Exploring the Millennium Run - Scalable Rendering of Large-Scale Cosmological Datasets

Roland Fraedrich, Jens Schneider, and Rüdiger Westermann

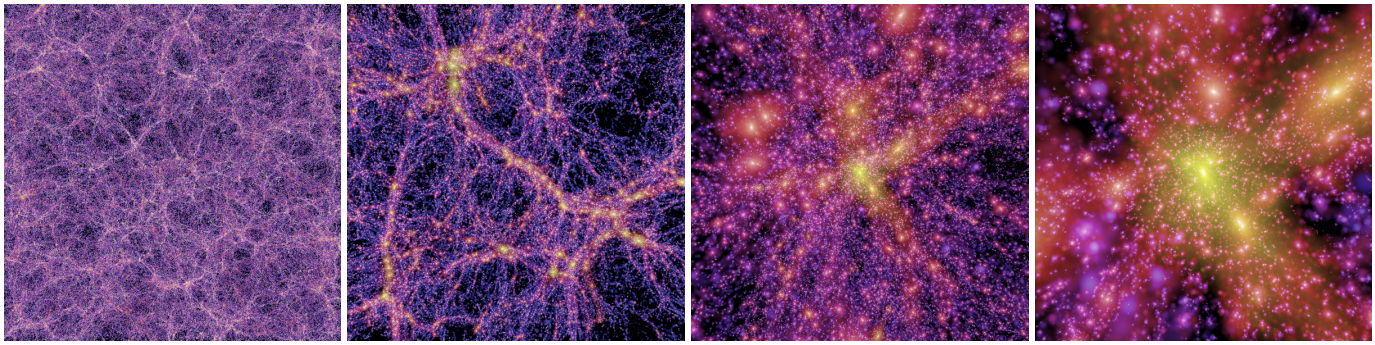

Fig. 1. Visualizations of the Millennium Simulation with more than 10 billion particles at different scales and a screen space error below one pixel. On a 1200x800 viewport the average frame rate is 11 fps.

Abstract—In this paper we investigate scalability limitations in the visualization of large-scale particle-based cosmological simulations, and we present methods to reduce these limitations on current PC architectures. To minimize the amount of data to be streamed from disk to the graphics subsystem, we propose a visually continuous level-of-detail (LOD) particle representation based on a hierarchical quantization scheme for particle coordinates and rules for generating coarse particle distributions. Given the maximal world space error per level, our LOD selection technique guarantees a sub-pixel screen space error during rendering. A brick-based page-tree allows to further reduce the number of disk seek operations to be performed. Additional particle quantities like density, velocity dispersion, and radius are compressed at no visible loss using vector quantization of logarithmically encoded floating point values. By fine-grain view-frustum culling and presence acceleration in a geometry shader the required geometry throughput on the GPU can be significantly reduced. We validate the quality and scalability of our method by presenting visualizations of a particle-based cosmological dark-matter simulation exceeding 10 billion elements.

Index Terms—Particle Visualization, Scalability, Cosmology.

1 Introduction

Particle-based cosmological simulations play an eminent role in reproducing the large-scale structure of the Universe. Today, numerical simulations of the dark-matter distribution using as much as billions of particles have been carried out, and the results of these simulations have to be analyzed for a better understanding of the structure formation process in the Universe.

One of the most popular simulations is the Millennium Run, where more than 10 billion particles have been used to trace the evolution of the matter distribution in a cubic region of the Universe with over 2 billion light-years on a side, resulting in terabytes of stored output. The simulation itself ran 28 machine days at 0.2 Tflops using 1 Tbyte RAM. By comparing the simulation results to large observational surveys, the physical processes underlying the buildup of real galaxies and black holes can be clarified. Especially by analyzing the substructure in simulated halos and subhalos, the hope is to prove or contradict recent claims and hypotheses on the matter distribution in the Universe.

- *Roland Fraedrich (E-mail: fraedrich@tum.de), Jens Schneider (E-mail: jens.schneider@tum.de), and Rüdiger Westermann (E-mail: westermann@tum.de) are with the Computer Graphics and Visualization Group, Technische Universität at München.*

Manuscript received 31 March 2009; accepted 27 July 2009; posted online 11 October 2009; mailed on 5 October 2009.
For information on obtaining reprints of this article, please send email to: tvcg@computer.org .

By today, for the largest available cosmological simulations an interactive visual analysis is not feasible, and the exploration is mainly performed via database queries, data mining techniques, and visualizations of vastly reduced subsets of the full dataset. On the other hand, it can be perceived that an even stronger awareness of the importance of interactive visualization techniques for data exploration is growing in the community. One important reason is that especially in cosmological data it is not yet completely understood what the relevant scientific information in the vast amount of stored output is. This makes it difficult to apply fully automatic analysis techniques, which typically need a precise specification of the structures to search for. As a consequence, the community increasingly tries to put experts and their capabilities into the center of the exploration process, to support them by visualization tools that can effectively exploit the humans' perceptual and cognitive abilities, and to let them interactively guide the search process.

To achieve interactive visualizations of cosmological particle simulations as large as the Millennium Run, we have to consider scalability with respect to the size of the data in the whole visualization process. From the massive amount of particles to be visualized it becomes clear that the challenge in cosmological visualizations today is to maintain interactive frame rates for datasets that vastly exceed the available main memory. Since the number of primitives to be sent to the GPU for rendering is significantly increasing, too, the visualization performance is strongly limited by bandwidth restrictions. In addition, even on recent high-end graphics accelerators (GPU) geometry throughput and rasterization load induced by particle rendering considerably limit the performance.

2 CONTRIBUTION

The primary focus of this paper is to address scalability limitations in the visualization of very large cosmological particle simulations and to investigate how far one can reach on recent PC architectures by reducing these limitations. The methods we propose are based on a hierarchical encoding of particle positions and attributes into a tree data structure. This data structure stores partitions of the underlying domain at ever finer resolution, with the sub-domains in each partition being quantized uniformly using 8 bit per coordinate component. The partitions are recursively refined until the spatial resolution of the simulation is reached.

To reduce the number of particles at each resolution level, particles falling into the same quantization bin are merged if the rendering of the resulting particle at this level does not introduce a visible error. By this means, we build a level-of-detail representation that allows reducing aliasing artifacts as well as the number of rendered primitives. In combination with a page-tree that stores a number of adjacent subdomains in one contiguous memory block, memory bandwidth and the number of disk seek operations can be reduced considerably.

Besides their coordinate, particles of cosmological simulations carry floating point attributes such as mass density, velocity dispersion, and radius, which significantly increase the data to be stored and transferred. To address this problem, we employ a vector quantization scheme after the attributes have been scaled logarithmically. Since the quantized data can be decoded efficiently on the GPU, we can trade CPU-GPU data transfer for GPU processing, resulting in three times faster streaming of large particle sets from disk into GPU memory.

