# Particle-based Visualization of Large Cosmological Datasets

Niko Lukač*
*supervised by: Borut Žalik*†

University of Maribor
Faculty of Electrical Engineering and Computer Science
Laboratory for Geometric Modelling and Multimedia Algorithms
Smetanova ulica 17, SI-2000 Maribor, Slovenia.

## Abstract

Large quantities of simulated cosmological particle-based data cause considerable problems when it comes to real-time visualization. This paper considers an out-of-core approach for solving visualization problems on a single-desktop workstation. The approach proposed in this paper consists of two phases: the data preprocessing and its visualization. During the preprocessing, the cosmological data is hierarchically organized and efficiently ordered. Before rendering, the data is streamed to the memory from the disk. The culling techniques, such as view frustum culling and level-of-detail (LOD) are applied for visualization. In most cases, the real-time visualization of large cosmological particle datasets is achieved.

**Keywords:** Cosmological data, Particle-based visualization, Level-of-detail

## 1 Introduction

A computer based 3D visualization of observed or simulated particle data provides an insight for astrophysicists into a better understanding of the universe's properties. Cosmological visualization is extremely important for enabling the interactive exploration of not fully understood phenomena. Because the quantity of cosmological particle-based data increases every year, it has become a challenge to visualize this data in real-time on a single desktop workstation despite the recent advances in technology.

This paper presents an approach for the efficient visualization of large cosmological datasets using various paradigms, such as hierarchical data structure (octree), data ordering using hierarchical clustering, particle culling (frustum culling and LOD), data streaming, and prefetching.

The paper is organized into 6 sections. Section 2 provides a short summary and comparison with related work. Section 3 briefly describes cosmological particle-based data origin and properties. Section 4 focuses on the proposed visualization approach. Section 5 describes the results of our experiments on a desktop workstation. The conclusion summarizes the paper and suggests for improvements.

## 2 Related work

Several out-of-core point-based rendering approaches have already been developed over recent years [1], in order to efficiently visualize particle-based data. There are different kinds of hierarchical data structures, and LOD methods. Most of these methods are targeted towards rendering surface or mesh-based point-cloud data [2, 3, 4, 5]. The proposed method is designed to visualize cosmological particle-based data that can exceed system memory and represent an entirely different point distribution i. e. the "cosmic web". Because of clear differences in point distribution, the proposed method differs from point-cloud surface rendering methods, even though there are some similarities: use of clustering for LOD simplification [5] and hierarchical culling [2, 3, 4].

Several different approaches exist for cosmological particle-based visualization: advanced visualization toolkits [6], particle raytracing [7], isosurfaces extraction [8], parallel-based visualization using clusters of computers [9], particle culling methods [10, 11], spatial data division by hierarchical data structures [10 ,11, 12], exploiting the advantages of GPU [9, 10, 11], particle splatting techniques [11, 12], and particle attributes quantization and/or compression [11].

## 3 Cosmological particle data

Cosmological data is obtained from either observations or simulations. Most simulations are linked to the cosmic expansion of the universe [13, 14, 15] within a spatial cube, the size of which is defined in megaparsecs [Mpc] (1 Mpc is ~$3.262 \times 10^6$ light years or ~$3.08 \times 10^{19}$ km). A time-step snapshot from cosmological simulation is defined using cosmological redshift $z$, to describe the

* niko.lukac@uni-mb.si
† zalik@uni-mb.si

specific time of expansion. There are different N-body algorithms (cosmological codes) [16] that are used to run a particle-based simulation. They are based on a theoretical model describing not fully understood cosmological phenomena, such as dark matter. The data from observations contains portions of the observable universe [17] and is considerably smaller in size than simulated data. It is evident that large simulated cosmological particle datasets contain millions, even billions of particles, which makes it difficult to visualize them in real-time. One of the better known cosmological simulations is Millennium simulation [13], based on a Cold Dark Matter (ΛCDM) model, and contains over $2160^3$ particles resulting in large ~320 GB datasets for each single time-step snapshot. Individual particle of a cosmological simulation can represent different structures (e. g. dark matter fluid) and has a range of various attached attributes, depending on the simulation type. Most common attributes are particle floating point values of mass, spatial position and velocity.

