Eurographics Symposium on Parallel Graphics and Visualization (2015)
C. Dachsbacher, P. Navrátil (Editors)

Large-Scale Parallel Visualization of Particle-Based Simulations using Point Sprites and Level-Of-Detail

S. Rizzi[1] M. Hereld[1] J. Insley[1] M.E. Papka[1,2] T. Uram[1] and V.Vishwanath[1]

[1]Argonne National Laboratory, United States
[2]Northern Illinois University, United States

Abstract

Recent large-scale particle-based simulations are generating vast amounts of data posing a challenge to visualization algorithms. One possibility for addressing this challenge is to map particles into a regular grid for volume rendering, which carries the disadvantages of inefficient use of memory and undesired losses of dynamic range. As an alternative, we propose a method to efficiently visualize these massive particle datasets using point rendering techniques with neither loss of dynamic range nor memory overheads. In addition, a hierarchical reorganization of the data is desired to deliver meaningful visual representations of a large number of particles in a limited number of pixels, preserving point locality and also helping achieve interactive frame rates. In this paper, we present a framework for parallel rendering of large-scale particle data sets combining point sprites and z-ordering. The latter is used to create a multi level representation of the data which helps improving frame rates. Performance and scalability are evaluated on a GPU-based visualization cluster, scaling up to 128 GPUs. Results using particle datasets of up to 32 billion particles are shown.

Categories and Subject Descriptors (according to ACM CCS): I.3.2 [Computer Graphics]: Graphics Systems—Distributed/network graphics

1. Introduction

Recent large-scale simulations [HMF*13] require the most powerful supercomputers and produce vast amounts of data. Computing these very large datasets pushes a supercomputer to its limits, but is also challenging from the points of view of storage, analysis, and visualization.

In simulations which generate regular grid data, volume rendering is the primary method for visualization. In some cases, particle-based datasets are also mapped into regular grid volumetric representations to study density fields or other properties. However, mapping particle data onto regular grids can result in sparse datasets, often making inefficient use of system memory. The method may be adequate for some simulations, but it becomes inapplicable at the largest scales. Even using methods such as Adaptive Mesh Refinement (AMR) to mitigate data sparsity, there still are undesired losses of dynamic range due to the discrete nature of the grid, severely constraining the ability to resolve individual particles. These shortcomings underline the necessity of finding other methodologies for visualization of large-scale particle datasets.

Visualizing particles directly from their spatial coordinates is a more natural way of tackling the problem. A direct particle rendering method should be efficient in its memory usage (only particle coordinates need to be stored) as well as accurate in terms of preserving the dynamic range produced by the simulation, since mapping particles to a discrete grid is not required. In addition, it should ideally deliver a good visualization of the density field while also being able to resolve individual particles. These arguments provide good motivation to investigate direct particle rendering methods. Among them, point sprites are well known in Computer Graphics and are readily available in common OpenGL implementations. Within the scope of this paper, we focus on large-scale GPU-accelerated parallel point sprite rendering.

Most often, visualizing the data at interactive frame rates is a crucial requirement. In these cases, and particularly for far camera views, it is largely inefficient to fully render all points (or polygons in case of a polygonal mesh) of a large-scale dataset into a viewport containing a limited number of pixels. Under these conditions, the rendering time for the full dataset becomes impractical, while the resulting images

DOI: 10.2312/pgv.20151149

are usually too crowded and do not convey useful information. Level-Of-Detail (LOD) or sampling techniques are commonly used to simplify, in a systematic way, the number of elements rendered. We propose to use z-ordering [PF03], a method used for hierarchical subsampling of regular grid data, to reduce the number of particles rendered while preserving their local properties and clustering.

We present a parallel implementation of OpenGL point sprite rendering using a z-ordered hierarchical rearrangement of particles. We present our method to read large-scale particle datasets in parallel from storage, along with a data reordering strategy to dynamically reduce the number of particles rendered. Next, we describe a parallel implementation of the point sprite method, and compare it to volume rendering in terms of performance and image quality. Finally, we demonstrate weak and strong scalability using a dataset of 32 billion particles on 128 GPUs.

Our main contributions are:

1. Demonstrating point sprites and z-ordering on large particle datasets of a scale not previously reported in the literature (32 billion particles).
2. Scaling both methods on a large GPU-based parallel visualization and analysis resource (128 GPUs) without requiring out-of-core processing or offline indexing of the data.
3. Applying the combination of points sprites and z-ordering to render ultra-high resolution images (6144x3072 pixels) of large-scale particle data without losses of dynamic range.

2. Related Work

As identified in the previous section, Point-Based Rendering (PBR) has some advantages over volume rendering in terms of efficient use of memory and dynamic range. Similarly, Goswami et. al [GEM*13] point out some of the advantages of PBR versus polygonal mesh rendering: (i) points are more efficient than triangles in regions where triangles may project to individual pixels; and (ii) points provide a more compact representation than triangles since there is no mesh connectivity required. In their work, an out-of-core method for rendering massive point clouds using multi-way kd-trees is presented. Parallel rendering is implemented using the Equalizer framework [EQU] with either sort-first or sort-last strategies. OpenGL points are used as rendering primitives on up to six parallel rendering nodes and data sizes scale up to 368 million points. Additional work focusing on efficient rendering of large point clouds from LIDAR can be found in [KBK08] and [PNS*11].

A multi-resolution hierarchy with level-of-detail selection is presented in [FSW09],. Particles are rendered as points and expanded using the geometry shader to a screen-aligned face. Data consisting of 10 billion (2160^3) particles from the Millennium Simulation Project are used and experiments are run on a single GPU. Similarly, a method for high-quality particle rendering with point sprites on GPUs is presented in [GSGP06]. Results are shown for up to 2.8 million particles on a single GPU. Rivi et. al [RGD*14] discuss a redesign of Splotch, a rendering algorithm for particle-based datasets from astronomical data or computer simulations. CUDA kernels have been incorporated for parallel rendering. Results are shown for 400 million particles of a GADGET simulation on a single GPU. Finally, [LMC04] shows results for 16 million (256^3) particles using seven rendering nodes on the Hewlett-Packard Sepia parallel rendering cluster.