Regardless of these efforts, geometry throughput and rasterization of particle splats on the GPU still remain critical bottlenecks for interactive visualizations. To further reduce this problem we propose techniques to discard particles from the rendering process as early as possible by fine-grain view-frustum culling and presence acceleration. The latter approach takes advantage of the fact that it is not necessary to render many particles in regions with small contributions to the integrals over physical quantities. These problems as well as ideas for further optimizations will also be addressed in the discussion at the end of this paper.

The remainder of this paper is organized as follows: In the next section we review previous work that is related to ours. An overview of the Millennium Simulation and the visualization technique we employ are given in Sections 4 and 5. Subsequently, we present the components of the proposed scalable rendering pipeline and discuss our strategies for data representation (Section 6), memory management (Section 7), and rendering (Section 8). In Section 9 we analyze the different components of our method and the framework as a whole. We conclude the paper with an outline of future research in this field.

3 RELATED WORK

Today, interactive data visualization is established as a key technology for the exploration of large and complex-structured datasets as they arise in almost all areas of science and engineering. There is a vast body of literature related to visual data analysis that we will not attempt to overview here, however [10, 16] discuss many related issues and provide useful references on these subjects.

Recent work on the visualization of astronomical data include the development of infrastructures for high-performance computing environments [2, 11, 19], volume rendering techniques for the visualization of star formation and planetary nebulae [14, 17], methods for visualizing uncertainty in large-scale astrophysical environments [12], and the comparison of different simulations by comparative visualization [1].

Other approaches underline the importance of user-tailored visualization and interaction techniques in the field of astrophysics, like the development of scalable WIM (world-in-miniature) user interfaces that facilitate travel and wayfinding at large spatial scales [13], or user-controlled interaction that supports multiple abstraction views of galaxy clusters and the selection of structures in spatial sub-domains [15, 21].

For particle data from cosmological simulations, software rendering [6, 20, 7] is still most commonly used for visual data analysis. Efficient multi-resolution point splatting techniques based on spatial subdivision schemes have been proposed in [8, 9, 26], but the size of today's simulations demands scalable out-of-core rendering approaches, which support these visualization techniques by efficient data management and transfer [4]. To cope with this ever increasing amount of data to be visualized, parallel visualization systems [2, 11] and infrastructures for visual data mining in distributed data sources [3] have been proposed so far, including mechanisms for data handling, querying, and interaction.

4 THE MILLENNIUM RUN

The goal of the Millennium Simulation Project[1] is to investigate the evolution of the Universe based on a Lambda-Cold Dark Matter (ΛCDM) model of the accelerating expansion of the universe. For this purpose, the growth of dark matter structure was traced from redshift $z = 127$ to the present by using a N-body/SPH-simulation in which the collisionless dark matter fluid was represented by a set of discrete point particles. With a total number of $2160^3 \approx 10^{10}$ particles, distributed in a periodic box of 2.23 billion lightyears, this is one of the largest particle-based simulations ever performed. For a detailed description of the numerical code used to compute the Millennium N-body simulation as well as the scientific results of the Millennium Project let us refer to the work by Springel et al. [23, 25].

In order to investigate the time-dependent dark matter distribution, the intermediate results of the simulation have been stored in 64 snapshots. Besides its own 3D floating point coordinate, each particle in a single snapshot carries additional physical quantities such as an adaptive smoothing radius, density, and velocity dispersion, each of which is encoded in one floating point value. For a single snapshot this results in a total amount of memory of 225 GB.

5 COSMOLOGICAL VISUALIZATION

The visualization technique we propose aims at showing the dark matter distribution in 3D space. It follows a visualization approach for cosmological particle data that is commonly used for offline visualizations of the Millennium Simulation [24]. In this approach, the squared dark matter density is integrated along the lines of sight and scaled logarithmically to increase the contrast in pixel brightness. The dark matter velocity dispersion, i.e., the spread of velocities of stars or galaxies in a cluster, is weighted by the squared dark matter density and averaged along the lines of sight. It is encoded into the color hue using a particular transfer function to emphasize regions of high dispersion.

It is clear that due to the nature of the SPH-model underlying the Millennium Simulation, along the lines of sight the physical quantities have to be reconstructed locally at the sampling points using an SPH kernel interpolation scheme. Such a scheme, however, requires access to all particles within the kernel support, meaning that extensive search operations supported by acceleration structures like kD-tress have to be performed. Unfortunately, such a resampling scheme can by no means be fast enough to be used in an interactive application.

The common approximation is thus to splat each particle separately onto the image plane, and to accumulate the quantities on a per-pixel basis before they are scaled appropriately. Figure 1 shows some images of the Millennium Simulation at different scales, which have been generated by this approach. Even though this approach is not physically correct, it has been shown to produce visualizations that are very similar to those generated by the SPH-based approach and allow for an equally good analysis of the data. In this approach, in analogy to the behavior of light, the total density of a single particle falls off as the square of its distance to the viewer. An additional fall-off is usually applied to fade-out the density field beyond some depth of interest.

The major requirement on a cosmological visualization technique capable of dealing with data as large as the Millennium Simulation is scalability throughout the entire visualization pipeline, including the streaming of the data from the hard disk to the graphics processor. The

[1] http://www.mpa-garching.mpg.de/galform/virgo/millennium/

major constraints in this pipeline are memory and bandwidth limitations, disk access latency, and the geometry and rasterization throughput on current graphics hardware. According to these constraints, an efficient cosmological visualization technique has to be built on the following components:

- A data representation that minimizes the amount of data to be transferred and rendered while preserving visual quality.
- An efficient out-of-core and in-core data management to quickly select and access the data.
- A rendering scheme that can effectively exploit the capabilities of current graphics accelerators to enable interactive visual data exploration.

6 Data Representation

Our visualization technique builds on a multi-resolution representation of large particle sets in combination with a data-specific compression scheme. The multi-resolution representation is stored in an adaptive octree data structure. Thus, accessing the data that is stored in this structure exhibits $O(\log_2(N))$ complexity (with N being the grid size corresponding to the maximum refinement level). In the current application we refine up to the spatial resolution of the simulation, i.e., to an effective grid size of $128K^3$. The sub-spaces represented by each octree node are discretized into regular grids of size 8^3.

6.1 Multi-Resolution Hierarchy

The multi-resolution hierarchy is constructed bottom-up in a preprocess, i.e., the original particle data is stored in the nodes at the finest level of the tree. At the coarser hierarchy levels, particles from the next finer level are either merged, copied, or omitted depending on their radius of influence and their density. The particular LOD-construction method we employ is described in Section 6.3. Figure 2 shows three nodes of the multi-resolution hierarchy, each of which represents a part of the region encoded in the previous node at increasing resolution.