# 4 Visualization approach

Our approach includes several methods aimed at reducing hardware load by rendering fewer particles, without visible loss of quality. The hardware load has the following constraints: GPU memory capacity, system memory capacity, disk capacity, disk latency, and a data transfer bandwidth between different components. Large-scale cosmological structures (e. g. filaments, clusters and halos) need to be preserved, in order to retain quality during the visualization. The proposed visualization solution is targeted towards static particle data, such as specific time-step snapshot of cosmological simulations.

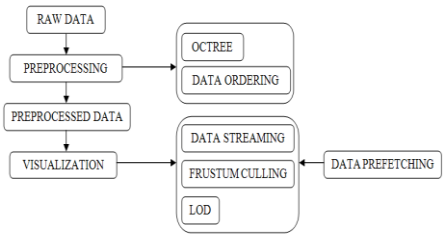


Figure 1: Workflow of the entire visualization approach.

The entire visualization process is shown in Figure 1. Firstly, the data is preprocessed by a hierarchical spatial subdivision using an octree, and efficient ordering. This phase is executed only once. Before rendering, the data has to be streamed from the disk to the system or, preferably, to the GPU memory. During the visualization phase, the rendering process is speeded-up by removing particles using view frustum culling, and LOD. Data prefetching is utilized to further increase the rendering performance.

## 4.1 Preprocessing phase

Firstly, the cosmological particle data is inserted into an octree structure. The octree is then constructed in a top-down manner, taking into account the particles positions. The inner nodes are only used for visibility testing, and the particles are stored inside the leaf nodes. The leaf's particles are stored on disk and later streamed to the GPU, because of the memory capacity constraint. A leaf node is constructed when there are fewer particles inside a given node than the threshold, which defines the maximum amount of particles that should be inside a leaf node at a specific tree depth. The threshold is defined as $1/(M–N+1)$ % of the number of particles from the node's parent node, where $N$ is the node's tree depth and $M$ is the maximum tree depth. When a node has more particles than the threshold, it will become an inner node and will be further divided. The particles inside a leaf node are spatially closer to the given leaf node's center than to any other node's center in the octree structure (see Figure 2). Although an inner node does not contain any data, it has information about the number of particles, which is the sum of the number of particles from all its descendants.

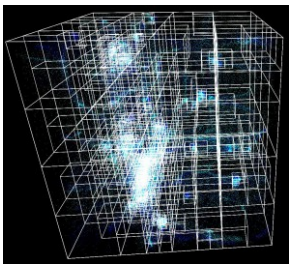

Figure 2: An example visualization of all octree nodes bounding boxes. The particles are visualized from all the viewable leaf nodes.

The adjacent leaf nodes are defined using the method described in [18] which is required for efficient data streaming and prefetching (see Subsections 4.5 and 4.6). At the end of preprocessing, the data inside each leaf node is additionally arranged using clustering, in order to minimize quality loss during LOD.

## 4.2 Data ordering

The order of the particle data inside a leaf node is the fundamental basis of the entire visualization quality. During the visualization phase, LOD calculates the amount of particles to be rendered for each visible leaf node. The data is read from the disk in linear order and streamed to the GPU, in order to gain the desired performance. Without careful data ordering, there would be huge quality losses during the visualization process, because of the removal of important particles.

The particle-based data inside a leaf is ordered according to the spatial distribution of the particles. A clustering algorithm is used that orders the particles using Euclidian distance metric. The results are particles that are bound in clusters according to their proximity. The smaller clusters (e. g. isolated particles) are more important than the larger clusters. The importance of a particle obtained in this way defines the particle order within the octree leaf node. Of course, the less important particles are retained, because they gain more importance

linearly as the camera moves closer. Consequently, at the end of the leaf node's final order, only the least important particles remain, belonging to the largest cluster.

As can be seen in Figure 3, the clusters of particles are arranged according to their capacities, in descending order. One particle from each arranged cluster is accepted in a bottom-up manner, and it is inserted into the final order, which is then stored on the disk for a particular leaf node.