Paraview [AGL05] is a well-known tool for parallel analysis and visualization. Use of Paraview for interactive analysis and visualization of large cosmological N-body simulations is discussed in [WHA*11]. The paper presents contributions to the Paraview code such as a particle reader for "cosmo" and "GADGET" formats, as well as a halo finder using the friends-of-friends algorithm. The data readers use a topologically rectangular 3D spatial decomposition, where each parallel process will obtain all of the particles in its assigned subspace. Data is read as an unstructured point data set in VTK and rendered in parallel. A Paraview visualization of a one billion particle simulation is presented, as well as a test analysis on a simulation using 16 million (256^3) particles in 256^3 and 1024^3 simulation grids. Statistical sampling techniques to reduce the size of large particle datasets for in-situ analysis and visualization are discussed in [WAF*11]. This implementation was tested in Paraview with 8 billion (2048^3) particles. Another example of Paraview used for visualization of cosmological data is presented in [NJB07], with two million SPH particles and two million dark matter particles per time-step from a GADGET simulation. The particle data is interpolated into a 128^3 element regular grid and imported into Paraview for visualization as isosurfaces.

There are a number of methods that integrate volume rendering and particle visualization. In [KAH07], there is a combination of volumetric fields representing interstellar gas densities and unstructured point sets for dark matter components. The particle data is encoded in an octree structure that is uploaded to GPU texture memory. In this way, the ray tracing process for the regular grid data can also take into account the particle data by sampling the octree structure in texture memory. Results are presented for a 256^3 structured grid and approximately 10K particles on a single GPU. Combustion simulations producing volume and particle data are discussed in [YWG*10]. Volume rendering is complemented with software-based particle rendering using point sprites. The implementation scales up to 15360 cores, 1620x1280x320 volumes, and 41.1 million particles, with image compositing dominating the total rendering time, as it is often the case in large-scale sort-last visualization. Among other methods combining particle and volume rendering, there is also Particle-Based Volume Rendering (PBVR). It was proposed by Sakamoto [SNKT07] in 2007, and it essen-

tially consists of sampling a 3D scalar field with particles, and subsequently projecting these particles into the image plane. The method was initially proposed as an alternative to ray casting entire volumes, but it kept evolving and it has been recently applied to render large-scale unstructured volume datasets [SMKK13].

Z-ordering, or Morton-ordering, uses space filling curves to convert n-dimensional integer coordinates into 1-D indices that can be sorted. The mapping preserves the locality of the original coordinates (i.e close points in N-dimensional space are also close in the sorted 1-D array.) This property has been used to create multi-resolution representations of regular grids [PF03] [BCH12] where the data can be accessed by ray tracing (or other methods) at different sampling resolutions. An algorithm in [CK10] proposes an extension to the method to use floating point values as coordinates. The algorithm is simple to implement, and there is an open source implementation available as part of the STANN library [STA]. The z-ordering method has also been used for building octrees in gravitational N-body codes [BGZ12]. Finally, [LZC14] present a z-order based method for sampling and surface reconstruction of large scale point clouds.

3. Implementation

This work has been implemented as part of vl3 [RHI*14], a parallel GPU-based framework for large-scale data visualization and analysis. vl3 is modular and facilitates the development and testing of new algorithms and methods, as well as deploying it on varied hardware platforms. In the following subsections we will describe our system design, the parallel rendering architecture of vl3, the domain decomposition and parallel data reading implemented, the hardware-accelerated method used for point sprite rendering, and a z-ordering based strategy for decomposing large-scale particle datasets into multi-level representations.

3.1. System Design

Our design is based on the following criteria:

1. Minimizing I/O operations: data is read only when execution of the program starts (Section 3.3). There are no particle exchanges afterwards. Network communication is used exclusively for compositing and synchronization.
2. Full utilization of the available parallelism: GPUs are kept busy rendering particles (Section 3.4) and used for compositing.
3. Minimizing transfers to/from GPU: particle data is rearranged (Section 3.5) and transferred to GPU after the initial reading from disk. No other transfers to GPU take place after the initial transfer.

3.2. Parallel Rendering Architecture

The vl3 parallel rendering architecture consists of the following modules:

Data Input: This module is responsible for reading data efficiently from a parallel filesystem and presenting it to the next stage in the pipeline via a common interface. There are readers for most volumetric data formats, including VTK, raw, HDF5, among others. As part of this work, a parallel reader for particle data has been added to the previously existing readers.

Rendering: vl3 parallelizes the rendering process by spatially decomposing the 3D region of interest into smaller equal-sized chunks (including ghost cells), and dividing them among multiple nodes of a distributed memory computer. In this work, we have implemented a parallel rendering module for large-scale particle datasets using point sprites with OpenGL and GLSL shaders, explained in Section 3.4

Compositing: When subsets of the full dataset are rendered by separate compute nodes, the result is a number of 2D images that need to be composited together into the final image. vl3 implements a number of parallel compositing algorithms, including DirectSend and BinarySwap.

Output: Once the rendering and compositing steps are complete, results must be displayed . vl3 supports multiple modalities, including batch mode, interactive visualization, and streaming. For larger datasets, vl3 can be run as a Message Passing Interface (MPI) application on a GPU cluster. In this case, large pixel count images can be streamed to high resolution displays such as 4K monitors or tiled displays [HIO*11].

3.3. Domain Decomposition and Parallel Data Reading

The 3-dimensional spatial domain is decomposed using a recursive binary partition scheme. Every MPI rank receives an equally-sized (in terms of spatial extents) subset of the whole domain. Due to the nature of the data generated by the simulation, the number of particles contained in those subsets does not vary much, which helps balancing the load among ranks.

After domain decomposition, every MPI rank owns a bounding box delimiting the spatial boundaries of the subset it has been assigned. As there is one MPI rank per GPU, this effectively means that a GPU will be responsible of rendering the subset of particles contained in its respective MPI rank bounding box.