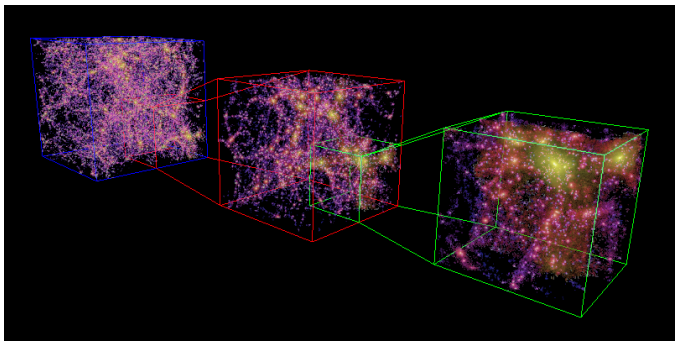

Fig. 2. Three nodes in the multi-resolution particle hierarchy with increasing resolution from from left to right.

As the entire dataset has a size of 225 GB, we split the spatial domain into 8^3 blocks to fit the corresponding data into main memory. For each block we compute the respective subtree of the octree in parallel. Upon completion of this process all subtrees are merged into one octree, which is then re-organized into a page tree to accommodate faster data access. The construction of this page tree is discussed in Section 7.1.

6.2 LOD-Selection

Given the size q of the quantization cells at a particular LOD, the maximum world space error that is introduced by quantizing particle coordinates to cell centers is $\delta = \sqrt{3} \cdot q/2$. During rendering a strategy is required to select those octree nodes that result in a sub-pixel error on the screen, and, thus, yield a visually continuous LOD rendering that is lossless wrt. the current image resolution.

Given the world space error δ, the vertical screen resolution res_y, and the vertical field of view $fovy$, we can determine the minimum distance $d_{\min}$ to the viewer as of which a node can be rendered before it has to be refined:

$$d_{\min} = \frac{\delta \cdot res_y}{2 \cdot \tan(fovy/2)} \cdot \tau.$$

Here, $\tau < 1$ is the maximum allowed screen-space error in pixels. To maintain this error during rendering, each node that is closer to the viewer than $d_{\min}$ has to be replaced by its 8 children.

6.3 Particle Thinning

The LOD-selection strategy tries to always render octree nodes whose projection is approximately equal or less than one pixel. Taking this into account, particles can be classified with respect to the number of pixels they would cover if they were rendered at a particular level: *points* cover at most one pixel and *sprites* cover more than one pixel.

Since all *points* in the same quantization bin will fall into the same pixel on the screen, they can be merged to one single *point* (see Figure 3). To determine the dark matter density and velocity dispersion of this point wrt. to the individual contributions, an density-weighted filtering is computed in a pre-process.

Sprites in one cell, on the other hand, can only be merged without visual loss if the differences of their radii h_i and h_j is not perceivable during rendering (see Figure 3). For this purpose, we use the fusion criterion $|h_i - h_j|/q < \varepsilon$, where ε is a sufficiently small number. If this condition is not fulfilled, the particles are stored separately at the respective level. Note that due to the local coherence of smoothing lengths in a SPH simulation, almost identical radii within one cell are very likely to occur, particularly at finer resolution levels.

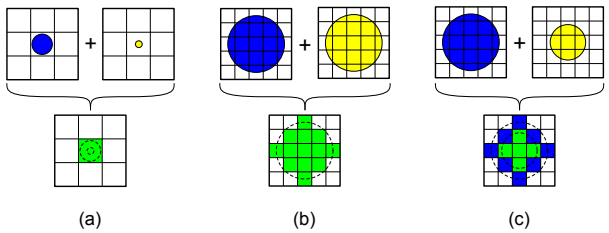

Fig. 3. Resolution dependent particle merging: (a) Particles with a radius less than the cell size are merged into one single point primitive. (b) Particles covering more than one pixel can only be merged if their radii are equal. (c) Otherwise, they can not be combined in a meaningful way.

In order to further decrease the number of particles per node, we introduce a rule for discarding particles depending on their density contribution. Based on $d_{\min}$ and the quadratic density fall-off, the projected squared dark-matter density ρ_i^2 of a particle on the screen can be computed. Together with the minimum density for color coding $\rho_{\min}$, we discard a particle if $\rho_i^2 < \rho_{\min}^2/n$. Here, n can be interpreted as the maximum number of discarded particles whose sum of squared densities is still smaller than $\rho_{\min}$. These particles do not have any influence on the final image.

The easiest way to find a reasonable value for n is to simulate the criterion on the GPU and to experiment with different values. As an objective criterion, the scene can be rendered with and without the threshold in order to compute the image difference. This procedure can be repeated for a certain number of different views or while flying through the dataset until a good threshold is found.

6.4 Attribute Compression

Besides their coordinates, the particles stored in the multi-resolution representation carry floating point attributes such as mass density, velocity dispersion, and radius. Due to the nature of the simulation these properties are stochastically dependent. Therefore, to reduce the memory required to store these attributes we employ a vector quantization scheme after the attributes have been scaled non-linearly.

As can be seen in the histogram in Figure 4, all three parameters have a high dynamic range. A standard vector quantizer, however, is designed to minimize the sum of squared errors. This is problematic because the same absolute distance between each data point and its respective quantization point is thus deemed acceptable by the vector quantizer.

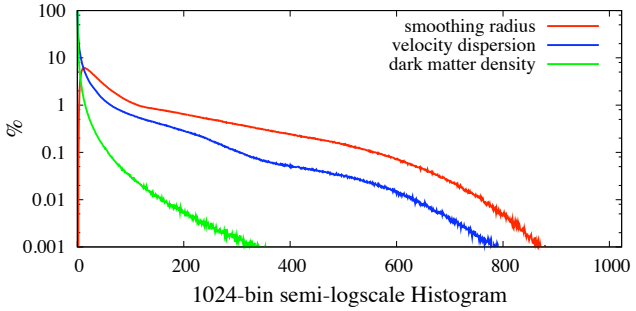

Fig. 4. Histogram of the three particle quantities. For each quantity, we split the range of values in 1024 equal intervals and count the number of occurences of a parameter value in each interval. Full ranges are $[6.73 \cdot 10^{-47}, 2.11]$ (smoothing radius), $[8.80 \cdot 10^{-2}, 4.84 \cdot 10^{8}]$ (dark matter density) and $[2.26, 1.46 \cdot 10^{7}]$ (velocity dispersion).