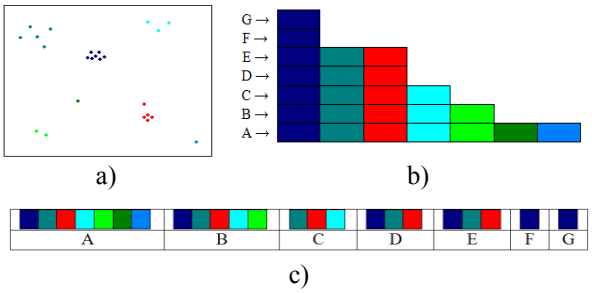


Figure 3: Data ordering - an illustrative example of particles data: a) colored clusters of particles, b) descending order of clusters according to capacities, c) final ordering of the particles within a stream.

There are several different and useful algorithms for spatial clustering [19]. The Chameleon hierarchical clustering algorithm, as proposed by Karypis et al. [20] was used in our implementation, because it offers a good trade-off between accuracy and speed.

## 4.3 Visualization phase

The visualization phase consists of three steps: particle culling, streaming, and rendering. Before rendering it is necessary to determine which particles to stream to the GPU. Frustum culling and LOD are employed for this purpose. The synergy of the culling and streaming allows the obtaining of suitable amount of data for rendering, based on the camera's viewpoint, and the view frustum's size. When the camera moves, any new data has to be streamed for rendering and the redundant data has to be removed from the memory. An illustration of the view frustum is shown in Figure 4.

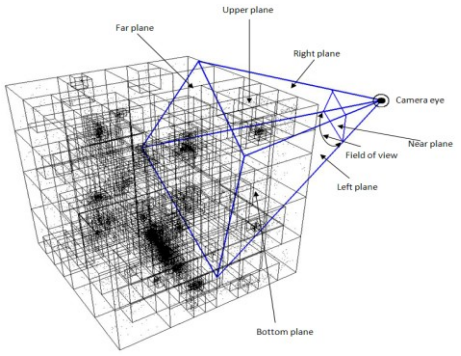


Figure 4: Illustration of the view frustum and the octree structure.

The view frustum culling and LOD can exploit the properties of the octree structure, because the data is preprocessed. The octree node's center points are used to represent the proximity of multiple particles within a specific spatial subvolume. Both culling methods traverse the octree structure from the root node and check the considered center points of the nodes for their visibility. Clearly, if the parent node is entirely outside the view frustum, its descendants are unchecked. The same applies if the entire node lies inside the view frustum. This allows for the skipping of many nodes during the culling process. The view frustum culling methods were used, as described in [21].

## 4.4 Level of Detail

After the view frustum culling is complete, the particles are further culled inside the view frustum using LOD. Better rendering performance is achieved, by adaptively adjusting the visualization details. The loss of quality is negligible, because the data was optimally ordered during the preprocessing phase. Consequently, the general particle distribution is preserved. The LOD streams the given leaf nodes data for rendering in linear order.

The LOD method uses the Euclidian distance metric between the center points of the leaf nodes and the position of the camera, in order to determine the amount of particles, to be loaded into the memory and then rendered for the given leaf node. The following equation is used for this:

$$\left(1 - \left(\frac{C_{dist}}{M_{dist}}\right)\right) * P_n \qquad (1)$$

where *Cdist* is the current distance from the camera to the leaf node, and *Mdist* is the maximum distance between the camera and the leaf node. In practice, this variable is limited by the distance from the front to the far plane of the view frustum. *Pn* is the total amount of particles in the leaf node. This equation provides similar effects as the traditional LOD error metric [1], which computes the projected screen space area of a leaf node.

The further the camera is from the considered leaf node, the fewer particles are rendered for this node. The direct consequence of this is that small details disappear from the greater distances, since larger clusters of particles are stuck within an area of few pixels or even a single pixel. The opposite happens, if the camera moves inside the given leaf node; in that case, all the particles inside the node are rendered. LOD quality can be observed in Figure 5, where the structure is retained intact even from the greater distances.

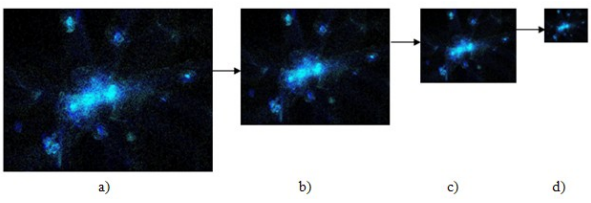


Figure 5: LOD on sample cosmological particle data (dark matter halo): a) 40%, b) 60%, c) 80% and d) 90% of *Mdist*.