Since ours is currently a post-processing approach (as opposed to in-situ), particles are stored by the simulation in a parallel filesystem and must be read from disk for visualization. In this work, we have implemented readers for the "cosmo" and "Generic IO" particle formats. Cosmo is a binary format for storing particle positions, velocities, particle identification numbers and other scalar values [WHA*11]. On the other hand, Generic IO is a format for efficient parallel I/O, either using Posix or MPI I/O, implementing the concepts discussed in [BFV*14]. At simulation time, every rank

writes its subset of particles in a Generic IO data "block". To improve I/O efficiency, blocks are aggregated and written into a discrete number of files in the parallel file system.

In practice, large-scale simulations run on thousands of cores, resulting in thousands of MPI ranks and an equal number of data blocks on disk. Post-processing tasks, such as visualization and analysis, may run on smaller clusters with a substantially lower number of ranks. As a result, every MPI rank in our implementation will be responsible for not just one, but several data blocks on disk. Once the domain decomposition stage finishes, every rank knows the extents of its bounding box. Afterwards, every rank obtains a list of all the blocks available on disk an checks their extents, comparing them with its bounding box to determine whether the block contains particles in the rank bounding box.A list of blocks is created containing the blocks needed by each rank, and finally, the blocks in the list are read in parallel from the file system. In this way, every rank reads from disk only the blocks containing particles in its bounding box, while the reading is efficiently performed in parallel.

3.4. Parallel Rendering of Point Sprites

Point sprites is a well-known technique to visualize particle systems. For a good introduction to its usage in OpenGL the reader is referred to [SWH13] . The point sprite functionality has been available in OpenGL since version 1.5, allowing to draw single points that are converted to camera-facing quads in the graphics pipeline. These quads can be textured to represent objects of interest. In our case, the objects are particles and 2D gaussian textures are used to represent them.

We leverage the CUDA N-body simulation demo [NHP07], part of the Nvidia CUDA GPU Computing SDK [CUD], where OpenGL point sprites are used to visualize particles evolving under simulated gravitational interaction. In our case, the technique is extended to operate in parallel as part of the vl3 framework, with every MPI rank taking care of all particles in its subregion of the simulation space. Vertex Buffer Objects (VBOs) are allocated on GPU memory and the particle information is copied into them. In every rendering cycle, the parallel renderers receive the position and orientation of the camera and render their particle data as point sprites, generating the resulting image in a Frame Buffer Object (FBO), whose contents are then read back and sent over the network to the corresponding compositing nodes. vl3 classes to support GLSL shaders are used to implement fragment and vertex shaders that modify the size of the sprites according to their distance to the camera, also applying a Gaussian texture to them. Alpha blending is enabled and set to accumulate color values on the color buffer.

3.5. Z-ordering for Level-Of-Detail

Rendering a large scale dataset in its totality could be considered a brute force approach. In cases like that, all particles are equally important and rendered no matter their location or contribution to the formation of structures. However, it is possible to substantially reduce the number of particles rendered while still preserving the particle clustering.

Level-of-Detail (LOD) is a well-known method in Computer Graphics to reduce the complexity of polygonal models or other data types, specially when the view camera is far from the object. In that case, rendering the data in full detail does not result in better image quality. Moreover, performance may also degrade due to the rendering of unnecessary graphics primitives. To overcome these problems, distance-based LOD defines different representations of a dataset with a number of primitives inversely proportional to the distance from camera to object. Similar approaches are particularly applicable to large-scale data.

In the scope of this work, we are interested in reorganizing a large number of particles and implementing an LOD scheme to avoid rendering the full dataset when either (i) the camera is far from the object; or (ii) frame rate constraints are imposed. There are multiple approaches in the literature where kd-trees are frequently used with the purpose of reordering, classifying, or sampling large-scale particle datasets (see for example [GEM*13] and [WAF*11].) In this paper, we propose a scheme for reordering particles according to their spatial distribution based on the Morton space filling curves, also known as z-ordering.

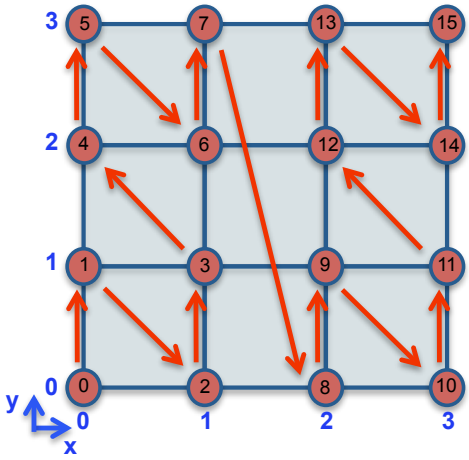


Figure 1: *Sorting in z-order a small set of particles arranged in a 4x4 grid. The numbers in blue show the discrete coordinates in the regular grid, whereas the numbers in each particle represent their order in the z-ordered sequence. Note the characteristic Z (or, more properly, N) patterns defined by the arrows.*

Originally, z-ordering schemes were proposed for regular grid data [PF03], where a Morton key is derived from interleaving the bits of the binary representation of integer

point coordinates. Figure 1 shows an example of the original approach, where the Z (or N) patterns are clearly distinguishable once the points have been sorted. For particle systems, though, coordinates are commonly expressed in floating point representation and the original algorithm has to be slightly modified. Connor and Kumar [CK10] describe an algorithm for fast comparison of floating point numbers in Morton order (i.e z-order) without computing a numeric key. Their algorithm is implemented in the STANN [STA] library. Testing it with a simple 4x4 pattern of points within a unit square (similar to the arrangement in Figure 1 but using floating point coordinates), we found that the STANN implementation failed to sort some of the particles in the correct z-ordering. For that reason, we decided to implement a z-ordering algorithm based on conversion from float to integer coordinates and apply the original scheme in [PF03] for regular grids, as [BGZ12] did for n-body simulations.

