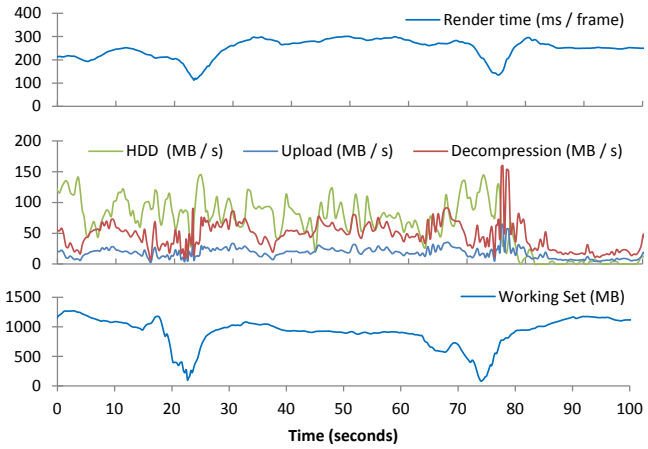

Figure 8. System performance during a flight through the data set.

In the future we will investigate the integration of our desktop system into a remote, client/server-based visualization infrastructure supporting astrophysicists in their explorations. Since our method avoids any time-consuming processing of particles at runtime, it is especially suited for multi-user scenarios, where at the same time images from different perspectives are requested. Since it is not possible in general to keep the working set of more than one user on the GPU, the working sets of all users have to be swapped in and out of the GPU periodically. The memory and, thus, bandwidth-aware design of our system accommodates an efficient realization of these operations.

ACKNOWLEDGMENT

We would like to thank Volker Springel from the Max Planck Society in Garching for his support with the data set. This publication is based on work supported by Award No. UK- C0020, made by King Abdullah University of Science and Technology (KAUST).

REFERENCES

[1] R. Fraedrich, S. Auer, and R. Westermann, "Efficient high-quality volume rendering of SPH data," *IEEE Trans. Vis. Comput. Graphics*, vol. 16, no. 6, pp. 1533–1540, 2010.

[2] V. Springel, "The cosmological simulation code GADGET-2," *Mon. Not. Roy. Astron. Soc.*, vol. 364, p. 1105, 2005.

[3] V. Springel, S. D. M. White, A. Jenkins, C. S. Frenk, N. Yoshida, L. Gao, J. Navarro, R. Thacker, D. Croton, J. Helly, J. A. Peacock, S. Cole, P. Thomas, H. Couchman, A. Evrard, J. Colberg, and F. Pearce, "Simulating the joint evolution of quasars, galaxies and their large-scale distribution," *Nature*, vol. 435, pp. 629–636, 2005.

[4] J. J. Monaghan, "Smoothed particle hydrodynamics," *Rep. Prog. Phys.*, vol. 68, pp. 1703–1758, 2005.

[5] D. Cha, S. Son, and I. Ihm, "GPU-assisted high quality particle rendering," *Computer Graphics Forum*, vol. 28, no. 4, pp. 1247–1255, 2009.

[6] P. A. Navrátil, J. L. Johnson, and V. Bromm, "Visualization of cosmological particle-based datasets," *IEEE Trans. Vis. Comput. Graphics*, vol. 13, no. 6, pp. 1712–1718, 2007.

[7] Y. Kanamori, Z. Szego, and T. Nishita, "GPU-based fast ray casting for a large number of metaballs," *Computer Graphics Forum*, vol. 27, no. 2, pp. 351–360, 2008.

[8] R. Yasuda, T. Harada, and Y. Kawaguchi, "Fast rendering of particle-based fluid by utilizing simulation data," in *Proc. Eurographics 2009 - Short Papers*, 2009, pp. 61–64.

[9] I. D. Rosenberg and K. Birdwell, "Real-time particle isosurface extraction," in *Proc. ACM SIGGRAPH Symp. Interactive 3D Graphics and Games (I3D)*, 2008, pp. 35–43.