## 4.5 Data streaming

The sizes of the large cosmological particle datasets exceed the memory capacity of any recent desktop workstation. The memory has to be used in an efficient manner, where only the recently viewed particles in a particular visualization instance are stored within the memory. Any preprocessed particle data from leaf nodes that have the same parent node are packed together and stored as a package into one file, in order to stream the data efficiently for thousands of octree leaf nodes from the disk to the memory. Threshold for a minimum package size has to be defined, because it is possible that a parent node does not contain 8 leaf nodes or that the capacities of the leaf nodes are small. In case the sum of the leaf nodes particles is below the threshold, the data of adjacent leaf nodes from other parent nodes is stored in the same file (see Figure 6). In our implementation the threshold was experimentally set to 1% of all particles for a given dataset. For a single snapshot dataset of the Millennium simulation, the minimum package file size is ~1.17 GB, storing only positions of the particles.

The reason for packing particle data together is based on the following premise: if one particular leaf node is rendered, then there is a high probability that the particles from the adjacent leaf nodes have be rendered, too. Prefetching is thus employed, at this stage.

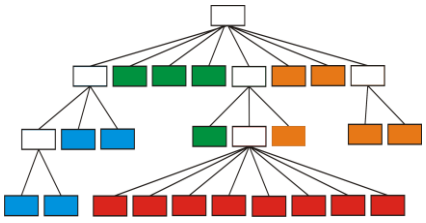

Figure 6: Example of non-balanced octree hierarchy, where the same colored leaf nodes are stored together.

## 4.6 Data prefetching

The use of available memory is maximized by prefetching proximal particle data outside the view frustum. A bounding sphere around the camera's center is constructed, for this purpose (see Figure 7).

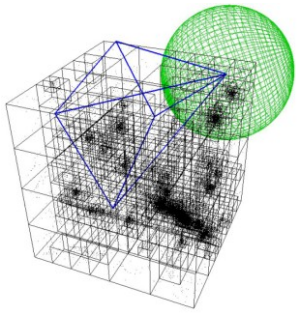

Figure 7: Illustration of the bounding sphere around the camera and the octree structure.

The radius of the bounding sphere is defined as the half distance between the front and the far plane of the frustum. This method is performed in the background and is efficient, when the camera is close to a specific area of interest. Based on the practical premise that the camera does not move over long distances when the area of interest is small, the required adjacent leaf nodes can be loaded into the memory. This applies to all the leaf nodes inside the defined bounding sphere or intersecting it, because the camera can move in any possible direction.

# 5 Results

This approach was tested on a single desktop workstation, using the following hardware: ATI Radeon HD 5750 GPU (1GB GDDR5 memory), AMD Phenom 1090T hexa-core CPU, 4 GB system memory (DDR3 1333 MHz), and SATA-II hard disk (7200 RPM, 1 TB capacity, 140MB/s average read speed). The OpenGL graphical library was used, where the particles were rendered as point primitive using color and alpha, without texture sprites. The streamed data from the disk to GPU was stored in VBO (Vertex Buffer Object) for each leaf. The visualization viewport resolution, was set at 1280x960, during the experiments.

In order to test the efficiency of the proposed visualization approach, several different and available cosmological particle simulation datasets were tested: Millennium (500 Mpc; z=0) [13, 24], MPA Larger box ΛCDM (479 Mpc; z=0) [23], GIF2 ΛCDM (110 Mpc; z=0) [14, 23], Mini Millennium (62.5 Mpc; z=0) [13, 24] and The Santa Barbara cluster (64 Mpc; z=0; gadget output) [15, 22].

The preprocessing required double the capacity of the input dataset. Around 80% of the preprocessing time was dedicated to data ordering for each dataset. However, this still took considerably less time than running a cosmological simulation, which takes several days on parallelized systems [13].