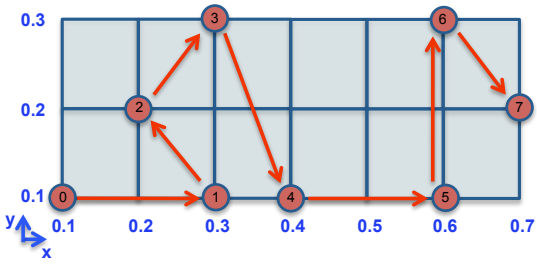


Figure 2: *Sorting in z-order a small set of particles with floating point coordinates. For simplicity, a two-dimensional system is shown. The number in each particle can be obtained from the particle coordinates and shows its order in the z-ordered sequence.*

X	Y	BinX	BinY	Shuffle	Key	z-ord
0.1	0.1	001	001	000011	3	0
0.3	0.1	011	001	001011	11	1
0.2	0.2	010	010	001100	12	2
0.3	0.3	011	011	001111	15	3
0.4	0.1	100	001	100001	33	4
0.6	0.1	110	001	101001	41	5
0.6	0.3	110	011	101101	45	6
0.7	0.2	111	010	101110	46	7

Table 1: *Converting floating point coordinates of a small set of particles to integer representation and sorting in z-order*

To illustrate the method, Figure 2 presents a simplified problem consisting of eight particles with floating point coordinates in two dimensions. The particle coordinates are shown in Table 1. Floating point coordinates are normalized, scaled, and converted to 3 bit integer representation. In this way, float 0.0 is mapped to binary 000, whereas float 0.7 is mapped to 111, using the entire range of the 3 bit binary representation. The bits in the binary coordinate representation are interleaved (Shuffle column), and the latter is converted to a numeric key, with which it is possible to sort the particles. This sorted list is the z-order representation of the system. The arrows in Figure 2 show the z-ordered sequence from the values computed in Table 1.

In our implementation we use 96 bit keys per particle (32 bits per coordinate component.) The 96 bit keys are subsequently sorted in parallel on each node using OpenMP. Thus, every MPI rank creates an array of particles sorted in z-order, by which the 3-dimensional particles are arranged into a 1-dimensional list.

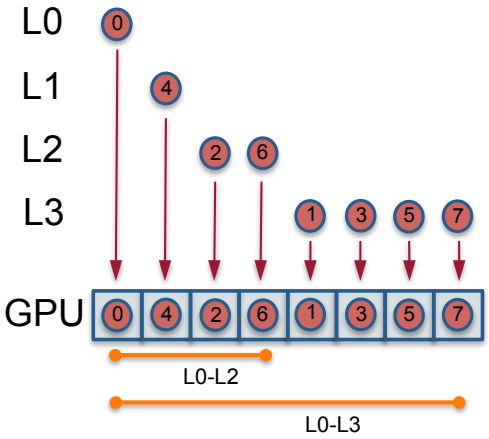
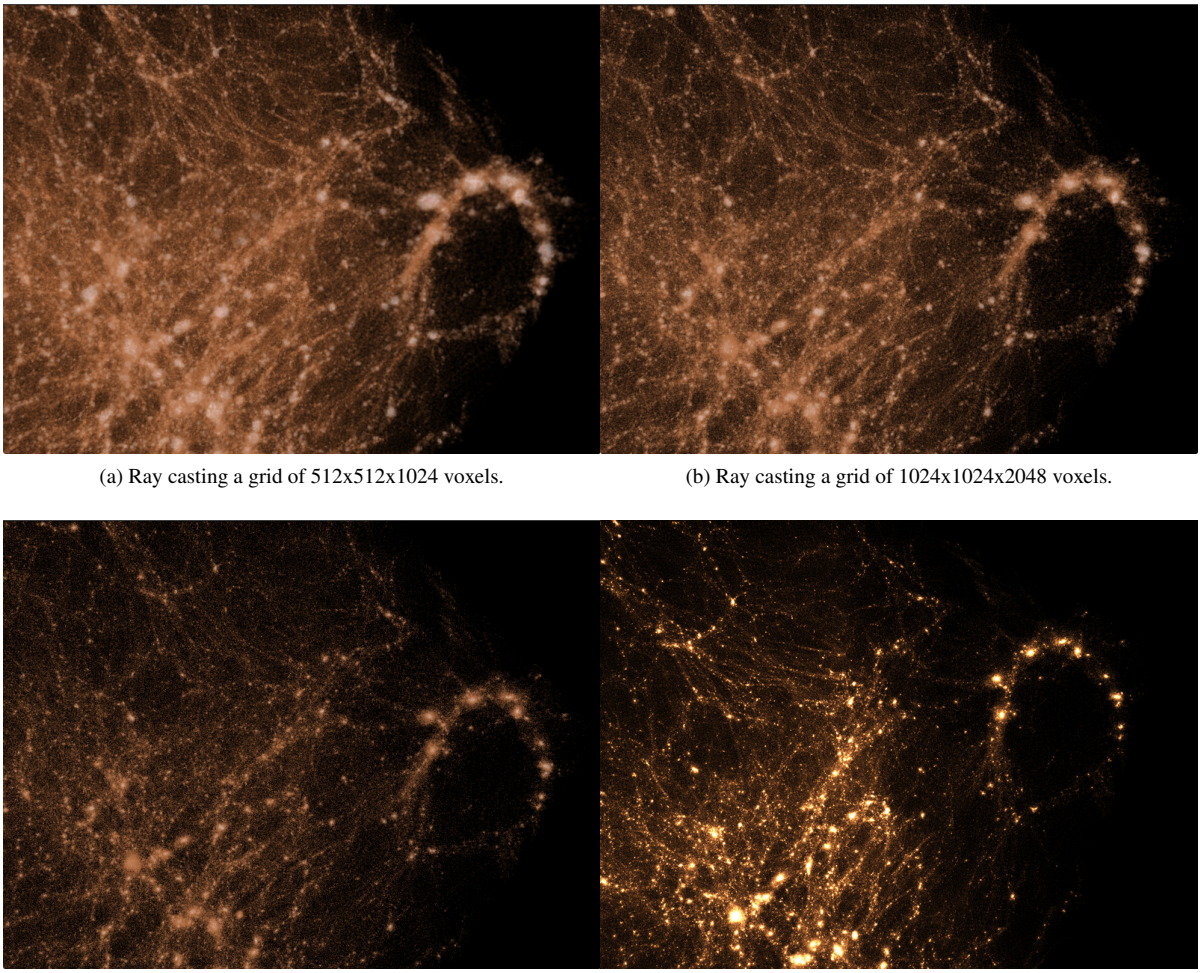


Figure 3: *Rearranging the z-ordered particles from Figure 2 in a multi resolution scheme. The entire particle set is transferred to the GPU after rearrangement. Orange segments show how successive levels are stored in consecutive positions of GPU memory.*

The z-ordered array can be further rearranged to create a *progressive* multi resolution representation of the data, as shown in Figure 3. The coarsest level contains a single particle (i.e the first particle appearing in the z-ordered array.) Successive levels refine the resolution by increasing the number of elements. Note, for example, how the combination of levels L0 and L1 contains particles with z-order index 0 and 4, which represent a good spatial sampling in Figure 2, and is similar to the z-order method for sampling regular grids in [PF03].