[10] Y. Li, C.-W. Fu, and A. Hanson, "Scalable WIM: Effective exploration in large-scale astrophysical environments," *IEEE Trans. Vis. Comput. Graphics*, vol. 12, no. 5, pp. 1005–1012, 2006.

[11] M. Hopf and T. Ertl, "Hierarchical splatting of scattered data," in *Proc. IEEE Visualization*, 2003, pp. 443–440.

[12] M. Hopf, M. Luttenberger, and T. Ertl, "Hierarchical splatting of scattered 4D data," *IEEE Computer Graphics and Applications*, vol. 24, no. 4, pp. 64–72, 2004.

[13] R. Fraedrich, J. Schneider, and R. Westermann, "Exploring the "Millennium Run" - scalable rendering of large-scale cosmological datasets," *IEEE Trans. Vis. Comput. Graphics*, vol. 15, no. 6, pp. 1251–1258, 2009.

[14] J. Krüger and R. Westermann, "Acceleration techniques for GPU-based volume rendering," in *Proc. IEEE Visualization*, 2003, pp. 38–43.

[15] E. Gobbetti, F. Marton, and J. Iglesias Guitián, "A single-pass GPU ray casting framework for interactive out-of-core rendering of massive volumetric datasets," *The Visual Computer*, vol. 24, no. 7–9, pp. 797–806, 2008.

[16] J. A. Iglesias Guitián, E. Gobbetti, and F. Marton, "View-dependent exploration of massive volumetric models on large-scale light field displays," *The Visual Computer*, vol. 26, no. 6–8, pp. 1037–1047, 2010.

[17] C. Crassin, "GigaVoxels: A voxel-based rendering pipeline for efficient exploration of large and detailed scenes," Ph.D. dissertation, Université de Grenoble, July 2011.

[18] M. Hadwiger, J. Beyer, W.-K. Jeong, and H. Pfister, "Interactive volume exploration of petascale microscopy data streams using a visualization-driven virtual memory approach," *IEEE Trans. Vis. Comput. Graphics*, vol. 18, no. 12, pp. 2285–2294, 2012.

[19] M. Balsa Rodriguez, E. Gobbetti, J. Iglesias Guitián, M. Makhinya, F. Marton, R. Pajarola, and S. Suter, "A survey of compressed GPU-based direct volume rendering," in *Eurographics 2013 - STARs*, 2013, pp. 117–136.

[20] M. Treib, F. Reichl, S. Auer, and R. Westermann, "Interactive editing of gigasample terrain fields," *Computer Graphics Forum*, vol. 31, no. 2, pp. 383–392, 2012.

[21] M. Treib, K. Bürger, F. Reichl, C. Meneveau, A. Szalay, and R. Westermann, "Turbulence visualization at the terascale on desktop PCs," *IEEE Trans. Vis. Comput. Graphics*, vol. 18, no. 12, pp. 2169–2177, 2012.

[22] W. Sweldens, "The lifting scheme: A construction of second generation wavelets," *SIAM J. Math. Anal.*, vol. 29, no. 2, pp. 511–546, Mar. 1998.

[23] K. Engel, M. Kraus, and T. Ertl, "High-quality pre-integrated volume rendering using hardware-accelerated pixel shading," in *Proc. ACM SIGGRAPH/Eurographics Workshop on Graphics Hardware*, 2001, pp. 9–16.

[24] J. Danskin and P. Hanrahan, "Fast algorithms for volume ray tracing," in *Proc. Volume Visualization*, 1992, pp. 91–98.

[25] C.-M. Ng, C.-T. Nguyen, D.-N. Tran, T.-S. Tan, and S.-W. Yeow, "Analyzing pre-fetching in large-scale visual simulation," in *Proc. Computer Graphics International (CGI)*, 2005, pp. 100–107.