After preprocessing the quality loss and the performances gained for different datasets, were tested. To measure these properties, a predetermined camera path was made, which does a fly-through in the visualized data. The camera position was initially aligned with the X coordinate axis and distanced by the *Mdist* variable from the LOD. The cameras viewpoint was set to face the center point of the octree root node. The predetermined fly-through consisted of 10 steps. For each step the camera made a full 360 degree rotation, using 1 degree steps around the center point, and afterwards moved towards the center point over 10% of the initial distance *Mdist*. The testing was completed when the camera's position was equal to the center point. The speed was measured by performing fly-through three times for each dataset, in order to obtain more reliable results. FPS (Frames per second) was measured for speed comparison. A modified fly-through was made, in order to measure the quality difference. Each frame was visualized with the LOD both enabled and disabled. The per-pixel difference from both frames (LOD on and off) was calculated using the Euclidian distance metric, in

order to measure the quality. This comparison is suitable, because the projected particles on the frustum's near plane are the end result of rendering. The experimental results of the average culled particles for the whole fly-through, average fps and average quality decrease when LOD is on, are shown in Table 1.

Table 1: Results of the experiments on different cosmological datasets.

| Cosmological particle-based dataset | Number of particles | Average culled particles | Average FPS | Average quality decrease |
|---|---|---|---|---|
| Millennium | 10 077 696 000 | 88.2% | 11.3 | 7.4% |
| MPA ΛCDM | 134 217 728 | 80.2% | 67.4 | 6.7% |
| GIF2 ΛCDM | 64 000 000 | 85.3% | 88.5 | 4.9% |
| Mini Mill. | 19 683 000 | 87.9% | 142.0 | 6.1% |
| The Santa Barbara | 16 777 216 | 86.7 % | 177.1 | 5.3% |

Proposed out-of-core visualization approach was not compared to brute-force rendering, because this is practically impossible. A desktop workstation would run out of memory, when trying to render huge amount of particles (exceeding GPU and system memory several times) using the brute-force rendering approach.

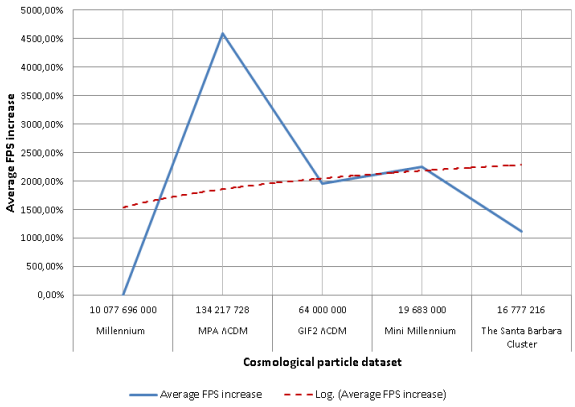


Figure 8: Average FPS increase (%).

FPS increased when employing LOD, providing a significant boost (see Figure 8). The quality decreased only slightly when the number of particles increased significantly (see Figure 9). When LOD was disabled, visualization remained in real-time, as long as there was enough GPU memory.

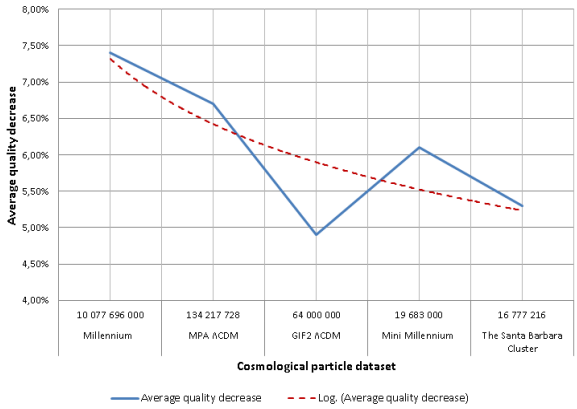


Figure 9: Average quality decrease (%).

Due to efficient data ordering and the hierarchical spatial subdivision, the average quality loss, which was around 5%, was hardly noticeable with the naked eye, as shown in the example in Figure 10. This is because the data ordering was done during the preprocessing phase, where the most clustered particles are of least importance (see Figure 10c).

a) b) c)

Figure 10: Sample visualized cosmological particle data using a) LOD enabled, b) LOD disabled and c) the difference (~5%).