Particles in upper levels of the multi resolution scheme appear first in the array, which can be further transferred to the GPU in its entirety. In this way, the particles appearing first in the GPU buffer object will correspond to the lower resolution representations and are used to provide an approximation of the full distribution. In most cases, even a small percentage of the full dataset sorted in this way provides a very good visual approximation of the data, requiring substantially lower times for rendering. Since one of the parameters for rendering a VBO in OpenGL is the buffer size, reordering



(a) Ray casting a grid of 512x512x1024 voxels.

(b) Ray casting a grid of 1024x1024x2048 voxels.

(c) Ray casting a grid of 2048x2048x4096 voxels.

(d) Point sprite rendering the full particle subset.

Figure 4: *Comparing image quality of ray casting and point sprite rendering. A close camera view of the dataset is shown.*

the particles in the VBO according to the scheme described above and controlling dynamically the size of the rendered buffer results in a very simple and efficient implementation of LOD. The full dataset can be rendered by simply specifying the full size of the VBO, which contains all the reordered particles. When a lower resolution representation is desired, the size parameter is reduced accordingly and a subset of the reordered dataset will be rendered (corresponding to the upper levels of the hierarchical representation), proportionally reducing the rendering time.

4. Evaluation

In this section we present experimental results for evaluating our implementation using a dataset of 32 billion (3200^3) particles. Our experimental testbed consists of Tukey, a computer cluster at Argonne National Laboratory. There are 96 compute nodes based on the AMD Dual Opteron 6128 processor, with 16 cores per node. Every node also has 64 GB RAM and two Nvidia Tesla M2070 GPUs, adding up to 6 terabytes of CPU RAM and 1.1 terabytes of GPU RAM for the entire system. The peak GPU performance for the system is estimated at over 98 teraflop. The nodes are connected via a QDR Infiniband interconnect and the MPI implementation is Mvapich2 [MVA] from Ohio State University (OSU).

We first compare ray casting volume rendering and the point sprites method in terms of image quality, memory requirements, and frame rates in subsection 4.1. Next, in subsection 4.2, we study the efficacy of reorganizing the particle data in a multi level layout to preserve the high density regions as we render a fraction of the full data. Finally, in subsection 4.3, we discuss the weak and strong scaling per-

formance of our implementation, and demonstrate scaling to a 32 billion particles dataset.

4.1. Comparison of Image Quality

We compare the image quality produced by ray casting volume rendering and the point sprites method. Density fields can be captured and visualized by ray casting. However, due to the discrete nature of the grid, multiple particles may be mapped into a single grid point, thus the method may fail to resolve individual particles severely impairing the results produced by high-resolution simulations.

A subset of about 21 million particles generated by a single rank of a large cosmological simulation was used for the experiments in this subsection. We map the particle dataset spanning 30x30x60 Mpc/h, for ray casting, onto regular grids with sizes of 512x512x1024, 1024x1024x2048, and 2048x2048x4096 elements. For point sprite rendering, all the available particles were used. Eight GPUs were used in all cases to accommodate the largest grid size in GPU memory. It is worth noting that a single GPU can easily store the number of particles used in this experiment for point sprite rendering, but we decided to use eight in all cases to minimize parameter variations from one method to another.

Figure 4a shows the result of a close view of the 512x512x1024 dataset. As we can see, densities are nicely captured, but the coarser resolution of the grid is clearly noticeable and the dynamic range is negatively impacted. A grid of 1024x1024x2048 elements in Figure 4b shows better resolution, but the areas of higher densities start to diffuse. In Figure 4c the grid is increased to 2048x2048x4096 elements, increasing memory requirements by a factor of eight, without significant improvement in image quality. On the other hand, the point sprite rendered image in Figure 4d clearly shows the higher density regions while preserving the individual resolution of each particle. The point sprite method, compared to ray tracing volume rendering, is also able to nicely capture mass densities directly from particle coordinates, delivering images of the highest quality. Most importantly, the method preserves the dynamic range of the simulation by being able to resolve individual particles.

Data sizes, memory requirements, and frame rates obtained are summarized in Table 2. Even though it is difficult to establish a strictly fair comparison due to a variety of factors affecting performance (selection of number of samples per ray in ray casting, point size and sprite size in point sprites), results in the table show that point sprites require a substantially lower memory footprint and are better suited to achieve interactive frame rates.

4.2. Image Quality with Level-Of-Detail

We evaluate the image quality obtained using by the hierarchical reordering of particles proposed in Section 3.5. In the scheme, particles are rearranged using z-ordering and stored on a single GPU Vertex Buffer Object and rendered using the OpenGL function *glDrawArrays()*. This function receives as one of its parameters the number of primitives to be rendered and we can dynamically modulate this parameter in each frame to control the number of particles sent to the graphics pipeline.