Especially with regard to the logarithmic color coding of squared dark matter density and velocitiy dispersion during rendering, it is a much better choice for the quantizer design to minimize the relative error, because large absolute errors are not acceptable for small values. From the histogram it is apparent that quantizing logarithms of values will drastically decrease relative errors, albeit at the cost of increasing the sum of squared errors. By logarithmizing (base 2) each component of the 3D data points prior to quantization we can reduce the maximum relative error from $3.63 \cdot 10^{39}$% to about 540% (average relative error from $3.63 \cdot 10^{39}$% to 13%) for an 8 Bit per 3D vector quantization. Note that this does not affect the coding efficiency, since it is sufficient to exponentiate the codebook entries prior to decoding. Also note that changing the base of the logarithm allows a further trade-off between minimization of squared errors and relative errors.

Figure 5 presents the resulting distortion induced by vector quantization using codebooks of varying size. As will be shown in the results, by using an 8 bit index with an average relative error of 13% we can already generate images which can hardly be distinguished from those that have been generated using the original data. By increasing the bit depth to 12 or 16 bit, the error can be reduced to 5% or 2% respectively.

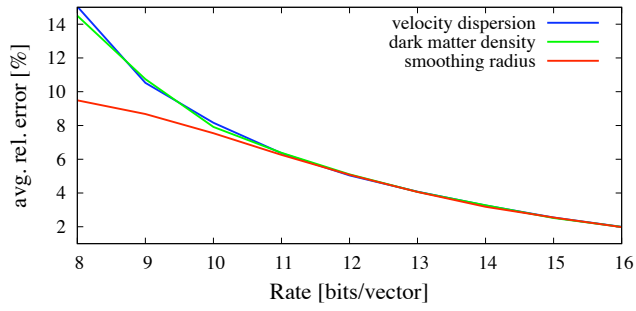

Fig. 5. Average relative error introduced by vector quantization of particle quantities with varying bit depth.

Since the quantized data can be decoded efficiently on the GPU by means of texture lookups, we can trade CPU-GPU data transfer for GPU processing, resulting in three times faster streaming. For details of the vector quantization we refer to Schneider et al. [22].

7 Memory Management

In every frame, the octree is traversed in a top-down fashion on the CPU to determine the nodes that have to be rendered. Of these, those nodes that are not residing in main memory have to be read from disk. During tree traversal, the minimum distance criterion (see Section 6.2) is used to determine the LOD for the current view, and once a node has been determined the traversal in the current sub-tree can be stopped. In addition, view frustum culling is applied to discard those subtrees that are not visible.

Even though frame-to-frame coherence significantly reduces the amount of data to be read from disk, for very large datasets and high-resolution display systems there is still a vast amount of data that has to be streamed from disk to CPU memory in an interactive visual exploration session. In the following we describe a number of mechanisms to keep the impact of data streaming on the frame rate as minimal as possible.

7.1 Disk Transfer

To reduce the number of disk seek operations and thus to increase the data throughput from disk to main memory, in each octree level the nodes having the same parent are packed together and stored as a single data block on disk. We refer to these collections as *pages* and treat them as one entity in terms of storage and transfer. If a page size is below a given minimum size, additional nodes below those stored in the page are added in a breadth-first manner. Since a page typically contains more data than is required in a particular view, the minimum page size has to be chosen such as to yield a good balance between performance and main memory usage. In our experiments, a page size of 3 MB turned out to be a good trade-off. Figure 6 illustrates the relation between a tree data structure and pages in this data structure. As can be seen, the pages itself represent a tree data structure, which we refer to as *page tree*.

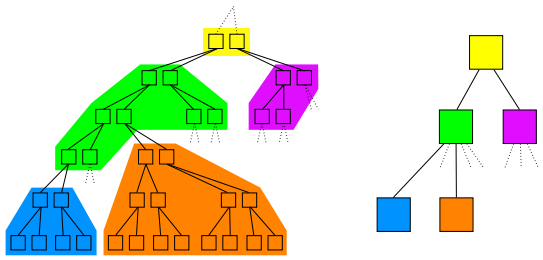

Fig. 6. Illustration of the relation between a tree data structure (left) and a page tree (right), which stores a number of nodes belonging to the same parent node.

The page tree is stored on disk and dynamically loaded into main memory during runtime. To determine whether the data of a page is required for rendering, it is sufficient to check the distance criterion for the parent of the page as it has a smaller d_{min} than all of its descendants. According to the movements of the viewer, the page tree is dynamically built top-down and destroyed bottom-up within main memory, such that a maximum amount of data can be re-used in every frame. Simultaneously, the octree is built and destroyed by adding and removing the nodes contained in these pages. Note that nodes only store pointer to the respective data blocks in the page tree, so that the data is not stored twice in main memory.

7.2 Pre-Fetching and Caching

For hiding disk access latency, we use asynchronous I/O combined with a pre-fetching scheme to load pages that might be required in upcoming frames. We load all nodes having a distance to the camera $d_{\text{cam}} \leq d_{\text{min}} + d_{\text{prefetch}}$, resulting in a sphere-shaped pre-fetching region that supports arbitrary rotations. As has been shown by Ng et al. [18], although the use of a sphere-shaped region increases the overall memory requirement, when moving along the line of sight the amount of data to be read from disk is only marginally increased.

The pre-fetching scheme loads every page that might be required after a camera translation of length d_{prefetch}. In order to hide disk access latency, this distance is chosen such that the time for loading the largest page is less than the time for moving the camera to the position where the node is required for rendering.

When flying through the dataset, there will eventually be pages, which are not required in main memory anymore because the corresponding pages have fallen outside the pre-fetching region. However, as long as enough main memory is available we do not remove the data but move it into a LRU cache. Hence, a page can be reused without any additional cost until the memory is required by another page.

8 Rendering

To render the particles stored in the set of selected octree nodes, they are copied into vertex array buffers on the GPU. These buffers are then rendered as described below. To optimize the resource management on the GPU, we avoid time-consuming creation and deletion of resources per node by using buddy memory allocation. This means that a set of vertex buffers of a size that it is sufficient for every node is allocated, and these buffers are split recursively into equally sized pieces if less memory is needed. If the data in refined blocks is not used anymore, the blocks are merged again to be available for larger vertex arrays. Like the LRU caching scheme used on the CPU, the data corresponding to nodes that are not required anymore are kept in graphics memory until they have to be paged out due to resource shortage.

The rendering approach is realized in two passes. In the first pass, the particles are projected onto the screen in order to accumulate the squared dark matter density and the weighted velocity dispersion along the lines of sight. The result is stored in a render target that is bound as a texture resource in the second pass. In this pass, the pixels final colors are computed by using the accumulated values as texture coordinates into a pre-computed 2D color transfer function.