Table 2 presents the averaged results for each even fly-through step for one of the largest tested particle dataset; Millennium. The average FPS dropped and the quality increased when the camera moved into the region of the visualized data. Within the areas of interest, LOD had less importance than frustum culling, which helped to preserve real-time visualization, since more nodes were being culled outside the view frustum, and data prefetching was being utilized in the background.

Table 2: Results for camera fly-through even steps of the Millennium dataset.

| Step | Camera distance from octree root node center point (% of *Mdist*) | Average culled particles | Average FPS | Average quality decrease |
|---|---|---|---|---|
| 2. | 80% | 96.9% | 15.3 | 2.9% |
| 4. | 60% | 84.2% | 8.1 | 6.4% |
| 6. | 40% | 88.5% | 9.2 | 8.1% |
| 8. | 20% | 94.8% | 11.9 | 5.9% |
| 10. | 0% | 95.3% | 14.2 | 2.4% |

Data prefetching was implemented on two background threads, each on its own CPU core, in order to have a smaller impact on the actual visualization. The FPS difference was measured by applying prefetching, using the same fly-through as before on the Millennium particle dataset. The results of this test are shown in Figure 11. The method paid off when the camera was close or inside the region of the visualized data.

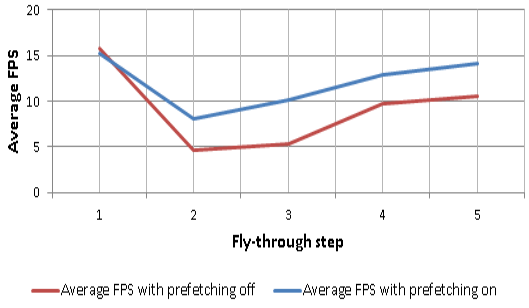


Figure 11: Average FPS for each camera fly-through even step of the largest tested dataset, with and without prefetching.

# 6 Conclusion

This paper shows that the large-structures of the cosmological particles are visualized in high details, with minimal impact from particle culling. We can efficiently render any cosmological particle dataset and in most cases achieve real-time visualization on a single desktop workstation, with a small quality loss.

We found that the main hardware bottlenecks on the desktop workstations are the disk read speed and on-demand data transfer from the memory to GPU in large quantities. In the near future such bottle-necks would limit real-time visualization, if the cosmological particle datasets snapshots are of the size of several terabytes. In order to overcome this problem, the presented approach could be extended using parallelization. There are also other ways for possible extension, such as data compression via GPU using technologies such as CUDA.

# Acknowledgements

Some of the used datasets in this paper are from simulations carried out by the Virgo Supercomputing Consortium [23]. We would like to thank V. Springel et al. [13] for giving us insights in to the Millennium simulation, L. Gao et al. [14] into the GIF2 simulation and K. Heitmann et al. [15] into the Santa Barbara cluster simulation. The Millennium Simulation databases used in this paper and the web application providing online access to them were constructed as part of the activities of the German Astrophysical Virtual Observatory (GAVO) [24]. Thanks to Gerard Lemson from GAVO for his support on the transmission of large Millennium simulation raw particle data. Thanks to Mel Krokos from the University of Portsmouth, U.K., who initiated this work.

# References

[1] M. Sainz, R. Pajarola, Point-based rendering techniques, ACM Computers and Graphics, vol. 28, no. 6, pp. 869-879, December 2004.

[2] C. Erikson, D. Manocha, W. V. Baxter III, HLODs for Faster Display of Large Static and Dynamic Environments, Proceedings of ACM Symposium on Interactive 3D graphics, New York, USA, March 2001.

[3] C. Dachsbacher, C. Vogelgsang, M. Stamminger, Sequential point trees, ACM Transactions on Graphics - Proceedings of ACM SIGGRAPH 2003, vol. 22, no. 3, pp. 657~662, New York, USA, July 2003.

[4] S. Rusinkiewicz, M. Levoy, QSplat: A Multiresolution Point Rendering System for Large Meshes, ACM SIGGRAPH Proceedings of the 27th annual conference on Computer graphics and interactive techniques, pp. 343-352, New York, USA, July 2000.