Method	Data Size	Mem.Req.	Fps
Raycast	512x512x1024	256 MB	3.3
Raycast	1024x1024x2048	2 GB	3.2
Raycast	2048x2048x4096	16 GB	2.6
PointSpr.	21 million part.	240 MB	10.5

Table 2: *Comparison of memory requirements and achievable frame rates for ray casting and point sprite rendering*

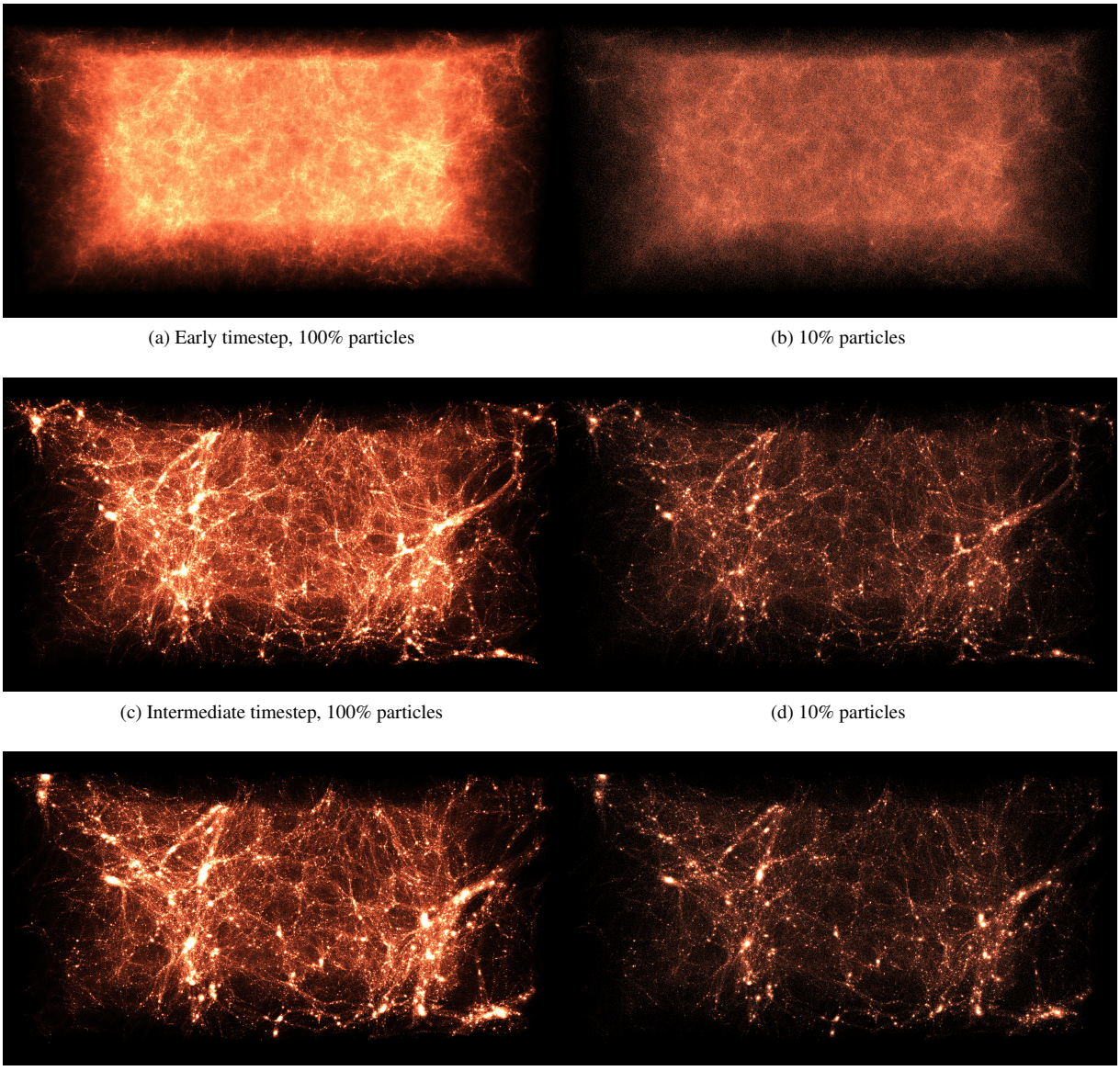
Figure 5 shows an example of the technique applied to a particle dataset from a cosmology simulation. The left column shows the view obtained when rendering the full buffer size as point sprites. On the right side, 10% of the particles are rendered for the same view.

It can be seen that reordering the particles in z-ordering captures nicely the areas of higher densities, preserving the underlying structure of the data even when not drawing a large fraction of the total particles. Multiple time steps are shown in Figure 5 to demonstrate that the method is applicable to data in all stages of the simulation.

4.3. Weak and Strong Scaling

We study the weak scaling performance of our parallel point sprite rendering implementation as we increase the number of GPUs while preserving the average number of particles rendered per GPU. We scale from 2 to 128 GPUs and each GPU loads about 250 million particles, thus, scaling from 500 million particles to 32 billion particles. We study two scenarios, where every GPU renders (i) its full subset of about 250 million particles; and (ii) a 10% fraction of its particles using the multi resolution method described in Section 3.5. We render the data to a final image of 6144x3072 pixels (resolution of our tiled display). We select a camera view such that all the particle subsets contained on each GPU contribute to the final image.

Figures 6a and 6b show the results of the experiments. We plot the maximum time of all GPUs in each case. On the left hand side of Figure 6a, there are 2 GPUs totaling about 500 million visible particles. On the right hand side, we scale to 128 GPUs containing a full dataset of 32 billion particles. The chart shows good weak scaling properties - we achieve 80% parallel efficiency at 128 GPUs with respect to 2 GPUs. It can be observed that the overall time to obtain a 6144x3072 pixel image of the full dataset takes a few seconds (orange line) – resulting in non-interactive frame rates.



(a) Early timestep, 100% particles (b) 10% particles

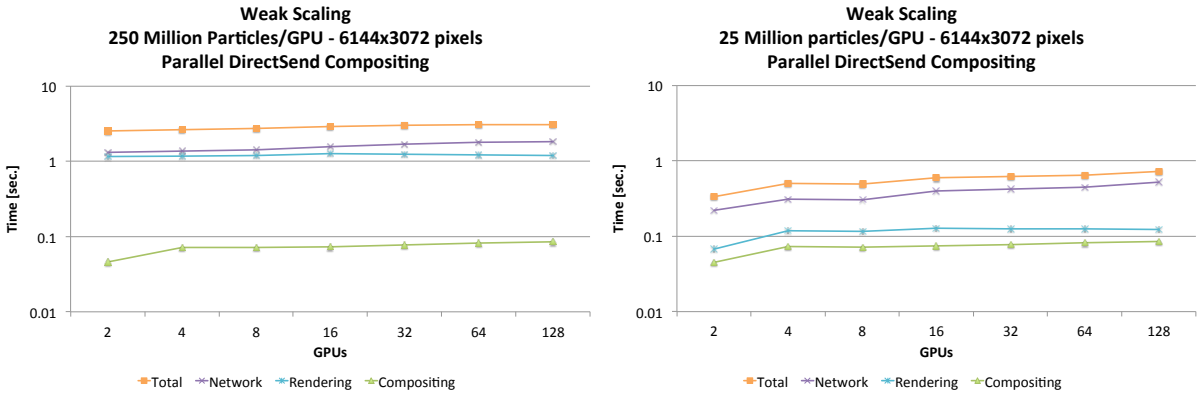
(c) Intermediate timestep, 100% particles (d) 10% particles