Due to our multi-resolution representation, the particle coordinates of a node are encoded according to their discretization within the corresponding sub-domain. By using the size of the node as scale factor and its position as translation vector, we can decode them into spatial coordinates. The particle attributes are looked up from the quantization codebook.

8.1 Particle Projection

The particle projection onto the screen is performed by splatting. Particles that are classified as *points* are simply rendered as point primitives. For *sprites*, we use a geometry shader to expand each particle to a screen aligned face that has to be large enough to cover the projected area of the particle's smoothing kernel. The respective size is computed as $r = h/\sqrt{1-h^2/d^2}$, where h is the smoothing radius and d the distance to the camera (see Figure 7).

Via texture coordinates the relative distance of each vertex of the face to the particle center is stored as vertex attribute, which is interpolated on a per-fragment basis during rasterization. In a pixel shader, we have to perspectively correct the interpolated distance r' in order to retrieve the minimal distance $s = r'/\sqrt{1+r'^2/d^2}$ of the respective line of sight to the particle center (see Figure 7). Now, s can be either used for computing the line integral through the smoothing kernel on the fly or to look-up the pre-computed integral value from a one-dimensional texture map.

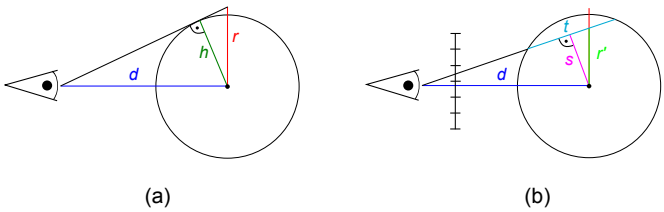

Fig. 7. Perspective correction for line of sight integration through a spherical particle. (a) Instead of using the smoothing length h as splat extend, the projective extent r at the distance d of the particle center is computed first. (b) During fragment shading the closest distance s from r' and d (given as fragment attributes) to the line of sight is computed. s is then used to determine the line integral through the particle.

8.2 Performance Issues on the GPU

Regardless of our efforts to minimize the number of rendered particles, geometry throughput and rasterization of particle splats on the GPU still remain critical bottlenecks for interactive visualizations. We further reduce these problem by discarding particles from the rendering process as early as possible in order to forgo the unnecessary expanding of splats and to reduce the number of primitives that are further processed and rasterized.

For this purpose we employ fine-grain view-frustum culling and in addition, we reapply presence acceleration that has been introduced in Section 6.3 on a per-particle basis. The impact of the latter technique is especially profitable, since we fade-out the densities at the end of our viewing distance. Thus, there is usually a significant number of particles that drop below the density threshold and can therefore be discarded.

Besides the number of particles, the choice of the proxy geometry for rendering point sprites has a considerable impact on the rendering performance as well. Using equilateral triangles instead of quads cuts the number of primitives into halves and significantly increases the geometry throughput, even though the area for rasterization is enlarged by a factor of $3\sqrt{3}/4$.

Finally, rendering multiple fragments to the same screen location in a rapid succession leads to a considerable lack of performance. To minimize this effect, the particle data should be shuffled to create a random order for rendering the particles.

9 Results and Analysis

We have implemented our visualization framework in C++ using DirectX 10 and HLSL for rendering. Our experiments have been carried out on a standard PC equipped with an Intel Core 2 Quad Q9450 2.66 GHz processor, 4 GB of RAM, and a NVIDIA GeForce GTX 280 with 1024 MB video memory and PCI Express 2.0 x16. The data was stored on a striped RAID 0 consisting of two ATA Hitachi HDS72101 hard disks. In all of our experiments we have quantized the particle attributes using 8 bit per component, and we have selected a maximum allowed screen space error of 0.8 pixels.

9.1 Data Representation

The original dataset has a total size of 225 GB, including particle density, radius of influence, and velocity dispersion. The generation of the multi-resolution hierarchy, the quantization of particle attributes, and the creation of the page tree took roughly 5:20 hours. In comparison to the simulation itself, which took 28 machine days on a supercomputer, and the fact that rendering movies of the dataset in software takes several hours as well, this duration is rather small.

The final octree has 10 LODs, with the coordinates of the particles in one node being quantized uniformly using 8 bits per component. Thus, at the finest LOD an uniform spatial resolution of 128K^3 is achieved, which is even slightly higher than the resolution (10^5) at which the simulation has been performed.

To validate our visualization approach with respect to speed and quality we have created two datasets, the first one using 8 bits to quantize particle quantities and the other one using 16 bits. This results in an overall compression rate of 6 : 1 and 4.8 : 1, respectively, for the particle data. The respective page-trees are 159 GB and 198 GB in size.

Table 1 shows detailed statistics for the multi-resolution particle hierarchy. Level 0 represents the finest LOD and contains all particles of the simulation. With increasing level, the number of particles are successively reduced due to particle thinning as proposed in Section 6.3. At finer LODs the effect is rather minor, which is not surprising due to the conservative approach we use. With increasing LOD, however, the reduction between two consecutive levels increases up to 91%.

Table 1. Statistics for each level of the particle multi-resolution hierarchy. *Merged* shows the number of merged particles from level $n-1$ to level n. *Skipped* gives the number of particles which have been skipped at level n due to presence acceleration, but which may contribute to a merged particle at level $n+1$. *Sprites* and *Points* show the number of sprites and points, respectively, with their sum being listed in *Particles*. *Average* gives the average number of particles per node.

Level	Merged	Skipped	Sprites	Points	Particles	Average
9	1 149 203 319	196 713 663	0	5 443 448	5 443 448	5 443 448
8	1 389 651 438	1 335 618 568	0	15 741 862	15 741 862	1 967 732
7	1 270 473 711	2 700 351 786	6 324 410	34 335 672	40 660 082	635 313
6	1 231 730 796	3 551 487 161	403 898 837	56 099 581	459 998 418	898 434
5	1 278 797 785	3 197 259 628	1 992 911 560	53 045 187	2 045 956 747	499 501
4	1 284 656 859	2 135 299 112	4 343 694 323	43 020 725	4 386 715 048	133 871
3	1 111 595 593	1 082 261 354	6 692 845 763	31 563 902	6 724 409 665	25 651
2	736 830 482	319 788 817	8 574 537 356	23 940 439	8 598 477 795	4 100
1	422 598 906	0	9 655 097 094	0	9 655 097 094	575
0	0	0	10 077 696 000	0	10 077 696 000	75