[5] M. Pauly, M. Gross, L. P. Kobbelt, Efficient Simplification of Point-Sampled Surfaces, IEEE Visualization Proceedings, pp. 163-170, Boston, USA, November 2002.

[6] M. Comparato, U. Becciani, A. Costa, B. Larsson, B. Garilli, C. Gheller, J. Taylor, Visualization, Exploration and Data Analysis of Complex Astrophysical Data, The Publications of the Astronomical Society of the Pacific, vol. 119, no. 858, pp. 898-913, August 2008.

[7] K. Dolag, M. Reinecke, C. Gheller, S. Imboden, Splotch: Visualizing Cosmological Simulations, New Journal of Physics, vol. 10, no. 12, 2008.

[8] P. A. Navrátil, J. L. Johnson, V. Bromm, Visualization of Cosmological Particle-Based Datasets, IEEE Transactions on Visualization and Computer Graphics, vol. 13, no. 6, pp. 1712-1718, 2007.

[9] Z. Jin, M. Krokos, M. Rivi, C. Gheller, K. Dolag, M. Reinecke, High-performance astrophysical visualization using Splotch, Procedia Computer Science, ICS 2010, vol. 1, no. 1, pp. 1769-1778, May 2010.

[10] T. Szalay, V. Springel, G. Lemson, GPU-Based Interactive Visualization of Billion Point Cosmological Simulations, November 2008.

[11] R. Fraderich, J. Schneider, R. Westermann, Exploring the Millennium Run-Scalable Rendering of Large-Scale Cosmological Datasets, IEEE Transactions Visualization and Computer Graphics, pp. 1251-1258, June 2009.

[12] M. Hopf, T. Ertl, Hierarchical Splatting of Scattered Data, IEEE Visualization Proceedings, pp. 433-440, Washington, USA, October 2003.

[13] 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, F. Pearce, Simulating the joint evolution of quasars, galaxies and their large-scale distribution, Nature, vol. 435, pp. 629–636, 2005.

[14] L. Gao, S. D. M. White, A. Jenkins, F. Stoehr, V. Springel, The subhalo population of LCDM dark haloes, Monthly Notices of the Royal Astronomical Society, vol. 355, no. 3, pp. 819-823, 2004.

[15] K. Heitmann, P. M. Ricker, M. S. Warren, S. Habib, Robustness of Cosmological Simulations I: Large Scale Structure, The Astrophysical Journal Supplement Series, vol. 160, no. 1, pp. 28-58, 2005.

[16] K. Heitmann, Z. Lukic, P. Fasel, S. Habib, M. Warren, M. White, J. Ahrens, L. Ankeny, R. Armstrong, B. O'Shea, P. M. Ricker, V. Springel, J. Stadel, H. Trac, The Cosmic Code Comparison Project, Computational Science and Discovery, vol. 1, no. 1, 2008.

[17] J. R. Gott III, M. Juric, D. Schlegel, F. Hoyle, M. Vogeley, M. Tegmark, N. Bahcall, J. Brinkmann, A Map of the Universe, The Astrophysical Journal, vol. 624, no. 2, pp. 463-484, 2005.

[18] J. Vörös, A strategy for repetitive neighbor finding in octree representations, Image and Vision Computing, vol. 18, no. 14, pp. 1085-1091, 2000.

[19] J. Han, M. Kamber, A. K. H. Tung., Spatial Clustering Methods in Data Mining: A Survey, Geographic Data Mining and Knowledge Discovery, vol. 21, pp. 1–29, 2001.

[20] G. Karypis, E. Han , V. Kumar, Chameleon: Hierarchical clustering using dynamic modeling, Computer, vol. 32, no. 8, pp. 68-75, 1999.

[21] U. Assarsson, T. Möller, Optimized view frustum culling algorithms for bounding boxes, Journal of Graphics Tools, vol. 5, no. 1, 2000.

[22] The Cosmic Data Bank, Online: http://t8web.lanl.gov/people/heitmann/arxiv/data.html (14.03.2008)

[23] MPA Numerical Cosmology, Online: http://www.mpa-garching.mpg.de/NumCos/ (30.10.2006)

[24] German Astrophysical Virtual Observatory (GAVO) – Virgo Millennium Database, Online: http://www.g-vo.org/Millennium/ (10.02.2011)