(e) Final timestep, 100% particles (f) 10% particles

Figure 5: *Image quality under varying levels of detail for different stages of the simulation. Note that the algorithm preserves the areas of higher densities, even when discarding a significant fraction of particles*

Figure 6b shows results for the same dataset under the multi level representation provided by the z-ordering, where 10% of the total particles are rendered without substantial loss of image quality. In this case too, we achieve good weak scaling with a parallel efficiency of 45% at 128GPUs. We observe an improved total time using the level of detail with respect to rendering the entire particle dataset, and thus, an improved frame rate leading to better interactive performance.

Strong scaling is evaluated selecting a particle dataset with a total of 2 billion particles, needing a minimum of 8 GPUs to fully accommodate it, and scaling the experiments to 128 GPUs while keeping the data size fixed. Figure 7 depicts the results obtained for an image size of 6144x3072 pixels. Network communication time for compositing dominates at scale, affecting the total frame time and the parallel efficiency, which is 27% at 128GPUs (with respect to 8



(a) Weak scaling, 100% particles

(b) Weak scaling, 10% particles

Figure 6: *Weak scaling evaluation. Subfigure 6a shows that a full dataset of 32 billion particles can be rendered at a resolution of 6144x3072 pixels when using 128 GPUs.*

GPUs.) It is also interesting to observe that for 128 GPUs, each GPU is responsible for about 15 million particles, a number comparable to the 25 million particles per GPU in the 10% weak scaling case, Figure 6b. Accordingly, all timings for 128 GPUs exhibit similar values in Figures 6b and 7 (compare values at the rightmost side of both figures).

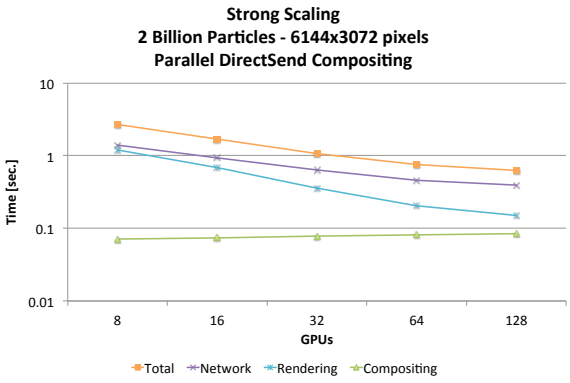


Figure 7: *Strong scaling from 8 to 128 GPUs. A subset of two billion particles is used in all cases.*

5. Conclusion and Future Work

In this paper we have shown multiple advantages of parallel hardware-accelerated point sprite rendering for rendering large-scale particle datasets. We have initially focused on rendering performance and image quality, scaling to a full dataset of 32 billion (3200^3) particles. The current implementation has shown good weak and strong scalability on up to 128 GPUs. The image quality provided by the proposed point sprite method preserves dynamic range and clearly outperforms ray tracing volume rendering, specially for closer views. Also, the hierarchical reordering method presented here can be used for Level-of-Detail and preserves important features in the data, even when rendering only a small fraction of the original data. Our implementation can also scale to large pixel counts, resulting in reasonable frame rates when combined with the LOD scheme.

Future work will concentrate on interactivity and scaling to even larger particle datasets. To improve interactivity, the LOD proposed in this paper could be dynamically adjusted depending on parameters such as camera distance or target frame rate. Also, there is room for optimization of the network communication time during compositing, currently one of the major observed bottlenecks. For interactive exploration of datasets consisting of more than a trillion particles, we will study combinations of Level-of-detail, out-of-core, progressive reading, and other methods.

Acknowledgement

This work was supported by the Office of Advanced Scientific Computing Research, Office of Science, U.S. Department of Energy, under Contract DE-AC02-06CH11357 including the Scientific Discovery through Advanced Computing (SciDAC) Institute for Scalable Data Management, Analysis, and Visualization. This research has been funded in part and used resources of the Argonne Leadership Computing Facility at Argonne National Laboratory, which is supported by the Office of Science of the U.S. Department of Energy under contract DE-AC02-06CH11357.

The authors would like to thank Halman Sabib, Katrin Heitmann, and the HACC team for providing the datasets used in this work.

References

[AGL05] AHRENS J., GEVECI B., LAW C.: Paraview: An end-user tool for large data visualization. *The Visualization Handbook 717* (2005), 731. 2

[BCH12] BETHEL E. W., CHILDS H., HANSEN C.: *High Performance Visualization: Enabling Extreme-Scale Scientific Insight*. CRC Press, 2012. 3

[BFV*14] BUI H., FINKEL H., VISHWANATH V., HABIB S., HEITMANN K., LEIGH J., PAPKA M., HARMS K.: Scalable parallel i/o on a blue gene/q supercomputer using compression, topology-aware data aggregation, and subfiling. In *Parallel, Distributed and Network-Based Processing (PDP), 2014 22nd Euromicro International Conference on* (2014), IEEE, pp. 107–111. 3

[BGZ12] BÉDORF J., GABUROV E., ZWART S. P.: A sparse octree gravitational n-body code that runs entirely on the gpu processor. *Journal of Computational Physics 231*, 7 (2012), 2825–2839. 3, 5

[CK10] CONNOR M., KUMAR P.: Fast construction of k-nearest neighbor graphs for point clouds. *Visualization and Computer Graphics, IEEE Transactions on 16*, 4 (2010), 599–608. 3, 5

[CUD] CUDA Code Samples. https://developer.nvidia.com/cuda-code-samples. Accessed: 2015-02-08. 4