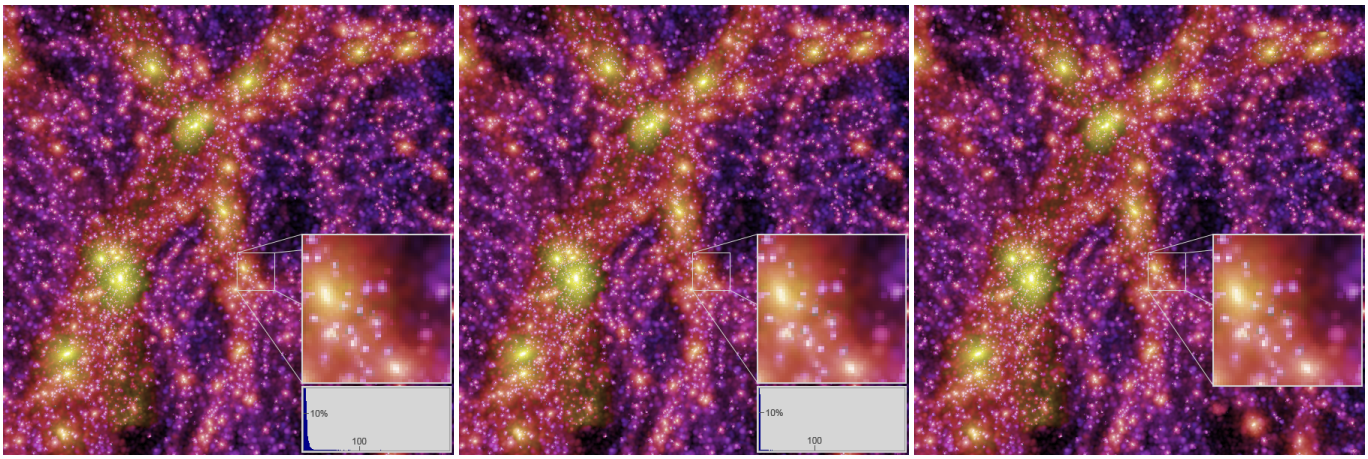

Fig. 8. The quality of vector quantization using 8 bit (left) and 16 bit (middle) for the compression of particle attributes is compared. On the right, the data was visualized using the original floating point attribute values. Close-up views are used to emphasize the quality of vector quantization. For the compressed data, the histogram of the color differences to the original image are shown below the close-up views.

In comparison to the total number of particles, the fraction that is represented as *points* is marginal up to level 7. In fact, this number is considerably larger, but since the *points* are typically part of very dense regions their number is reduced significantly by the proposed particle thinning strategy. While the total number of particles decreases for ever coarser LODs, the average number of particles per node is steadily increasing due to the decreasing number of nodes. In addition, on larger scales the data is distributed more uniformly across the nodes, resulting in a higher fill rate of the quantization bins. According to the increasing average number of particles per node, on levels 4 and higher the amount of memory required to store the particles exceeds the selected page size. Therefore, the collection of several nodes into one page does only occur on the four finest LODs.

In Figure 8 we compare the quality that can be achieved by vector quantization of particle attributes using 8 bits and 16 bits per component. In both cases the differences to the original data can hardly be seen, and they are only noticeable by investigating a magnified area in the data. To give an objective measure of the difference, we compute the Euclidean distance of all pixels in the RGB color space for multiple views. The histogram of the difference between the 8 bits compressed data and the original data backs up the visual impression by an average error of 2.0 in color space, where 255 corresponds to full intensity. With regard to the compression rate of 12 : 1, a high fidelity can be achieved. Comparing the 16 bit dataset with the uncompressed version, the difference is not even visible in the zoomed area so that the images can be regarded as visually equivalent. The analysis of the histogram proves that the images are not identical, but with an average error of 0.2 these differences are negligible.

9.2 Performance Measurements

To analyze the impact of the various parts of our visualization approach on its performance, we have measured memory consumption, data transfer rates, geometry throughput, and rendering performance during a straight flight at constant velocity through the Millennium Simulation. The velocity was 12.76 million lightyears per second, at a side length of the cubic simulation domain of 2.23 billion lightyears. The results are shown in Figure 9. By using a straight path, the amount of data in the spherical pre-fetching region that can be re-used in every frame is minimized. The particle data was rendered onto a 1200×800 viewport.

As can be seen in the uppermost plot, the main memory consumption is more or less constant around 1.5 GB. This is due to the pre-fetching scheme, which effectively hides variations in bandwidth requirements that are imposed by spatial variations of the particle densities. In the hard disk transfer rate in the second plot these variations are much more apparent, which is in particular caused by a large number of parallel I/O requests. However, with a maximum of about 105 MB/s we are still far away from the maximum data throughput of the striped RAID 0, which is above 200 MB/s. Most of the time, the transfer rate is below 60 MB/s. Hence, the sustained transfer rate of today's standard hard disks is sufficient for data streaming. Even if the used disk system would not be able to achieve the required peak rate, this would probably not result in a loss of detail since the measured transfer is only caused by data pre-fetching.

The third plot in Figure 9 shows the amount of graphics memory that is required for rendering. With an average size of 202 MB and a

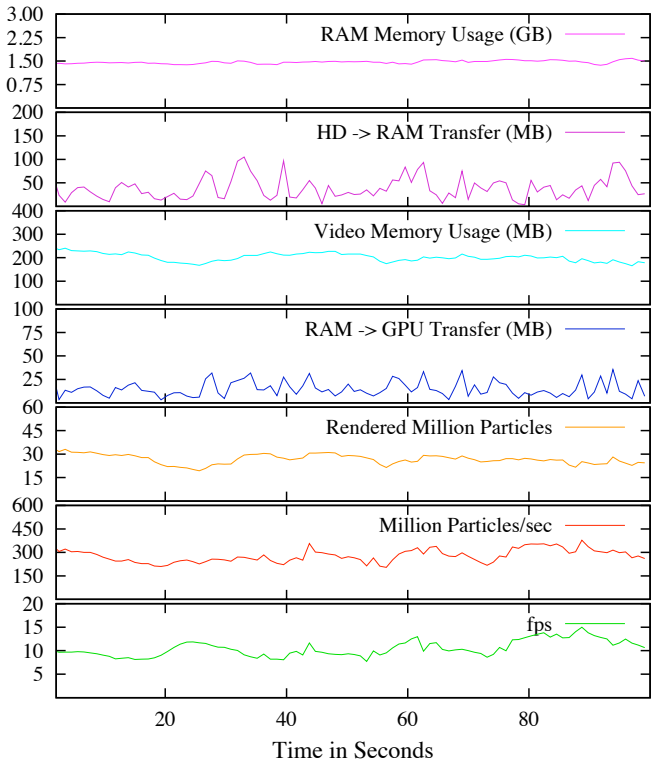

Fig. 9. Performance measurements and memory statistics for a flight through the Millennium dataset. Memory and bandwidth requirements are not at the system limits. Geometry throughput on the GPU effectively limits the performance when rendering onto a 1200×800 viewport.