[EQU] Equalizer. http://www.equalizergraphics.com/. Accessed: 2015-04-08. 2

[FSW09] FRAEDRICH R., SCHNEIDER J., WESTERMANN R.: Exploring the millennium run-scalable rendering of large-scale cosmological datasets. *Visualization and Computer Graphics, IEEE Transactions on 15*, 6 (2009), 1251–1258. 2

[GEM*13] GOSWAMI P., EROL F., MUKHI R., PAJAROLA R., GOBBETTI E.: An efficient multi-resolution framework for high quality interactive rendering of massive point clouds using multi-way kd-trees. *The Visual Computer 29*, 1 (2013), 69–83. 2, 4

[GSGP06] GRIBBLE C. P., STEPHENS A. J., GUILKEY J. E., PARKER S. G.: Visualizing particle-based simulation datasets on the desktop. In *British HCI 2006 Workshop on Combining Visualization and Interaction to Facilitate Scientific Exploration and Discovery* (2006), pp. 1–8. 2

[HIO*11] HERELD M., INSLEY J. A., OLSON E. C., PAPKA M. E., VISHWANATH V., NORMAN M. L., WAGNER R.: Exploring large data over wide area networks. In *Large Data Analysis and Visualization (LDAV), 2011 IEEE Symposium on* (2011), IEEE, pp. 133–134. 3

[HMF*13] HABIB S., MOROZOV V., FRONTIERE N., FINKEL H., POPE A., HEITMANN K.: Hacc: extreme scaling and performance across diverse architectures. In *Proceedings of SC13: International Conference for High Performance Computing, Networking, Storage and Analysis* (2013), ACM, p. 6. 1

[KAH07] KÄHLER R., ABEL T., HEGE H.-C.: Simultaneous gpu-assisted raycasting of unstructured point sets and volumetric grid data. In *Proceedings of the Sixth Eurographics/Ieee VGTC conference on Volume Graphics* (2007), Eurographics Association, pp. 49–56. 2

[KBK08] KREYLOS O., BAWDEN G. W., KELLOGG L. H.: Immersive visualization and analysis of lidar data. In *Advances in visual computing*. Springer, 2008, pp. 846–855. 2

[LMC04] LIANG K., MONGER P., COUCHMAN H.: Interactive parallel visualization of large particle datasets. In *Proceedings of the 5th Eurographics conference on Parallel Graphics and Visualization* (2004), Eurographics Association, pp. 111–118. 2

[LZC14] LI E., ZHANG X., CHEN Y.: Sampling and surface reconstruction of large scale point cloud. In *Proceedings of the 13th ACM SIGGRAPH International Conference on Virtual-Reality Continuum and its Applications in Industry* (2014), ACM, pp. 35–41. 3

[MVA] MVAPICH: MPI over InfiniBand, 10GigE/iWARP and RoCE, Network-Based Computing Laboratory, Department of Computer Science and Engineering, Ohio State University. http://mvapich.cse.ohio-state.edu/. Accessed: 2014-05-25. 6

[NHP07] NYLAND L., HARRIS M., PRINS J.: Fast n-body simulation with cuda. *GPU gems 3*, 1 (2007), 677–696. 4

[NJB07] NAVRÁTIL P. A., JOHNSON J. L., BROMM V.: Visualization of cosmological particle-based datasets. *Visualization and Computer Graphics, IEEE Transactions on 13*, 6 (2007), 1712–1718. 2

[PF03] PASCUCCI V., FRANK R. J.: Hierarchical indexing for out-of-core access to multi-resolution data. In *Hierarchical and Geometrical Methods in Scientific Visualization*. Springer, 2003, pp. 225–241. 2, 3, 4, 5

[PNS*11] PDELLEPIANE M., NICCOLUCCI F., SERNA S. P., RUSHMEIER H., VAN GOOL L.: Real-time rendering of massive unstructured raw point clouds using screen-space operators. In *Proceedings of the 12th International conference on Virtual Reality, Archaeology and Cultural Heritage* (2011), Eurographics Association, pp. 105–112. 2

[RGD*14] RIVI M., GHELLER C., DYKES T., KROKOS M., DOLAG K.: Gpu accelerated particle visualization with splotch. *Astronomy and Computing 5*, 0 (2014), 9 – 18. 2

[RHI*14] RIZZI S., HERELD M., INSLEY J., PAPKA M. E., URAM T., VISHWANATH V.: Performance modeling of vl3 volume rendering on gpu-based clusters. In *EGPGV14: Eurographics Symposium on Parallel Graphics and Visualization* (2014), Eurographics Association. 3

[SMKK13] SAKAMOTO N., MAEDA N., KAWAMURA T., KOYAMADA K.: High-quality particle-based volume rendering for large-scale unstructured volume datasets. *Journal of Visualization 16*, 2 (2013), 153–162. 3

[SNKT07] SAKAMOTO N., NONAKA J., KOYAMADA K., TANAKA S.: Particle-based volume rendering. In *Visualization, 2007. APVIS'07. 2007 6th International Asia-Pacific Symposium on* (2007), IEEE, pp. 129–132. 2

[STA] The Simple, Tbread-safe Approximate Nearest Neighbor (STANN) C++ Library. https://sites.google.com/a/compgeom.com/stann/. Accessed: 2015-02-08. 3, 5

[SWH13] SELLERS G. M., WRIGHT R. S., HAEMEL N.: *OpenGL SuperBible: comprehensive tutorial and reference*, 6th ed. Addison-Wesley Professional, 2013. 4

[WAF*11] WOODRING J., AHRENS J., FIGG J., WENDELBERGER J., HABIB S., HEITMANN K.: In-situ sampling of a large-scale particle simulation for interactive visualization and analysis. In *Computer Graphics Forum* (2011), vol. 30, Wiley Online Library, pp. 1151–1160. 2, 4

[WHA*11] WOODRING J., HEITMANN K., AHRENS J., FASEL P., HSU C.-H., HABIB S., POPE A.: Analyzing and visualizing cosmological simulations with paraview. *The Astrophysical Journal Supplement Series 195*, 1 (2011), 11. 2, 3

[YWG*10] YU H., WANG C., GROUT R. W., CHEN J. H., MA K.-L.: In situ visualization for large-scale combustion simulations. *IEEE Computer Graphics and Applications 30*, 3 (2010), 45–57. 2