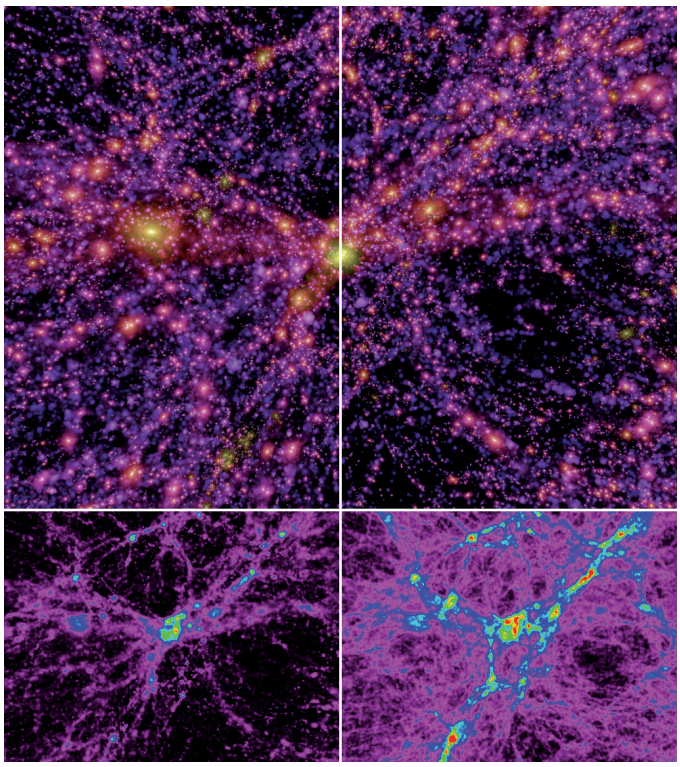

Fig. 10. Top: Comparison of image quality with (left, 10.4 fps) and without (right, 4.4 fps) presence acceleration. Bottom: Pixel overdraw in the respective upper image. Hue encodes the overdraw from black (0 fragments) over magenta (500 fragments) to red (3000 fragments).

maximum size of 246 MB, the memory requirements are remarkably low. In addition, with a CPU-GPU transfer rate of at most 35 MB/s we are far below the bandwidth that is available on current graphics hardware. In combination with the proposed pre-fetching scheme, the data to be rendered is resident in main memory when it is needed and can directly be streamed to the GPU.

The lower three plots of Figure 9 show the performance measurements for rendering. Overall, an average frame rate of 10.5 fps is achieved, with a minimum and maximum of 7.7 and 15.0 fps, respectively. To analyze the effectiveness of the proposed data reduction strategies, we have tried to render the same flight without applying particle thinning and presence acceleration, yielding an average frame rate of 1.63 fps.

Regarding geometry throughput, we achieve a peak performance of 394 million particles per second, with 280 million particles per second in average. This is interesting, because the theoretical maximum of our target architecture is at roughly 300 million triangles per second. Due to fine-grain view frustum culling and presence acceleration, however, the number of effectively rendered sprites is significantly below the number given in the diagram. Based on benchmarks we can state that the maximal throughput of our rendering approach is in fact about 220 million sprites per second, where the loss in performance is introduced by the geometry shader. This throughput is further reduced when rendering very dense regions. Despite our multi-resolution strategy, an overdraw of more than 2000 is not unusual in the current application, clearly indicating that rasterization will become the rendering bottleneck.

Figure 10 illustrates this problem and the effect of the density based culling strategy proposed in Section 8.2. The bottom pictures give information about the fragment overdraw for rendering with both culling techniques enabled (left) and without these techniques (right), leading to the same visual output (top image). The color coding ranges from an overdraw of 0 (black), 500 (magenta) to 3000 (red) with a linear interpolation of the color's hue for the values between 500 and 3000. Obviously, the number of pixels with a high overdraw as well as the maximum overdraw of fragments is much lower in the left image. Combined with GPU view frustum culling this results in a performance gain of 236%, yielding 10.4 fps instead of 4.4 fps. The culling techniques for themselves achieve frame rates of 9.2 (view frustum culling) and 4.7 fps (density-based culling).

Figure 11 illustrates the scalability of our system with regard to the viewport resolution. We attribute this scalability to the presence acceleration scheme, as it effectively prevents the geometry load from scaling linearly with screen resolution. Consequently, the drop in performance is mainly due to the increased fragment load, which — due to the use of a geometry shader — scales more gracefully.

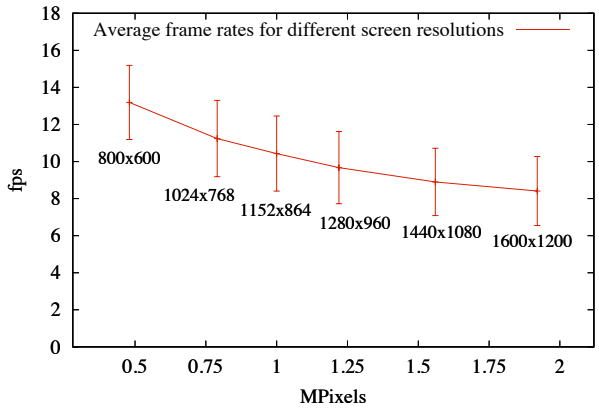

Fig. 11. Average frame rate and standard deviation for rendering the Millennium Simulation data at different viewport resolutions.

10 Discussion

The results presented in the previous section have proven the capability of our proposed techniques to render datasets with billions of particles at interactive frame rates on common PC hardware. By quantizing particle coordinates into a regular spatial grid and by using vector quantization to compress particle attributes, compression rates of up to 6 : 1 are achieved at very high fidelity. By employing an octree-based multi-resolution hierarchy, the amount of data to be transferred and rendered is considerably reduced. Based on an efficient memory management the streaming of data from hard disk to the video memory is completely hidden from the user for continuous camera movements. The measurements in Figure 9 clearly demonstrate the scalability of the framework, allowing for the interactive rendering of much larger datasets than the Millennium Run.

The bottleneck in the rendering of large particle data turns out to be the geometry throughput on recent GPUs. Despite their highly parallel streaming architecture, the massive geometry load induced by the rendered particles lets the GPU become the limiting performance factor. To overcome this limitation, one possible solution is to reduce the number of rendered particles by using alternative rendering primitives. For instance, recent findings in the context of GPU-based terrain rendering [5] show significant advantages of ray-casting over rasterization for the rendering of high-resolution polygonal terrains, where the polygons are approximately one pixel in size. The reason is that ray-casting can effectively exploit the regular grid structure underlying such fields and avoids pixel overdraw, while the performance of rasterization is limited by the polygon throughput on the GPU. Similar to this approach, it might be promising to resample regions exhibiting a high particle density into volumetric textures, and to render the dataset by means of a hybrid scheme using volume ray-casting and particle rasterization.

11 Conclusion and Future Work

In this paper, we have presented a scalable approach for the interactive visualization of large cosmological particle data on recent PC architectures. By quantizing particles into a multi-resolution octree structure and introducing rules for particle merging and deletion when no visual error is implied, we can significantly reduce the amount of data while maintaining a user-defined screen space error. Combined with an out-of-core and in-core data management to efficiently access the data and optimized particle splatting we are able to visualize one time-step of the Millennium Run exceeding 10 billion particles at interactive frame rates. To our best knowledge, this is the first approach that is able to interactively visualize particle data of this size.

As the massive geometry and rasterization load is the limiting performance factor of current particle rendering, our primary focus in the future will be to investigate alternative visualization techniques that have a better scalability with regard to the number of particles, such as volume ray-casting. By employing those techniques for overdense regions, we hope that a hybrid approach will lead to a considerable better rendering performance and scalability.

Acknowledgments

We would like to thank Volker Springel from the Max-Planck Society in Garching for giving us insight into the scientific needs and his visualization approach, and his colleague Gerard Lemson for his support with the dataset. Additionally, we would like to thank our colleague Christian Dick for his helpful discussions.

References

[1] J. Ahrens, K. Heitmann, S. Habib, L. Ankeny, P. Mccormick, J. Inman, R. Armstrong, and K. liu Ma. Quantitative and Comparative Visualization Applied to Cosmological Simulations. *Journal of Physics Conference Series*, 46, 2006.

[2] K. Brodlie, D. A. Duce, J. R. Gallop, M. S. Sagar, J. Walton, and J. Wood. Visualization in Grid Computing Environments. In *Proc. IEEE Visualization*, pages 155–162, 2004.

[3] H. R. Childs, E. Brugger, K. S. Bonnell, J. S. Meredith, M. Miller, B. Whitlock, and N. Max. A Contract Based System For Large Data Visualization. In *Proc. IEEE Visualization*, page 25. IEEE Computer Society, 2005.

[4] M. Cox and D. Ellsworth. Application-controlled Demand Paging for Out-of-core Visualization. In *Proc. IEEE Visualization*, pages 235–244, Los Alamitos, CA, USA, 1997. IEEE Computer Society Press.

[5] C. Dick, J. Krüger, and R. Westermann. GPU Ray-Casting for Scalable Terrain Rendering. In *Proceedings of Eurographics 2009 - Areas Papers*, pages 43–50, 2009.

[6] K. Dolag, M. Reinecke, C. Gheller, and S. Imboden. Splotch: Visualizing Cosmological Simulations, July 2008.

[7] A. Henderson. ParaView Guide, A Parallel Visualization Application. Kitware Inc., 2008.

[8] M. Hopf and T. Ertl. Hierarchical splatting of scattered data. In *Proc. IEEE Visualization*, pages 433–440, 2003.

[9] M. Hopf, M. Luttenberger, and T. Ertl. Hierarchical Splatting of Scattered 4D Data. *IEEE Computer Graphics and Applications*, 24(4):64–72, 2004.

[10] C. Johnson and C. Hansen. *Visualization Handbook*. Academic Press, Inc., Orlando, FL, USA, 2004.

[11] R. Kähler, D. Cox, R. Patterson, S. Levy, H.-C. Hege, and T. Abel. Rendering The First Star in The Universe - A Case Study. In R. J. Moorhead, M. Gross, and K. I. Joy, editors, *Proc. IEEE Visualization*, pages 537–540. IEEE Computer Society, IEEE Computer Society Press, October/November 2002.

[12] H. Li, C.-W. Fu, Y. Li, and A. Hanson. Visualizing Large-Scale Uncertainty in Astrophysical Data. *IEEE Transactions on Visualization and Computer Graphics*, 13(6):1640–1647, 2007.

[13] Y. Li, C.-W. Fu, and A. Hanson. Scalable WIM: Effective Exploration in Large-scale Astrophysical Environments. *IEEE Transactions on Visualization and Computer Graphics*, 12(5):1005–1012, 2006.

[14] A. Lintu, L. Hoffman, M. A. Magnor, H. P. A. Lensch, and H.-P. Seidel. 3D Reconstruction of Reflection Nebulae from a Single Image. In *VMV*, pages 109–116, 2007.

[15] J. Miller, C. Quammen, and M. Fleenor. Interactive Visualization of Intercluster Galaxy Structures in the Horologium-Reticulum Supercluster. *IEEE Transactions on Visualization and Computer Graphics*, 12(5):1149–1156, 2006.

[16] T. Munzner, C. Johnson, R. Moorhead, H. Pfister, P. Rheingans, and T. S. Yoo. NIH-NSF Visualization Research Challenges Report Summary. *IEEE Computer Graphics and Applications*, 26(2):20–24, 2006.

[17] P. A. Navrátil, J. L. Johnson, and V. Bromm. Visualization of Cosmological Particle-Based Datasets. In *IEEE Transactions on Visualization and Computer Graphics (Proc. IEEE Visualization 2007)*, Nov/Dec 2007.

[18] C.-M. Ng, C.-T. Nguyen, D.-N. Tran, T.-S. Tan, and S.-W. Yeow. Analyzing Pre-fetching in Large-scale Visual Simulation. *Computer Graphics International Conference*, 0:100–107, 2005.

[19] J. P. Ostriker and M. L. Norman. Cosmology of the Early Universe Viewed Through the New Infrastructure. *Commun. ACM*, 40(11):84–94, 1997.

[20] D. J. Price. SPLASH: An Interactive Visualisation Tool for Smoothed Particle Hydrodynamics simulations, 2007.

[21] A. Rosiuta, G. Reina, and T. Ertl. Flexible Interaction with Large Point-Based Datasets. In *Theory and Practice of Computer Graphics '06*, 2006.

[22] J. Schneider and R. Westermann. Compression Domain Volume Rendering. In *Proc. IEEE Visualization*, 2003.

[23] V. Springel. The Cosmological Simulation Code GADGET-2. *Monthly Notices of the Royal Astronomical Society*, 364(4):1105–1134, December 2005.

[24] V. Springel, J. Wang, M. Vogelsberger, A. Ludlow, A. Jenkins, A. Helmi, J. F. Navarro, C. S. Frenk, and S. D. M. White. The Aquarius Project: The Subhalos of Galactic Halos. *Monthly Notices of the Royal Astronomical Society*, 391:1685–1711, Dec. 2008.

[25] 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*, 435:629–636, June 2005.

[26] T. Szalay, V. Springel, and G. Lemson. GPU-Based Interactive Visualization of Billion Point Cosmological Simulations. *CoRR*, Nov. 2008.